CYCLONE: THE FIRST INTEGRATED TIMING AND POWER ENGINE FOR ASYNCHRONOUS SYSTEMS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Wenmian Hua

May 2020

CYCLONE: THE FIRST INTEGRATED TIMING AND POWER ENGINE FOR ASYNCHRONOUS SYSTEMS

Wenmian Hua, Ph.D.

Cornell University 2020

Asynchronous circuits have potential advantages of higher speed and lower power consumption compared to their synchronous counterparts, but their poor CAD support is a major issue limiting adoption. A key component of any CAD flow for chip design is timing analysis, and the timing analysis problem for asynchronous logic is much more complicated than that for synchronous logic.

Previous work on timing showed that such systems under AND-causality and fixed delay exhibit periodicity properties. In this thesis, we first give a graph-based proof of the exact periodicity properties of such systems. Our result under weaker assumption of system connectivity, provides a theoretical foundation for these properties to be applied and exploited in more general asynchronous circuits containing bundled data logic or synchronous components.

Based on the theoretical results, we present the first known integrated timing and power analysis engine capable of handling large asynchronous timing circuits. For timing, we introduce the notion of performance and correctness slack for asynchronous circuits; for power, we compute both the static and dynamic components. We provide a hierarchical approach to constructing the event-dependency graph, and use the Galois framework for the parallelization to achieve fast runtime. The net result is Cyclone, a fast and accurate engine for both static timing and power analysis of asynchronous circuits.

Along the way, we also explore the problem of cycle time of min-max system,

one of the important algorithmic problems that we will encounter when extending our timing analysis to more general systems that contain OR-causality. We provide a new algorithm to solve this problem and prove its correctness. We also show that the algorithm runs reasonably fast in practice.

BIOGRAPHICAL SKETCH

Wenmian Hua was born in Xuzhou, Jiangsu, China, and he grew up and lived there until the age of 18. Wenmian graduated from Number 1. Middle School of Xuzhou, and attended University of Illinois at Urbana-Champaign in 2010, where he worked on research in electronic design automation under the guidance of Professor Martin D.F. Wong, and research in number-theoretic random walks under the guidance of Professor A.J. Hildebrand. He graduated with a Bachelor's Degree in Electrical and Computer Engineering, and a Bachelor's Degree in Mathematics in May 2014.

Wenmian was enrolled in the Ph.D. program in Electrical and Computer Engineering at Cornell University since August 2014, under the guidance of Professor Rajit Manohar and Professor Zhiru Zhang. He has been a visiting scholar at Yale University since August 2017 as well, followed his thesis advisor, Professor Rajit Manohar. His current research interests include electronic design automation, asynchronous very large scale integration design, design and analysis of algorithms, and graph theory.

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor, Professor Rajit Manohar for his continuous support on my Ph.D. study and research. His immense knowledge, ideas, and encouragement allowed me to explore, and finally achieve good results on the challenging research problems presented in this thesis. Also, I would like to express my appreciation to rest of my committee: Professor Christoph Studer and Professor Zhiru Zhang for their encouragement on my research, and their insightful comments on my presentation which helped me to further widen my knowledge in the related research areas.

I would also like to thank my collaborator Yi-Shan Lu and his advisor Professor Keshav Pingali for their great support on Galois framework and fast implementation of the algorithms. It was also a great pleasure to collaborate with my labmates Samira Ataei, Edward Bingham, Ruslan Dashkin and Yihang Yang on several research projects.

My sincere thanks go to all of my friends. In particular, I would like to thank Jiushi Bao, Lin Chen, Siyuan Dong, Yuhao Du, Rong Huang, Wentian Huang, Siyuan Gao, Naiyi Li, Fengjiao Liu, Zhan Liu, Xinxin Nie, Linbo Shao, Lei Shi, Xiaozhong Sun, Danyang Wang, Wen Wang, Xiang Wu, Danyi Xiong, Wenjie Xiong, Haotian Zhang, MingRui Zhang, Qiuyuan Zhang, Yihan Zhou, Yuecheng Zhou for helping me deal with difficulties in my graduate life.

Last but not least, my appreciation goes to my parents Ya Zuo and Youzhu Hua, for their great support and professional advices on my career path.

The work is supported in part by the NSF Grant 1419949 and 1420026, and DARPA FA8650-18-2-7850 and it is in collaboration with Professor Keshav Pingali's research group in University of Texas, Austin. Some of the contents presented in this thesis (including the abstract) are based on our work [21, 20].

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTIONS

1.1 Asynchronous Circuits

Circuit design styles can be roughly classified into two categories: synchronous circuits and asynchronous circuits.[1] In synchronous circuits, changes of states of memory components are synchronous to the global clock signal. In asynchronous circuits, there are no such global clocks, and they often use request and acknowledgement signals that indicate completions of instructions and operations between local modules and components. Asynchronous circuits have many potential advantages over synchronous circuits, including higher speed, lower power consumptions, and robustness to process and environmental variations. Most recently, researchers have shown the benefits of asynchronous circuits in the context of field-programmable gate arrays [26] and neuromorphic computing [46, 1]. Unfortunately, researchers interested in evaluating or adopting asynchronous circuits are immediately confronted with the issue of the lack of design support by mainstream commercial flows.

1.2 Asynchronous Static Timing Analysis

A key component of any CAD flow for chip design is timing analysis, and the timing analysis problem for asynchronous logic is quite different from its synchronous counterpart. As is shown in Figure 1.1, timing analysis of synchronous circuits can be effectively done by analyzing acyclic regions of combinational logic separated

[1] This chapter is partially based on the "Introduction" sections in our work [21, 20].

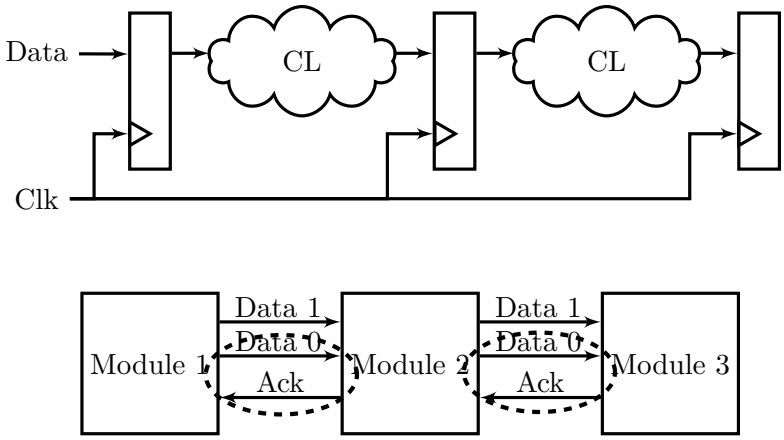

Figure 1.1: Synchronous vs asynchronous circuit

by clocked state-holding elements. Max-delay constraints (setup time) determine the performance of the circuit, and min-delay constraints (hold time) determine if the circuit will operate correctly. In asynchronous circuits, many common asynchronous logic gates (the C-element is an obvious example) are inherently state-holding. The time of each signal transition depends on that of many other signal transitions, and the timing behaviors of asynchronous circuits must be modeled by examining cyclic paths of timing dependencies. Thus, traditional timing analysis methods and tools for synchronous circuits cannot be easily adapted for this purpose.

The problem of timing analysis for asynchronous circuits has been explored in many previous papers. One approach is to adapt commercial synchronous static timing analysis tools for use with asynchronous designs as in [13]. This approach involves manually (or scripted) removal of timing arcs so that the resulting timing graph is acyclic. The actual performance is then calculated by combining timing paths [40]. To be accurate, this approach requires that the designer knows the cyclic critical path in the asynchronous circuit—which will be very challenging and time consuming when the circuit is complex or generated by a synthesis procedure.

Another approach is to directly build and analyze timing graph models for asynchronous circuits. Petri nets [43], marked graphs [8], and repetitive event-rule (*RER*) systems [5] (and its extended version: extended repetitive event-rule (*XRER*) system [28]) are several representative models for asynchronous timing analysis. Properties of these models and timing of asynchronous circuits have been well studied in both mathematics (e.g. [16, 15, 12, 19]) and engineering literature [5, 22, 28, 38, 27, 44], and it has been established that asynchronous systems with AND-causality, fixed delays and strong connectivity assumption will eventually show periodic behavior. Maximum cycle ratio algorithms, surveyed by Dasdan [10], can be used to (i) characterize the performance of circuits assuming AND-causality and fixed delays, and (ii) derive performance bound for data-dependent timing analysis [45].

From perspectives of theories, most of the previous performance analysis of AND-causal asynchronous systems was approximate or asymptotic, so it is precluded from analyzing timing constraints required for correctness. Hulgaard [22] proved a stronger result that such circuits will eventually become periodic and some of the recent papers on Max-Plus Algebra [22, 19, 7, 51, 39] further showed that the corresponding system associated to Max-Plus matrix will be periodic after finite transience and gave bounds on this interval. Most of these theories are based on the assumption that the system is strongly connected, and they are not applicable to the system containing both synchronous and asynchronous logic (because there is no path from the circuit to the external clock input – a violation of the strong connectivity requirement). Furthermore, a simple data FIFO implemented with bundled-data logic is also an example of an asynchronous circuit with more than one strongly connected component, and such a limitation also precludes these theories to be directly applicable to analysis of such systems. The work [22] actually

proposed a modification of the TSE algorithm in certain types of non-strongly connected systems. The modification already relied on the empirical observation that often such systems also exhibit periodicity, and their work applied the modified algorithm to the timing verification of "STARI" [22, 14], where the system is not strongly connected and contains both synchronous and asynchronous components.

From the perspectives of actual implementations of asynchronous timing analysis, none of previous efforts consider slew rates, and therefore, they cannot compute accurate gate delays and power using sophisticated delay models such as non-linear delay model (NLDM). Lack of conventions in timing report of asynchronous timing analysis (timing constraints, critical paths, performance, timing slack, .etc) as in synchronous counterpart is also a problem. There is no known framework in the literature, to our best knowledge, that is capable of analyzing timing of large asynchronous systems effectively and efficiently.

1.3 Cyclone: A Static Timing and Power Engine for Asynchronous Circuit

In this thesis, We adopt the *RER* system model introduced first by Burns [5]. We first provide a graph-based rigorous proof of a stronger result on the timing analysis of *RER* systems, and therefore, the timing characteristics of asynchronous circuits. We establish that an *RER* system with and-causality and fixed delays always exhibits exact periodicity and can be exactly characterized by a periodic function after finite number of occurrences k of each transition. Our proof of system periodicity does not require the assumption that *RER* systems are strongly connected, which thus provides a theoretical foundation, for the exact periodicity

property to be applied and exploited in more general asynchronous circuits containing bundled data logic or synchronous components. Our results hold regardless of initial conditions and the nature of the cycles in the *RER* system, and it turns out that the time taken to reach periodic steady-state is quite short for many practical asynchronous circuit topologies. The graph-based proof also gives intuitions on an important concept: performance slack, which we will need in Cyclone timing engine.

Then, we address several major differences and difficulties involved in asynchronous timing and power engine based on the theoretical results. These include (i) cell characterization, (ii) timing graph construction, (iii) steady-state slew rate computation using NLDM in the presence of cycles of gates, (iv) the maximum cycle ratio algorithm. We also introduce the asynchronous analog of the notions of arrival time, required time, performance slack and correctness slack. After all the timing issues are solved, power analysis of asynchronous can be easily incorporated under the same framework. The net result is Cyclone, an implementation of a comprehensive asynchronous timing and power analysis engine. The engine takes as input an asynchronous circuit netlist, multiple cell libraries using NLDM in the Synopsys liberty (.lib) format, and timing constraints from desired asynchronous circuit family for correctness. It performs a multi-corner analysis and reports maximum cycle ratio, power consumption, and two types of timing slacks: (i) the performance slack of each gate, which determines how much a gate can be slowed down without impacting the performance of the asynchronous circuit; and (ii) the correctness slack of each timing constraint, showing any violations of timing requirements or the margin that remains. We parallelize Cyclone using the Galois framework, and achieve up to 18.76× speedup for performance analysis compared to the sequential runs. We also plan an open-source release of Cyclone

to the community.

1.4 Metastability-Free Interface between Synchronous and Asynchronous Logic

In this thesis, we will also conclude, based on the theories we proved, that if the primary inputs and outputs of an asynchronous circuit are placed in environment where all the inputs are computed in a single clock domain that is slower than the natural frequency of the asynchronous logic, all the outputs of the asynchronous circuit will be synchronous with respect to the same clock after a finite initial time interval. Hence, we can build metastability-free interfaces between synchronous and asynchronous logic. We will provide an example of one such interface in Chapter 3.

1.5 A Fast Algorithm on Cycle Time of Min-Max Systems

Our work opens doors for lots of interesting future research. The current implementation of Cyclone timing and power engine assumes AND-causality, which gives conservative timing and power analysis for asynchronous circuits. The actual circuits can have both AND-causality and OR-causality, and Lee first introduced extended repetitive event-rule (*XRER*) system in [28] for asynchronous timing analysis that considers certain forms of OR-causality in the circuit. The papers [16, 15] also studied the properties of min-max functions, a closely related problem in mathematics literature. It turns out that the *XRER* also exhibits periodicity property [28, 16, 15]. As we mentioned that maximum cycle ratio char-

acterizes the cycle time of the *RER* system, and thus the performance of circuits with AND-causality only, the analogy "min-max cycle time" will characterize the cycle time of the *XRER* system, and thus the performance of more general circuits, with both AND-causality and OR-causality. Unfortunately, unlike maximum cycle ratio problem, where there are efficient algorithms surveyed by Dasdan [10], there is no known polynomial algorithm that calculates min-max cycle time of *XRER* system.

The problem of min-max cycle time of *XRER* systems extends far beyond the asynchronous timing analysis. With slightly more work, one can show that a restricted version of this problem is closely related to the problem of solving mean payoff games (MPGs) [42, 11] and cyclic games [17, 49]. Mean payoff games play important roles in model checking and automata theory, and the decision version of this problem – deciding the winner in mean payoff games is shown to be in NP $\cap$ Co-NP [54], and more precisely, UP $\cap$ Co-UP [23]. As a result, the problem is very unlikely to be an NP-Complete problem, because otherwise, NP = Co-NP, which will be an astonishing result. Indeed, the problem is also shown to be solvable in pseudopolynomial time [54, 4] and subexponential time [3]. A recent breakthrough on parity games, (a problem that is polynomial-time reducible to the MPGs and thus "slightly easier"), shows that parity games can be solved in quasi-polynomial time [6]. However, after decades of research, there is no known polynomial time algorithm for any of these problems. There are very few known problems sharing similar time complexity status, such as the decision version of integer factorization problem, which is also in NP $\cap$ Co-NP, UP $\cap$ Co-UP, solvable in pseudopolynomial time and subexponential time, but not known in P.

With these preliminaries on the problem of min-max cycle time of *XRER* sys-

tems, we will consider the following equivalent problem: cycle time of min-max system. In this thesis, we will first introduce the formulism of the min-max system and its cycle time, and then provide a brand new algorithm on finding the cycle time of min-max system. We will prove that the algorithm terminates and returns the correct results as desired, and show that the algorithm runs reasonably fast in practice.

1.6 Thesis Outline

This thesis is outlined as following:

- Chapter 2 introduces the *RER* system formulism and presents the main theoretical results of *RER* systems that we leverage for Cyclone analysis.
- Chapter 3 describes the Cyclone analysis flow, focusing on its major differences and difficulties compared to synchronous timing and power analysis.
- Chapter 4 includes our experimental results, which shows the effectiveness and efficiency our Cyclone timing and power engine.
- Chapter 5 introduces a fast algorithm on cycle time of min-max system, a theoretical foundation for a potential extension to our current Cyclone engine.
- Chapter 6 concludes this thesis and gives future work.

CHAPTER 2

THEORIES OF ASYNCHRONOUS STATIC TIMING ANALYSIS

In this chapter, we will present the core theoretical results, namely, the exact periodicity timing property of asynchronous systems under AND-causality and fixed delay assumptions.[1]

2.1 Preliminaries

We will first provide preliminaries for our work. Functionalities of asynchronous circuits will be primarily expressed using the *production rule set (PRS)* . We adopt the *repetitive event rule (RER) system* model introduced first by Burns [5] for static timing and power analysis on asynchronous systems. Some basic definitions from previous work [32, 5, 22] are repeated here for completeness.

Definition 1. *A production rule (PR) is a construct of the form* $G \mapsto t$*, where* t *is a simple assignment (a transition), and* G *is a boolean expression, which is called the guard of the PR. A gate with output* x*, pull-up network* B^+*, and pull-down network* B^- *corresponds to the production rules*

$$B^+ \mapsto x\uparrow \qquad (2.1)$$
$$B^- \mapsto x\downarrow .$$

A production rule set (PRS) is the concurrent composition of all the production rules in the set. A production rule set is used to describe a network of gates. For

[1] This chapter is based on our work in [21], and part of our work in [20].

example, a two-input NAND gate with input a and b and output c would be represented by the following production rules:

$$a\&b \mapsto c\downarrow \qquad (2.2)$$
$$\sim a|\sim b \mapsto c\uparrow .$$

Definition 2. *An event-rule (ER) system is a pair $\langle E, R\rangle$ where E is a set of events, and $R \subseteq E \times E \times \mathbb{R}_{\geq 0}$ is a set of rules that define timing constraints.*

A rule $r = (e_1, e_2, \alpha)$ is written as $e_1 \overset{\alpha}{\mapsto} e_2$, and indicates that the event e_2 cannot occur until α time units after e_1 occurs. e_1 is said to be the source of r (denoted src(r)), and e_2 is said to be the target of r (denoted tgt(r)). There could be events f for which there are no rules that constrain their timing. In other words, there is no $r \in R$ such that tgt(r) = f. These are referred to as initial events. For a non-terminating computation, the set of events E as well as the set of rules R are infinite.

Often systems with non-terminating computations of interest are repetitive. Hence, although the sets E and R are infinite, they can be represented as unfoldings of a compact, finite representation.

Definition 3. *A repetitive event-rule (RER) system is a pair $\langle E', R'\rangle$ where E' is a set of transitions, and $R' \subseteq E' \times E' \times \mathbb{R}_{\geq 0} \times \mathbb{N}$ is a set of rule templates.*[2] *The ER system corresponding to $\langle E', R'\rangle$ is given by $\langle E, R\rangle$, where $E = E' \times \mathbb{N}$ and R consists of rules of the form $\langle u, i\rangle \overset{\alpha}{\mapsto} \langle v, i + \epsilon\rangle$ where $r = (u, v, \alpha, \epsilon) \in R'$ and $i \in \mathbb{N}$. Here, the non-negative integer i is called the occurrence index and ϵ is called the occurrence index offset of r.*

[2] *RER* systems can also be augmented with a finite initial prefix to capture initialization in a circuit. We omit this technical detail from this thesis, as it has no impact on our analysis or main results.

Definition 4. *Given an ER system, the function $t : E \to \mathbb{R}_{\geq 0}$ is said to be a timing function for the ER system iff*

$$t(f) \geq t(e) + \alpha, \ \forall \ e \overset{\alpha}{\mapsto} f \in R. \tag{2.3}$$

Initial events can be assigned an arbitrary time by a timing function, as there are no rules that constrain them. Notice that there might be multiple rules which have the same target. In this case, each rule places a constraint on the time when the event f can happen and all constraints must be satisfied; hence, this representation captures AND-causality.

The *timing simulation* $\hat{t}$ is a special timing function which captures the notion of the actual time at which each event occurs.

Definition 5. *The timing simulation $\hat{t}$ is the timing function defined by*

$$\hat{t}(f) = \max\{\hat{t}(e) + \alpha \mid e \overset{\alpha}{\mapsto} f \in R\} \tag{2.4}$$

with the convention that the max operator evaluates to 0 when the set is empty (for initial events).

We assume that the timing simulation assigns time 0 to every initial event unless otherwise specified. Note that the definition of timing simulation is similar to "As Soon As Possible Scheduling." It is easily established that for any timing function t, $\hat{t}(e) \leq t(e)$ for all events e [5]. Given a potential timing function for the system, we introduce the notion of the slack of a rule in the *ER* system.

Definition 6. *Given a function $t : E \to \mathbb{R}_{\geq 0}$, the timing slack function $s_t : R \to \mathbb{R}$ induced by t is defined by:*

$$s_t(e \overset{\alpha}{\mapsto} f) = t(f) - (t(e) + \alpha). \tag{2.5}$$

The timing slack is another way to identify whether or not a given function t is in fact a valid timing function; t is a valid timing function iff the slack function is non-negative.

It is sometimes convenient to visualize event rule systems as well as repetitive event-rule systems in graphical format. For event rule systems, the acyclic *constraint graph* is used to visualize timing constraints [5]. The graph is constructed from the *ER* system: the set of vertices is the set of events E and a rule $e \overset{\alpha}{\mapsto} f \in R$ is displayed as an edge from e to f with an edge weight label corresponding to the delay α.

For repetitive event-rule systems, a similar visualization approach is used and is referred to as the *collapsed* constraint graph [5]. The set of vertices of the graph is the set of transitions E', and each rule template $\langle u, v, \alpha, \epsilon \rangle$ is displayed as an edge from u to v with an edge weight α. In addition, if $\epsilon > 0$, the edges are marked with lines (called "ticks") where the number of lines is the value of ϵ.

The following definitions deal with the graphical representation of the *ER* and *RER* systems.

Definition 7. *A path p of length n (denoted as $l(p) = n$) in an RER system is a list of arcs $(e_0, e_1, ..., e_n)$ where $(e_i, e_{i+1}, \alpha_i, \epsilon_i) \in R'$, for some $\alpha_i \in \mathbb{R}_{\geq 0}$ and $\epsilon_i \in \mathbb{N}$. We define the delay along the path p by $\delta(p) = \sum_i \alpha_i$ and the epsilon-sum $\epsilon(p) = \sum_i \epsilon_i$. A path is a simple cycle iff $e_0 = e_n$ and $e_i \neq e_j$ for all other pairs i, j with $i \neq j$.*

A path p of length n in an ER system is a list of arcs $(e_0, e_1, ..., e_n)$ where $e_i \overset{\alpha_i}{\mapsto} e_{i+1} \in R$, for some $\alpha_i \in \mathbb{R}_{\geq 0}$. We define the delay along the path p by $\delta(p) = \sum_i \alpha_i$. Given a timing function t, we can define the slack of the path $s_t(p)$ as

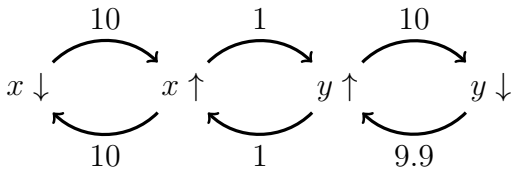

Figure 2.1: Example *RER* system

$\sum_i s_t(e_i \overset{\alpha_i}{\mapsto} e_{i+1})$.

We examine paths in two graphs (the constraint graph and the collapsed constraint graph) in the analysis below. To resolve ambiguity, we use the term *e-path* to refer to a path in the *ER* system rather than the *RER* system.

For example, consider the *RER* system $\langle E', R' \rangle$ where

$$\begin{aligned} E' &= \{x\uparrow, x\downarrow, y\uparrow, y\downarrow\} \\ R' &= \{(x\downarrow, x\uparrow, 10, 0), (x\uparrow, x\downarrow, 10, 1), (x\uparrow, y\uparrow, 1, 1), \\ &\qquad (y\uparrow, x\uparrow, 1, 0), (y\uparrow, y\downarrow, 10, 0), (y\downarrow, y\uparrow, 9.9, 1)\} \end{aligned}$$

The graphic representation of this *RER* system is shown in Figure 2.1 (similar to an example explored in [38]), with part of the *ER* system corresponding to this *RER* system shown in Figure 2.2. In this example, the initial events are $\langle y\uparrow, 0\rangle$, and $\langle x\downarrow, 0\rangle$. Thus $\hat{t}(\langle y\uparrow, 0\rangle) = 0$ and $\hat{t}(\langle x\downarrow, 0\rangle) = 0$. Consider the transition $x\uparrow$. We can set $t(x\uparrow, 0)$ to be any number greater than ten to obtain a valid timing function. However, by definition, only $\hat{t}(\langle x\uparrow, 0\rangle) = \max\{\hat{t}(\langle y\uparrow, 0\rangle + 1, \hat{t}(\langle x\downarrow, 0\rangle) + 10\} = \max\{0 + 1, 0 + 10\} = 10$ is the *timing simulation*.

While general *RER* systems allows arbitrary non-negative integers $\epsilon(r)$ for rules $r \in R'$, in practical cases, $\epsilon(r)$ is often 0 or 1. Furthermore, if a rule with delay α has $\epsilon(r) > 1$, it can also be represented by $\epsilon(r)$ consecutive directed edges, each with delay $\alpha/\epsilon(r)$ and $\epsilon = 1$. Thus, without loss of generality, we assume that $\epsilon(r) \in \{0, 1\}$, and thus $\max\{\epsilon(r)\} = 1$ for all of the rules in the *RER* systems

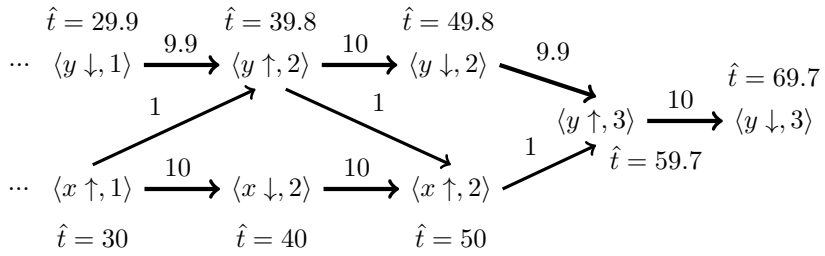

Figure 2.2: Part of the *ER* system corresponding to Figure 2.1

discussed in the rest of the chapter. Given this assumption, we have:

Corollary 1. *For any path or e-path p,*

$$l(p) \geq \epsilon(p) \tag{2.6}$$

A straightforward generalization of Definition 5 (timing simulations) from edges to paths is also stated without proof below.

Corollary 2. *If p is an e-path from e to f, $\hat{t}(f) \geq \hat{t}(e) + \delta(p)$.*

Definition 8. *An RER system is said to be valid if for each transition e in the RER system, there is a path p from e back to itself with $\epsilon(p) = 1$ and $\delta(p) > 0$.*

Theorem 1. *In a valid RER system,*

$$\forall n_e > 0 \ : \ \hat{t}(\langle e, n_e\rangle) > \hat{t}(\langle e, n_e - 1\rangle) \quad .$$

Proof. By assumption, for any event e there is a path p_e from e to itself with $\epsilon(p_e) = 1$. By construction of the *ER* system, there is a corresponding e-path p_{e,n_e} from $\langle e, n_e - 1\rangle$ to $\langle e, n_e\rangle$. By Corollary 2,

$$\hat{t}(\langle e, n_e\rangle) - \hat{t}(\langle e, n_e - 1\rangle) \geq \delta(p_{e,n_e}) > 0 \tag{2.7}$$

as desired. □

The assumption of being valid is a technical one. In any *RER* system that corresponds to a physical circuit, we will have $\hat{t}(\langle e, n_e \rangle) > \hat{t}(\langle e, n_e - 1 \rangle)$ since this says that the ${n_e}^{\text{th}}$ signal transition occurs after the $(n_e - 1)^{\text{th}}$ signal transition on the same variable. If the *RER* system does not satisfy the constraint of being valid but satisfies the inequality in Theorem 1 by virtue of corresponding to a circuit, we can compute the smallest gap between successive transitions of the same event and add a new "self loop" rule template with a delay that is smaller than the smallest gap. This will not change the timing simulation and result in an *RER* system that satisfies our technical requirement.

Definition 9. *An RER system is live if each simple cycle S in the system satisfies:* $\epsilon(S) \geq 1$.

All of the *RER* systems discussed in the following sections are assumed to be live and valid. If an *RER* system is not live, then all the transitions on cycles S with $\epsilon(S) = 0$ can be collapsed into a single transition without loss of generality [5].

Definition 10 (critical cycle)**.** *The cycle ratio of a cycle C is defined to be $\delta(C)/\epsilon(C)$. A critical cycle C of an RER system is a simple cycle with the maximum cycle ratio $p^\star$, given by $p^\star = \max_C \delta(C)/\epsilon(C)$ over all cycles C.*

Note that this definition of cycle ratio is different from that of the cycle mean, $p^\star = \max_C \delta(C)/l(C)$, which corresponds to a special case to our assumption, when every edge in the *RER* system has a tick, as in [16, 19, 7, 51, 39].

We assume that the maximum delay of a rule $r \in R'$ in the *RER* system is denoted as α_{max}. In most of previous work, the collapsed constraint graph is assumed to be strongly connected. In this thesis, we make a weaker assumption—namely, that for any transition f in the *RER* system, there exists a path p from a

transition on some critical cycle to f. Given this assumption and the definition of N, we have:

Corollary 3. *For any transition f in the RER system, there exists a path p from a transition e on some critical cycle C to f such that $0 \leq l(p) \leq N - l(C) \leq N - \epsilon(C) < N$. Notice that when f is on a critical cycle, $e = f$ and $l(p) = 0$.*

In what follows, we examine e-paths in the *ER* that correspond to cycles in the corresponding *RER* system. We sometimes refer to these e-paths as cycles—although the e-path in the *ER* system is in fact acyclic, it corresponds to a cycle of transitions in the corresponding *RER* system.

2.2 Proof of Periodicity

Given these technical preliminaries, we will prove our main technical result Theorem 3, the exact periodicity in an *RER* system with and-causality and fixed delay, based on graph and path properties, which can be stated as follows: There are integer constants $M \leq lcm\{1, 2, ..., N\} < 3^N$ [18] (M is defined later in the proof, which characterizes the period of the system, a similar parameter to $\epsilon^\star$ in [22] and the bound here corresponds to the worst case scenario and is very conservative) and

$$k^\star = \frac{\min\{2Np^\star + N\alpha^\star, 3Np^\star\}}{p^\star - p'} + (3M + 1)N$$

(p' is also defined later in the proof, which characterizes, and is at most, the second largest cycle mean of the system) such that for all transitions e and all integers $n \geq k^\star$,

$$\hat{t}(\langle e, n + M\rangle) - \hat{t}(\langle e, n\rangle) = Mp^\star.$$

The work [16] implied that the system will have the same (asymptotic) cycle time under our weaker assumption of system connectivity/topology. Papers [19, 7, 51, 39] gave bounds on transience of the corresponding matrix and system before exhibiting periodicity and pointed out that the period will be the same if it is strongly connected. Our work on exact periodicity properties of systems, however, makes weaker assumption about system connectivity/topology, and thus directly applies to more general classes of systems, especially the circuits/systems that we will discuss later. Furthermore, our graph-based proof gives intuitions and is closely related to an important concept: performance slack, which will be used in Cyclone timing and power engine in the next chapter. Finally, although our bounds on $k^\star$ are not directly comparable to those of [19, 7, 51, 39] because of the different assumption, definition, and proof strategy, parameters and terms in our formula still show some similar patterns and relations to previous results, which also reveals some intrinsic properties of these systems.

The rest of this section establishes this main technical result through a number of small steps. The next three lemmas are straightforward.

Lemma 1. *Let f be a non-initial event in an ER system. Then, $\exists$ at least one rule $r = e \overset{\alpha}{\mapsto} f \in R$ such that $s_{\hat{t}}(r) = 0$.*

Proof. If f is a non-initial event, then there is at least one rule that has f as a target. This means that in the equation for $\hat{t}(f)$ (definition 5), the set on the right-hand side of the equation is non-empty. Hence, there is some $r = e \overset{\alpha}{\mapsto} f \in R$ such that $\hat{t}(f) = \hat{t}(e) + \alpha$. By definition, $s_{\hat{t}}(r) = 0$. □

Corollary 4. *For every non-initial event, there exists a path from some initial event to it that has zero slack.*

As can be seen in Figure 2.2, from any event, we can always find a bolded path (zero slack rule) back to an initial event.

Consider the *ER* system that corresponds to an *RER* system. For any non-initial event $\langle f, n_f \rangle$, there is a zero slack e-path from some initial event to it (Corollary 4). Consider some event $\langle e, n_e \rangle$ on such a zero slack e-path that includes $\langle e, n_e \rangle \overset{\alpha_1}{\mapsto} \langle u_1, n_{u_1} \rangle ... \overset{\alpha_m}{\mapsto} \langle u_m, n_{u_m} \rangle \overset{\alpha_{m+1}}{\mapsto} \langle f, n_f \rangle$.

Lemma 2 (Zero Slack Path Timing Property)**.** *For all* $i \in \mathbb{Z}$ *where* $i + \min(n_e, n_f) \geq 0$*,*

$$\hat{t}(\langle f, n_f + i \rangle) - \hat{t}(\langle f, n_f \rangle) \geq \hat{t}(\langle e, n_e + i \rangle) - \hat{t}(\langle e, n_e \rangle). \tag{2.8}$$

Proof. By Corollary 2, given any e-path p from $\langle e, n_e + i \rangle$ to $\langle f, n_f + i \rangle$, we have:

$$\hat{t}(\langle f, n_f + i \rangle) - \hat{t}(\langle e, n_e + i \rangle) \geq \delta(p) \tag{2.9}$$

By construction, we are given a zero slack e-path

$$p_0 = (\langle e, n_e \rangle, \langle u_1, n_{u_1} \rangle, \ldots, \langle u_m, n_{u_m} \rangle, \langle f, n_f \rangle). \tag{2.10}$$

Therefore,

$$\delta(p_0) = \hat{t}(\langle f, n_f \rangle) - \hat{t}(\langle e, n_e \rangle). \tag{2.11}$$

Since the *ER* system is a generated from an *RER* system, there is an e-path p_i from $\langle e, n_e + i \rangle$ to $\langle f, n_f + i \rangle$ corresponding to p_0, and $\delta(p_0) = \delta(p_i)$. Hence,

$$\hat{t}(\langle f, n_f + i \rangle) - \hat{t}(\langle e, n_e + i \rangle) \geq \delta(p_i) = \delta(p_0). \tag{2.12}$$

Using (2.11) for $\delta(p_0)$ and re-arranging the inequality gives us the result. □

The proof of Lemma 2 relies on the repetitive nature of the *ER* system obtained from the *RER* system—if a zero-slack path exists between $\langle e, n_e \rangle$ and $\langle f, n_f \rangle$, then

the delay on that path is a bound on the minimum delay between $\langle e, n_e + i\rangle$ and $\langle f, n_f + i\rangle$ for any $i \geq -\min(n_e, n_f)$.

Lemma 3. *Let $k \geq 2$, k, l_1, ..., $l_k \in \mathbb{Z}$ and $k \nmid l_1, l_2, ..., l_k$, then there exists $m \neq n$, $m, n \leq k$, $m, n \in \mathbb{N}^+$ such that*

$$l_n - l_m \equiv 0 (mod\ k). \tag{2.13}$$

The proof has the same idea as in [19], and we repeat it here for completeness.

Proof. Since none of l_1, ..., l_k is divisible by k,

$$l_i \equiv j_i (\text{mod } k), \tag{2.14}$$

for all $i \leq k$, $i \in \mathbb{N}^+$, where $j_i \in \{1, 2, ..., k-1\}$. Since we have k j_i's, but only $k-1$ different elements, the pigeonhole principle indicates the existence of $i_1 \neq i_2$, $i_1, i_2 \leq k$, $i_1, i_2 \in \mathbb{N}^+$, such that $l_{i_1} \equiv l_{i_2} (\text{mod } k)$, which proves Lemma 3. □

With all the preparation Lemmas above, the following Lemma shows that the timing gaps of the occurrences of the same transition can be considered in blocks with finite sizes, where each block is bounded above by a quantity related to the critical cycle ratio.

Lemma 4 (Upper Bound)**.** *Given an event $\langle f, n_f\rangle$ and $I \in \mathbb{N}^+$ such that $n_f \geq NI$, there exists $q \leq N, q \in \mathbb{N}^+$ such that*

$$\hat{t}(\langle f, n_f\rangle) - \hat{t}(\langle f, n_f - qI\rangle) \leq qIp^\star, \tag{2.15}$$

where $p^\star$ is the critical cycle ratio (Definition 10).

Proof. By Corollary 4, there always exists a zero slack e-path from some initial event $\langle g, 0\rangle$ to the non-initial event $\langle f, n_f\rangle$. By finding back zero slack rule from $\langle f, n_f\rangle$, we will obtain such a path.

If $n_f \geq NI$, then this zero slack e-path has at least NI edges (because $\max\{\epsilon(r)\} = 1$). We only have N unique transitions in the *RER* system, so some transition (denoted as e) must appear at least $\frac{NI}{N} + 1 = I + 1$ times on this e-path. Consider the $I + 1$ occurrences of transition e that are closest to the initial event $\langle g, 0\rangle$ in time, denoted by events $\langle e, n_0\rangle$, $\langle e, n_1\rangle$, ..., $\langle e, n_I\rangle$, where $0 \leq n_0 < ... < n_I \leq NI$. (Notice that e might occur more than $I + 1$ times, and we only choose the $I + 1$ occurrences with the smallest n_i's).

We can decompose the large cycle ($\langle e, n_0\rangle$, ..., $\langle e, n_1\rangle$, ..., $\langle e, n_I\rangle$) into I cycles ($\langle e, n_0\rangle$, ..., $\langle e, n_1\rangle$), ($\langle e, n_1\rangle$, ..., $\langle e, n_2\rangle$), ..., ($\langle e, n_{I-1}\rangle$, ..., $\langle e, n_I\rangle$). We denote these cycles as $C_1, ..., C_I$. Then by definition, we have that:

$$n_m - n_{m-1} = \epsilon(C_m) \tag{2.16}$$

$$\hat{t}(\langle e, n_m\rangle) - \hat{t}(\langle e, n_{m-1}\rangle) = \delta(C_m) \tag{2.17}$$

for all $m \leq I, m \in \mathbb{N}^+$.

If I is not divisible by any $l_i's$, then $I \geq 2$, and we can apply Lemma 3 with $k = I$ and $l_i = n_i - n_0$ for $i \leq I$, $i \in \mathbb{N}^+$. Thus, there must exist $a < b \leq I$, $a, b \in \mathbb{N}^+$ such that $I \mid l_b - l_a = n_b - n_a$. If $I \mid l_i$ for some $i \leq I$, $i \in \mathbb{N}^+$, then $I \mid n_i - n_0$. We conclude that in either case, there must exist $a < b \leq I$, $a, b \in \mathbb{N}$ such that $I \mid n_b - n_a$.

Let $n_b - n_a = qI = \sum_{i=a+1}^{b} \epsilon(C_i)$ where $q \leq N$, $q \in \mathbb{N}^+$. Notice that we can further decompose the cycles $C_{a+1}, ..., C_b$ into simple cycles S_i's without changing the total weight and epsilon-sum of the cycles. By (the case when $i < 0$) in

Lemma 2, we have

$$\begin{aligned}
&\hat{t}(\langle f, n_f\rangle) - \hat{t}(\langle f, n_f - qI\rangle) \\
&\leq \hat{t}(\langle e, n_b\rangle) - \hat{t}(\langle e, n_b - qI\rangle) \\
&= qI\frac{\sum_{i=a+1}^{b}\ \delta(C_i)}{\sum_{i=a+1}^{b}\epsilon(C_i)} \\
&= qI\frac{\sum_{i}\ \delta(S_i)}{\sum_{i}\ \epsilon(S_i)} \\
&\leq qI\max\{\frac{\delta(S_1)}{\epsilon(S_1)}, \frac{\delta(S_2)}{\epsilon(S_2)}, ...\} \\
&\leq qIp^{\star}
\end{aligned} \tag{2.18}$$

as desired. Note that if the q in (2.15) is found by exactly carrying the steps described above, the equality in (2.15) holds only when all of the equalities in (2.18) hold, which further implies that

$$\frac{\delta(S_1)}{\epsilon(S_1)} = \frac{\delta(S_2)}{\epsilon(S_2)} = ... = \max\{\frac{\delta(S_1)}{\epsilon(S_1)}, \frac{\delta(S_2)}{\epsilon(S_2)}, ...\} = p^{\star}. \tag{2.19}$$

That is, all of the simple cycles S_i's are critical cycles. □

Remark 1. *Our proof of this Lemma also has some similar structures to that of Lemma 3 in [19] when we set up a comparison between timing simulations and the critical cycle ratio based on our Lemma 3. We introduced the parameter I in Lemma 4 as we will need it in later parts of the proof. It is needed when we have to consider the case where an RER system has multiple critical cycles, and some cycles have $\epsilon(\cdot) > 1$. In those cases, the proof will need to examine timing gaps in blocks whose size is an integer multiple of I where I corresponds to the value of the epsilon-sum.*

Given an *RER* system, we can easily compute the timing simulation until every transition occurs at least n times, giving us $\hat{t}(\langle f, i\rangle)$ for all $f \in E'$ and all $0 \leq i \leq n$.

While computing these values of $\hat{t}$, we can record all rules that have zero slack. Note that there might be multiple rules with a given target that have zero slack, in which case we record just one of them.

Algorithm 1 Computing M and p' for periodicity proofs.

```
Initialize M = 1, p' = 0, n = 0,
while n < NM do
    n = NM
    ∀f ∈ E' compute t̂(⟨f, n⟩) recording slack zero rules
    for each node f ∈ E' do
        Find a zero slack e-path back from ⟨f, n⟩
        for each of ≥ M simple cycles G_i in the path do
            if δ(G_i)/ε(G_i) < p* then
                p' = max{p', δ(G_i)/ε(G_i)}
            else
                M = lcm{M, ε(G_i)}
            end if
        end for
    end for
end while
```

We can compute $p^\star$ using algorithms surveyed in [10], and we will discuss this in more details in the next chapter. In the following analysis, we also need to define two additional parameters, M and p', related to the least common multiple of number of the epsilon-sum of all the critical cycles, and the second largest cycle ratio, respectively, as shown in Algorithm 1. Note that the algorithm might seems expensive in the worst case, but the two quantities are only defined here to bound the time that it takes for the system to exhibit periodicity. The work [22] also claims without a proof an easy way to calculate M for practical applications, which will be discussed in the next chapter. Furthermore, we would also like to comment that the for the majority of practical systems, even the algorithm proposed here turns out to be much faster than its worst case performance at the first glance, because M is typically reasonably small.

Examining Algorithm 1, note that using the same reasoning used in Lemma 4,

a slack zero e-path from an initial event to $\langle f, NM\rangle$ must contain at least M cycles. In the algorithm, G_i denotes the simple cycles decomposed from these cycles. The algorithm terminates (when M is not increased by the for loop) because M is bounded from above by the least common multiple of the $\epsilon(C)$'s for all critical cycles C. Also, note that $\epsilon(G_i) \mid M$ for any critical cycle G_i and $\delta(G_i)/\epsilon(G_i) \leq p'$ for any non-critical G_i obtained while applying the algorithm above. The quantities M and p' appear many times in our equations and bounds, and we will give an intuition on the roles they play in our analysis later.

Suppose we pick an arbitrary event $\langle f, n_f\rangle$ in the *ER* system where $n_f \geq NM$, obtain a zero slack e-path from an initial event to it, and find the $q \leq N, q \in \mathbb{N}^+$ as in Lemma 4 such that

$$\hat{t}(\langle f, n_f\rangle) - \hat{t}(\langle f, n_f - qM\rangle) \leq qMp^\star. \tag{2.20}$$

Then, we also have the following corollary based on Lemma 4.

Corollary 5 (Lower Bound)**.** *If at least one of the simple cycles S_i (as in Lemma 4) is a critical cycle, then*

$$\hat{t}(\langle f, n_f + kM\rangle) - \hat{t}(\langle f, n_f\rangle) \geq kMp^\star \tag{2.21}$$

for all $k \in \mathbb{N}^+$.

Proof. Assume that some transition e on the zero slack e-path is on the critical cycle denoted S_x. By construction, $\epsilon(S_x) \mid M$. By Lemma 2,

$$\begin{aligned}\hat{t}(\langle f, n_f + kM\rangle) - \hat{t}(\langle f, n_f\rangle) &\geq \hat{t}(\langle e, n_e + kM\rangle) - \hat{t}(\langle e, n_e\rangle) \\ &\geq kMp^\star,\end{aligned}$$

where the last step follows from applying Corollary 2 to the e-path obtained from S_x repeated $M/\epsilon(S_x)$ times. □

Theorem 2 (Stability of Steady-State)**.** *For any transition f, if $n_f \geq NM$, and*

$$\hat{t}(\langle f, n_f - kM\rangle) - \hat{t}(\langle f, n_f - (k+1)M\rangle) \geq Mp^\star \tag{2.22}$$

for all $0 \leq k \leq N-1$, $k \in \mathbb{N}$, then

$$\hat{t}(\langle f, n_f + kM\rangle) - \hat{t}(\langle f, n_f + (k-1)M\rangle) = Mp^\star \tag{2.23}$$

for all $k \in \mathbb{N}^+$.

Proof. For the event $\langle f, n_f\rangle$, we follow the steps in the proof of Lemma 4 and obtain $q_1 \leq N, q_1 \in \mathbb{N}^+$ such that

$$\hat{t}(\langle f, n_f\rangle) - \hat{t}(\langle f, n_f - q_1 M\rangle) \leq q_1 Mp^\star. \tag{2.24}$$

Assumption (2.22) implies that (2.24) is an equality, and (2.22) is an equality for all $0 \leq k < q_1$. Hence, all the simple cycles S_i's from Lemma 4 are critical cycles. Applying Corollary 5 (with $k = 1$),

$$\hat{t}(\langle f, n_f + M\rangle) - \hat{t}(\langle f, n_f\rangle) \geq Mp^\star. \tag{2.25}$$

(2.25) extends (2.22) to $k = -1$. We now repeat the argument, starting from event $\langle f, n_f + M\rangle$ to obtain $q_2 \leq N, q_2 \in \mathbb{N}^+$ such that

$$\hat{t}(\langle f, n_f + M\rangle) - \hat{t}(\langle f, n_f + M - q_2 M\rangle) \leq q_2 Mp^\star. \tag{2.26}$$

The extended (2.22) forces the inequalities to be equalities. That is,

$$\hat{t}(\langle f, n_f + M\rangle) - \hat{t}(\langle f, n_f\rangle) = Mp^\star. \tag{2.27}$$

The result readily follows by a simple induction. □

Remark 2. *The paper [22] mentioned that, in strongly connected systems where cycles have more than one tick, timing values being determined by a critical cycle*

once does not necessarily mean that timing values have entered the periodic steady-state. Additional unfoldings might be necessary before the steady-state is reached, and the number is closely related to solutions to the Frobenius problem [50]. This theorem also reflects this point, and provides an alternative way to verify the timing periodicity (including in systems that are not strongly connected but satisfy our assumptions). In particular, if the periodicity property is maintained MN times, the system is guaranteed to be in the periodic steady-state.

Lemma 5 (Bound of Slack)**.** *Let $\langle f, n_f \rangle$ be some event in the ER system where $n_f \geq N$. Then, there exists some transition e on a critical cycle C such that (i) there is an e-path p with $0 \leq l(p) < N$ from some $\langle e, n_e \rangle$ to $\langle f, n_f \rangle$; and (ii) $s_{\hat{t}}(p) < \min\{Np^\star + N\alpha_{max}, 2Np^\star\}$, where α_{max} denotes the largest delay of any rule in this RER system as defined earlier. Notice that when f is on a critical cycle, $e = f$, $l(p) = 0$ and $s_{\hat{t}}(p) = 0$.*

Proof. We know that there exists a zero slack e-path p_0 from some initial event $\langle g, 0 \rangle$ to $\langle f, n_f \rangle$. p_0 can be decomposed into simple cycles S_i and a cycle-free path $p_1 = (g, e_1, ..., e_{l(p_1)-1}, f)$ with $0 \leq l(p_1) < N$ in the *RER* system. Note that p_0 corresponds to the path starting from $\langle g, 0 \rangle$, traversing some of the e_i's, then the cycles S_i, and then the rest of the e_i's until we reach $\langle f, n_f \rangle$. Therefore,

$$\hat{t}(\langle f, n_f \rangle) = \delta(p_0) = \sum_i \delta(S_i) + \delta(p_1). \tag{2.28}$$

By Corollary 3, we can find a path p_2 with $0 \leq l(p_2) \leq N - l(C) < N$ from the transition e to f. Set p to be the e-path corresponding to p_2. First notice that by

Corollary 1, $\epsilon(p_2) \leq l(p_2) \leq N - l(C) \leq N - \epsilon(C)$, then

$$\begin{aligned}
\hat{t}(\langle e, n_e \rangle) &\geq \left\lfloor \frac{n_e}{\epsilon(C)} \right\rfloor \delta(C) \\
&= \left\lfloor \frac{n_f - \epsilon(p)}{\epsilon(C)} \right\rfloor \delta(C) \\
&\geq \left\lfloor \frac{n_f - (N - \epsilon(C))}{\epsilon(C)} \right\rfloor \delta(C) \\
&> \left(\frac{n_f - (N - \epsilon(C))}{\epsilon(C)} - 1 \right) \delta(C) \\
&= \frac{n_f - N}{\epsilon(C)} \delta(C) \\
&= (n_f - N) p^\star.
\end{aligned} \tag{2.29}$$

Thus, we have

$$\begin{aligned}
s_{\hat{t}}(p) &= \hat{t}(\langle f, n_f \rangle) - \hat{t}(\langle e, n_e \rangle) - \delta(p) \\
&= \sum_i \delta(S_i) + \delta(p_1) - \hat{t}(\langle e, n_e \rangle) - \delta(p) \\
&\leq \sum_i \delta(S_i) + \delta(p_1) - \hat{t}(\langle e, n_e \rangle) \\
&< \frac{\sum_i \delta(S_i)}{\sum_i \epsilon(S_i)} \sum_i \epsilon(S_i) + \delta(p_1) - (n_f - N) p^\star \\
&< n_f p^\star + N \alpha_{max} - (n_f - N) p^\star \\
&= N p^\star + N \alpha_{max}.
\end{aligned} \tag{2.30}$$

Similarly, by Corollary 3, we can find a path p_3 with length $0 \leq l(p_3) \leq N - l(C)$ from the transition e to g. Combining p_3 with p_1, we construct path $(e, \ldots, g, \ldots, f)$ in the *RER* system and decompose it into simple cycles G_i's and a cycle-free path $p_4 = (e, e_1, ..., e_{l(p_4)-1}, f)$. Together with Corollary 1, we have $0 \leq \epsilon(p_4) \leq l(p_4) < N$. We set p to be the e-path corresponding to p_4 that ends in event $\langle f, n_f \rangle$ and starts with event $\langle e, n_e \rangle$. Thus,

$$\delta(p_1) + \delta(p_3) = \sum_i \delta(G_i) + \delta(p). \tag{2.31}$$

Also, notice that

$$\begin{aligned}
&\sum_i \epsilon(S_i) + \sum_i \epsilon(G_i) + \epsilon(p) \qquad (2.32)\\
&= \epsilon(p_0) + \epsilon(p_3)\\
&\leq n_f + N - l(C)\\
&\leq n_f + N - \epsilon(C).
\end{aligned}$$

Then, we have

$$\begin{aligned}
s_{\hat{t}}(p) &= \hat{t}(\langle f, n_f\rangle) - \hat{t}(\langle e, n_e\rangle) - \delta(p) \qquad (2.33)\\
&= \sum_i \delta(S_i) + \sum_i \delta(G_i) - \delta(p_3) - \hat{t}(\langle e, n_e\rangle)\\
&\leq (n_f + N - \epsilon(C))\frac{\sum_i \delta(S_i) + \sum_i \delta(G_i)}{\sum_i \epsilon(S_i) + \sum_i \epsilon(G_i)} - \hat{t}(\langle e, n_e\rangle)\\
&\leq (n_f + N - \epsilon(C))p^\star - \left\lfloor \frac{n_e}{\epsilon(C)} \right\rfloor \delta(C)\\
&= (n_f + N)p^\star - \delta(C) - \left\lfloor \frac{n_f - \epsilon(p)}{\epsilon(C)} \right\rfloor \delta(C)\\
&< (n_f + N)p^\star - \delta(C) - \left\lfloor \frac{n_f - N}{\epsilon(C)} \right\rfloor \delta(C)\\
&< (n_f + N)p^\star - \delta(C) - \left(\frac{n_f - N}{\epsilon(C)} - 1 \right) \delta(C)\\
&= (n_f + N)p^\star - (n_f - N)p^\star\\
&= 2Np^\star
\end{aligned}$$

as desired. □

Theorem 3 (System Periodicity)**.** *Let* $n \geq k^\star = L + NM$ *where*

$$L = \frac{\min\{2Np^\star + N\alpha^\star, 3Np^\star\}}{p^\star - p'} + (2M + 1)N, \qquad (2.34)$$

Then, for all transitions f*,*

$$\hat{t}(\langle f, n + M\rangle) - \hat{t}(\langle f, n\rangle) = Mp^\star. \qquad (2.35)$$

Proof. For any $n \geq L$ and transition f, there exists a zero-slack e-path from some initial event to $\langle f, n\rangle$ and find positive integer $q_1 \leq N$ as in Lemma 4 such that

$$\hat{t}(\langle f, n\rangle) - \hat{t}(\langle f, n - q_1 M\rangle) \leq q_1 M p^\star. \tag{2.36}$$

We use S_i to denote the ith decomposed simple cycle as in Lemma 4. There also exists a zero slack e-path from some initial event to $\langle f, n - q_1 M\rangle$, and we apply Lemma 4 again to obtain $q_2 \leq N$ such that

$$\hat{t}(\langle f, n - q_1 M\rangle) - \hat{t}(\langle f, n - q_1 M - q_2 M\rangle) \leq q_2 M p^\star. \tag{2.37}$$

We use S'_i to denote the ith decomposed simple cycle from Lemma 4. Define $K_k = \sum_{i=1}^{k} q_i M$, where we repeat the aforementioned procedure k times. k is selected so that

$$K_k > n - N(M+1) \quad \text{and} \quad K_{k-1} \leq n - N(M+1). \tag{2.38}$$

Since the change from K_{k-1} to K_k is $q_k M$ and $q_i \leq N$ for all i, this also means $K_k \leq n - N$. Therefore,

$$\begin{aligned}
&(K_k - q_1 M)(p^\star - p') \\
&\geq K_k(p^\star - p') - NM(p^\star - p') \\
&> (n - N(M+1))(p^\star - p') - NM(p^\star - p') \\
&\geq (L - N(M+1))(p^\star - p') - NM(p^\star - p') \\
&= \min\{2Np^\star + N\alpha^\star, 3Np^\star\}.
\end{aligned} \tag{2.39}$$

Lemma 5 implies that there exists an e-path p_0 with $l(p_0) < N$ from some event $\langle e, n_e\rangle$ to $\langle f, n\rangle$, where e is on some critical cycle C. Thus,

$$n - n_e = \epsilon(p_0) \leq l(p_0) < N \quad \text{and} \quad n_e - K_k > 0 \tag{2.40}$$

by Corollary 1 and the earlier constraint on K_k. Corollary 2 implies that

$$\begin{aligned}\hat{t}(\langle e, n_e\rangle) - \hat{t}(\langle e, n_e - K_k\rangle) &\geq \left\lfloor \frac{K_k}{\epsilon(C)} \right\rfloor \delta(C) \\ &> \frac{K_k}{\epsilon(C)}\delta(C) - \delta(C) \\ &= p^\star K_k - \epsilon(C)p^\star \\ &\geq p^\star K_k - l(C)p^\star \\ &\geq p^\star K_k - Np^\star.\end{aligned} \tag{2.41}$$

Since $n - K_k \geq N$ again by earlier constraint on K_k, Lemma 5 also provides an e-path p_{-K_k} from $\langle e, n_e - K_k\rangle$ to $\langle f, n - K_k\rangle$ such that

$$s_{\hat{t}}(p_{-K_k}) < \min\{Np^\star + N\alpha^\star, 2Np^\star\}. \tag{2.42}$$

Assume, toward a contradiction, that the inequality in (2.36) is strict. We use the contrapositive of Corollary 5 (with $k := q_1$ and $n_f := n - q_1 M$) to conclude that none of the simple cycles S_i''s are critical cycles. Using the same argument as (2.18),

$$\begin{aligned}&\hat{t}(\langle f, n - q_1 M\rangle) - \hat{t}(\langle f, n - q_1 M - q_2 M\rangle) \\ &\leq q_2 M \frac{\sum_i \ \delta(S_i')}{\sum_i \ \epsilon(S_i')} \\ &\leq q_2 M p'.\end{aligned} \tag{2.43}$$

Since $n - K_{k-1} \geq (M+1)N > NM$, we can repeatedly apply this reasoning to obtain inequalities of the form (2.43). Telescoping them, and combining with (2.36), we get

$$\hat{t}(\langle f, n\rangle) - \hat{t}(\langle f, n - K_k\rangle) < q_1 M p^\star + (K_k - q_1 M)p'. \tag{2.44}$$

Let p_0 be the e-path corresponding to p_{-K_k} but from $\langle e, n_e\rangle$ to $\langle f, n\rangle$. Then, by

(2.41) and (2.44),

$$\begin{aligned} & s_{\hat{t}}(p_0) - s_{\hat{t}}(p_{-K_k}) \\ & = \hat{t}(\langle f, n\rangle) - \hat{t}(\langle e, n_e\rangle) - \hat{t}(\langle f, n-K_k\rangle) + \hat{t}(\langle e, n_e - K_k\rangle) \\ & < q_1 M p^\star + (K_k - q_1 M)p' - (p^\star K_k - N p^\star) \\ & = N p^\star - (K_k - q_1 M)(p^\star - p'). \end{aligned} \tag{2.45}$$

Together with (2.42) and (2.45),

$$\begin{aligned} s_{\hat{t}}(p_0) &< \min\{2Np^\star + N\alpha^\star, 3Np^\star\} \\ &\quad - (K_k - q_1 M)(p^\star - p') < 0. \end{aligned} \tag{2.46}$$

This is a contradiction, because slack value is non-negative. Thus, the equality in (2.36) must hold. Then, by Lemma 5, all of S_i's are critical. Thus, for all $n \geq L$, we must have

$$\hat{t}(\langle f, n+M\rangle) - \hat{t}(\langle f, n\rangle) \geq Mp^\star. \tag{2.47}$$

Theorem 2 then implies that for all $n \geq L + NM$,

$$\hat{t}(\langle f, n+M\rangle) - \hat{t}(\langle f, n\rangle) = Mp^\star \tag{2.48}$$

as desired. □

Remark 3. *Our proof based on graph and path properties (slack of paths) provides a different view from previous work on how the system enters the periodic steady-state with time period $Mp^\star$.*

Specifically, think of all the events in the system as partitioned into blocks, with block sizes equal to integer multiples of M but less than NM. If there is positive slack from $\langle e, n_e\rangle$ (e is on a critical cycle) to $\langle f, n\rangle$ (which is not in steady-state), then there is always a minimum reduction in slack—namely, $p^\star - p'$ after each block:

from $\langle e, n_e + k_1 M\rangle$ *to* $\langle f, n + k_1 M\rangle$, *from* $\langle e, n_e + k_1 M + k_2 M\rangle$ *to* $\langle f, n + k_1 M + k_2 M\rangle$ *etc. Since the slack on the path is finite by Lemma 5, there will be an upper bound on the time it takes for slack on the path to vanish. For example, in Figure 2.2,* $s_{\hat{t}}(\langle x\uparrow, 1\rangle \overset{1}{\mapsto} (\langle y\uparrow, 2\rangle) = 8.8$, *while* $s_{\hat{t}}(\langle x\uparrow, 2\rangle \overset{1}{\mapsto} (\langle y\uparrow, 3\rangle) = 8.7$. *After each block (block size is one in this simple case because each cycle has an epsilon-sum of one), there is at least* $10 - 9.9 = 0.1$ *reduction of slack on the path, because transition* $x\uparrow$ *is on the critical cycle and occurs with a slower rate. After the slack vanishes, the timing of occurrence of transition* f *will be determined by that of* e, *and thus determined by the critical cycle ratio, which further implies that the transition* f *will satisfy the property stated in the inequality 2.47. Theorem 2 then implies that if a transition satisfies this property for long enough time, it will always be in the steady-state.*

Remark 4 (Complexity)**.** *We know that* $\max\{\epsilon(C)\} = N$ *for any simple cycle in the RER system. In the worst case, where* $p^\star \approx p'$ *and* $M = lcm\{1, 2, \ldots, N\}$, *we know that* $M < 3^N$ *[18]. Thus, it can take very long time,* $O(\frac{N3^N}{p^\star - p'})$, *for the system to enter the steady-state. It justifies the intuition provided by the authors in [12] and [22], that it might take long for the system to become periodic if there exists a cycle with cycle ratio close to* $p^\star$. *Consider the example shown in Figure 2.1. This example takes a very long time to reach the steady-state because* $p^\star \approx p'$, *a similar example to the Figure 3 in [38]. There are only four transitions, but the system does not show periodicity (period 20) until after 3612 time units, when each transition occurs more than 180 times.*

Fortunately, the majority of practical systems exhibit periodicity very quickly, and the worst case described rarely occurs. Also, since the theoretical results consider the worst case in every single step of analysis, the bound is extremely loose in practice as well.

2.3 Timing Constraints

Given prior work, the key performance indicator for the asynchronous circuit is the value of $p^\star$, the maximum cycle ratio in the *RER* system. In addition, the periodicity result (Theorem 1) states that there is an integer M such that the circuit's timing is only periodic every M iterations. Hence, we track M different times for each transition to completely characterize the timing function $\hat{t}$.

For practical applications of our theory on asynchronous circuits in steady-state, it turns out that the M value defined in Algorithm 1 is too pessimistic, and the calculation is thus also unnecessarily expensive. The work [22] claims without a proof that if the system has nontrivial strongly connected components $G_0, G_1, ..., G_n (n \leq |E'|)$, then $M = lcm_i\{gcd\{\epsilon(C) : C \in G_i\}\}$, where gcd denotes the greatest common divisor.

In what follows, we will introduce some important timing concepts that we will use for our Cyclone engine in the next chapter. We assume that for every event in the system, there is a path from an event on a critical cycle to it, the condition needed for Theorem 1 to hold. Without loss of generality, Cyclone engine will focus on the system with only one nontrivial strongly connected component. Note that this will be equivalent to assuming there is a path from every event in a critical cycle to all other events in the system. Furthermore, in this case, a conservative M value will be the ϵ-sum of any critical cycle we find in the system. All of the following work can be readily generalized to more general systems with multiple nontrivial strongly connected components, and in which case, one can first run strongly connected component algorithm, followed by taking the least common multiple of M's of all those nontrivial strongly connected components as the final M value.

2.4 Arrival time, required time, and performance slack

We now introduce the notion of arrival time, required time, and performance slack for all the events in the asynchronous system. The *performance slack* of a signal transition is defined to be the amount that the transition can be slowed down without affecting the performance of the whole system. Increasing the delays of edges on a critical cycle further slows down the asynchronous circuit. Hence, any transition on a critical cycle has zero performance slack.

Since we know that there is a path from every event on a critical cycle to all other events in the *RER* system, we can compute the *arrival time* of all the other events in the system relative to the events on the critical cycle. We select one event on a critical cycle to have time $t = 0$ arbitrarily such that all the events have non-negative arrival times. All other event arrival times are determined by timing propagation similar to max-delay propagation in synchronous timing analysis. Note that we are *guaranteed* not to propagate through cycles, since we have already computed the arrival times for all the events on this critical cycle.

The required time can be computed in a manner analogous to the required time in synchronous logic, by working backward from the critical cycle. The initial condition used is that each event on the critical cycle has a required time that is equal to its arrival time.

Finally, we have M different performance slack values for each signal transition, obtained by subtracting the arrival time from the required time. As the circuit is static, the final slack value for a transition is given by the minimum across all M values. We report this value as the *performance slack* of a signal transition. This is the amount by which that signal transition could be delayed without impacting

the performance of the asynchronous circuit.

Remark 5. *Note that performance slack definition here (per signal transition or event) is slightly different from the slack in Definition 6, which is per rule. The intuition is that unlike the slack in Definition 6, the performance slack for every transition implicitly uses events on critical cycles of the circuit under steady-state as reference points.*

2.4.1 Correctness slack

While asynchronous circuits have been widely described as not having any "timing closure" problems, this is not entirely accurate. While asynchronous circuits can be designed to be robust to timing, they also have some timing constraints necessary for correct operation. It has been shown that asynchronous circuits that must operate correctly without any timing constraints are extremely limited in what they can compute [37, 34]. One of the mildest forms of timing assumptions is the isochronic fork [37], which requires that a wire branch is faster than an adversarial sequence of gates [33]. More generally, we express our timing constraints using a *timing fork*, borrowing a notion from timed distributed systems [9].

A *timing fork* $\langle r,i\rangle : \langle x,j\rangle < \langle y,k\rangle$ consists of vertex $\langle r,i\rangle$ (the "root"), and two paths in the *RER* system: $p_{\langle r,i\rangle,\langle x,j\rangle}$ from $\langle r,i\rangle$ to $\langle x,j\rangle$ and $p_{\langle r,i\rangle,\langle y,k\rangle}$ from $\langle r,i\rangle$ to $\langle y,k\rangle$, where $j,k \in \{i,i+1\}$. The timing fork requires that the correctness slack $sl = \min\{delay(p_{\langle r,i\rangle,\langle y,k\rangle})\} - \max\{delay(p_{\langle r,i\rangle,\langle x,j\rangle})\} \geq 0$. Note that a path might include the output transition of a gate, or might end at the input of a gate.

Timing forks are generated during logic synthesis, as different asynchronous logic families have different timing forks. Hence, we assume they are inputs to our

timing engine. In our evaluation section, we have isochronic fork timing constraints as well as the "bundling" timing constraint required for bundled-data asynchronous circuits to illustrate two different asynchronous circuit families that our engine can analyze.

We remark that the notion of a timing fork is similar to the concept - relative timing constraints [35] and generalizes the hold time requirement in synchronous logic. The hold time requirement corresponds to a timing fork that begins at the root of the common clock subtree for two flip-flops, with one path going through the launching flip-flop and combinational logic to the input of a capturing flip-flop, and the second path going directly to the clock input of the capturing flip-flop.

CHAPTER 3

APPLICATIONS OF THEORIES

Our result under weaker assumption provides a theoretical foundation, for the exact periodicity property to be applied and exploited in more general asynchronous circuits containing bundled data logic or synchronous components. In this chapter, we are going to show two main applications of our theories. We will first introduce Cyclone, which is to our best knowledge, the first comprehensive description and implementation of asynchronous static timing and power engine. Then, we will show that it is possible to interface asynchronous circuits to synchronous logic without metastability, provided all inputs to the asynchronous circuit are clocked.

3.1 Cyclone: An Open-Source Static Timing and Power Analysis Engine

With all of the preliminaries presented in sections II, we now describe Cyclone asynchronous timing and power analysis flow.[1] Our asynchronous circuit design is specified using a hierarchical format that contains the logical specification of the gates for each component, transistor sizing information, and connectivity. From the design description, we automatically generate the SPICE netlist and circuit netlist for the design. Unique gates or small groups of gates that are repeatedly used in the design are factored out into *cells*, and the netlist is rewritten so that all

[1]This section is based on our work [20].

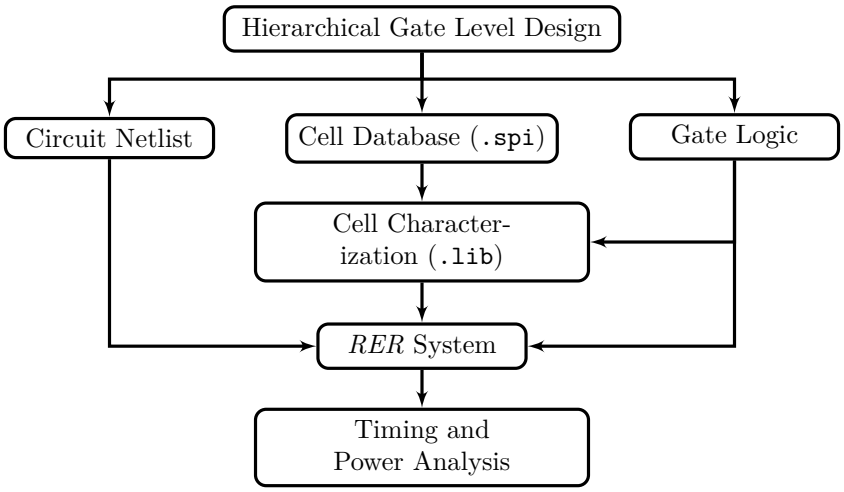

Figure 3.1: Overview of Cyclone analysis flow

gates correspond to cell instances with individual unique cells being specified by their SPICE netlists. This information is used for cell characterization, followed by timing graph generation and then timing and power analysis. The overall flow is outlined in Figure 3.1.

3.1.1 *RER* Construction and *RER* Skeletons

Unlike synchronous logic where the timing arcs are specified in the `.lib` file, in asynchronous logic the same gate used in different contexts may have different timing arcs. The index priority simulation (IPS) algorithm for quasi delay-insensitive (QDI) circuits developed in [28] computes the *RER* system from the gates and the state of the circuit when reset is asserted [28].

There are two major challenges with IPS algorithm: (i) *limited scalability*: the algorithm requires simulating the *entire* asynchronous circuit digitally until a repeating state is found—which can be *slower* than timing analysis in practice when the circuit is large; (ii) *limited capability*: the algorithm is designed to only handle

systems with AND-causality and some limited forms of OR-causality, and cannot handle common scenarios that arise such as dual-rail encoded data. Our *RER* system generation method adopts the idea of digital simulation of asynchronous circuits, but has a number of modifications and improvements to address both limitations of IPS algorithm, and will work for general asynchronous circuit families.

To address limited scalability, notice that the goal of IPS is to encode event dependencies as *RER* systems. These dependencies are determined by the local circuit within an individual process. We know the process boundaries within our circuits, because the input to the timing analysis flow is a hierarchical design. Hence, we partition the design into individual components at the level of modules with input and output channels, and generate for each module its local *RER* system. Also, we record the events that are shared between processes in the environment. This entire procedure only has to be executed once per unique process in the system; if modules are re-used, this step need not be repeated.

We refer to this as generating an *RER skeleton*—a building block of the overall *RER* system for the complete circuit. A circuit also needs to be closed by the environment in order for IPS algorithm to generate the correct *RER* skeleton. In our implementation, we connect data sources, which always send input data, and data sinks, which always consume output data, when necessary.[2]

To address limited capability, consider a simple scenario where data is encoded as a dual rail value on true and false rails. If we use IPS, then the *RER* skeleton generated will only cover transitions resulting from either the true or false rail, but not both, because it always simulates and includes the data branch with highest index priority (predetermined) in its state graph. We build a complete *RER* skele-

[2]This is correct for slack elastic systems [30] like pipelined asynchronous circuits. More general systems will require more sophisticated techniques to generate valid environments.

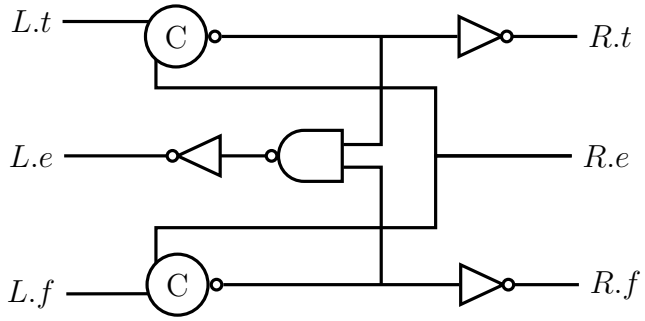

Figure 3.2: 1-bit WCHB FIFO Circuit

ton by adopting the similar idea presented in [24], and tracking for both true and false data branches of the circuit to include all possible digital states for the *RER* skeleton. Our implementation does not assume the perfect symmetry as in [24], and can exclude non-reachable state transitions due to any timing constraints. Thus, our method can be applied to more general asynchronous circuit families. Based on this accurate state graph, we can construct a complete *RER* skeleton for local circuits. Assuming AND-causality, this *RER* skeleton provides a worst-case estimate of the timing analysis for asynchronous circuits as desired. This is similar to the situation in synchronous logic, where an upper bound on the maximum delay is computed during static timing analysis. Figure 3.2 gives the circuit of 1-bit weak-conditioned half-buffer (WCHB) and 3.3 shows its complete *RER* skeleton.

The final complete *RER* system for the circuit is constructed by combining *RER* skeletons for all components. The skeleton of two communicating processes will have shared events; those events can be merged to construct the combined *RER* system. This procedure avoids global digital simulation of the circuit, and is significantly faster than using traditional IPS.

The complete *RER* system obtained in this manner is the *timing graph* that we use for further analysis.

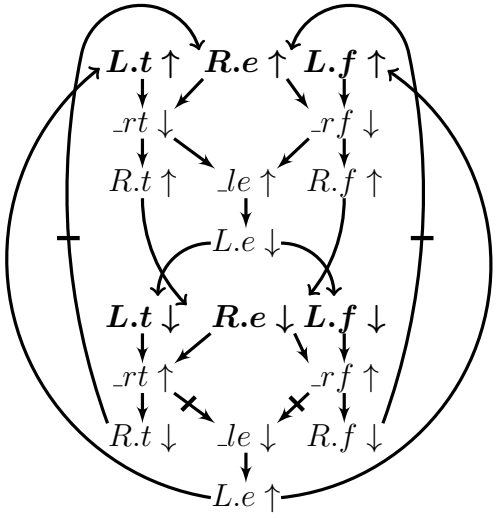

Figure 3.3: Complete *RER* skeleton for WCHB. Events generated by the environment are shown in bold.

3.1.2 Cell Characterization

SPICE netlists for individual cells are simulated in HSPICE for cell characterization. Note that previously characterized cells can omit this step by using the pre-computed cell information. We store characterization results in Synopsys Liberty (`.lib`) format. For timing analysis, we use the standard "non-linear delay model" (NLDM) for gate delay, and both delay and output transition time are stored in 2D tables, indexed by input slew rate and output load. For power analysis, we store leakage power and internal energy. Cell leakage power is a single value for each input vector; internal energy per timing arc is stored in 2D tables, indexed by input slew rate and output capacitance as well.

It should come as no surprise that the problem of characterizing combinational gates in asynchronous circuit design is the same as that in its synchronous counterpart. However, asynchronous circuits can include general state-holding gates with internal feedback, and there are several tricks to characterize such gates.

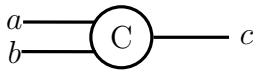

Figure 3.4: C-element

a	b	c
0	0	0
0	1	c_{prev}
1	0	c_{prev}
1	1	1

$$a\&b \mapsto c\uparrow$$
$$\sim a\& \sim b \mapsto c\downarrow$$

Figure 3.5: C-element truth table and PRS

The LiChEn tool can characterize asynchronous standard cells including state-holding elements [41]. However, LiChEn represents state-holding gates with feedback in the logic expression, and can only characterize single output cells as a result. Consider the C-element example in Figure 3.4, whose truth table and PRS are shown in Figure 3.5. The functionality of this C-element can be expressed as $c = a * b + c_{prev} * !a * b + c_{prev} * !b * a$, which leads to feedback of pin c because of the state-holding property of this gate. Instead, we use the PRS introduced in chapter 2, which permits pull-up and pull-down expressions to be specified directly without feedback. This makes it straightforward to analyze state-holding gates and characterize their timing properties.

Another property one needs to pay attention to when characterizing general state holding gates is the "separate conditions" for output rising and falling transitions. To elaborate this point, let us look at the difference between the characterization of a 2-input AND gate shown in Figure 3.6 and Figure 3.7 and the 2-input C-element shown above.

Figure 3.6: AND gate

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

$$a\&b \mapsto c\uparrow$$
$$\sim a|\sim b \mapsto c\downarrow$$

Figure 3.7: AND gate truth table and PRS

```
pin(c) {
  direction : output;
  function :  "a*b";
  timing() {
    related_pin : "a";
    timing_sense : positive_unate;
    when :  "b";
    cell_rise(delay_mxn)
       ...
    }
    cell_fall(delay_mxn)
       ...
    }
  }
}
```

Figure 3.8: AND delay information in (`.lib`)

In both gates, in order to characterize the propagation delay from the signal transition $a\uparrow$ to $c\uparrow$, we have to first set b to 1 and c to 0. Then, a is changed from 0 to 1, and we can determine the propagation delay from $a\uparrow$ to $c\uparrow$. For the combinational AND gate in Figure 3.6, without changing b, changing a back to 0 would change c back to zero; however, the characterization engine has to change b to 0 to characterize the signal transition from $a\downarrow$ to $c\downarrow$ in the C-element. In terms of the Synopsys Liberty file format, the "cell_rise" and "cell_fall" information for pin c corresponds to different input conditions ($b = 1$ and $b = 0$), and thus they have to be measured and characterized separately. The delay information saved in (`.lib`) for the 2-input AND gate and 2-input C-element have the formats shown in Figure 3.8 and Figure 3.9, respectively.

```
pin(c) {
  direction : output;
  function :  "a*b+c*!a*b+c*!b*a";
  timing() {
    related_pin : "a";
    timing_sense : positive_unate;
    when :  "b";
    cell_rise(delay_mxn)
      ...
    }
  }
  timing() {
    related_pin : "a";
    timing_sense : positive_unate;
    when :  "!b";
    cell_fall(delay_mxn)
      ...
    }
  }
}
```

Figure 3.9: C-element delay information in (`.lib`)

In our implementation, we extract information from the gates to determine all the different timing scenarios for a state-holding gate. Since we do not need to worry about the problem of feedback loops in the logic function expression encountered in [41], we can readily handle those standard cells with multiple outputs.

3.1.3 Steady-State Slew Rate Computation

We need to annotate timing graph edges with delay values and calculate internal energy. Using NLDM, delay and internal energy values can be extracted from 2D look-up tables indexed by input slew rate and output capacitance. In synchronous timing and power analysis, circuits are partitioned into acyclic regions separated by

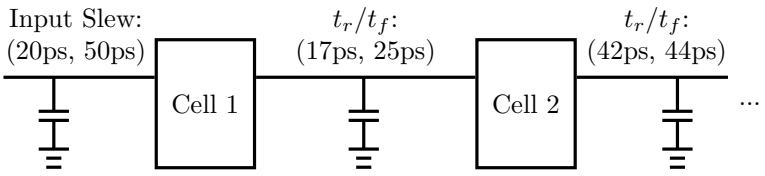

Figure 3.10: Slew range propagation between stages. The final value will depend on cell sizes and parasitic capacitance.

clocked elements and the input slew rates are determined by parameters related to the clock input or user input (e.g. driving cells). For asynchronous circuits, however, there is no such "break point" that can be used to obtain the input slew rate of each stage. Instead, the input slew rate of a cell is given by the rise/falling transition time of its input, which in turn depends on the slew rate of the cell before it. Such a cyclic dependence needs a different technique to extract the slew rates used for delay and power calculation.

Fortunately, the rising/falling transition time of a cell output is a weak function of its input slew rate. Even starting with a wide range of input slew rate at a cell, the falling/rising transition time converges to a narrow range after a few stages of propagation, as the example shown in Figure 3.10.

More precisely, suppose $f(x) \in \mathbb{R}_{>0}$ is the output transition time for a cell output given the input transition time $x \in \mathbb{R}_{>0}$ and the output load. We note that given an interval $[a, b]$ with $b - a \geq \delta$, $f(b) \geq f(a)$ and $|(f(b) - f(a))/(b - a)| \leq c < 1$. Thus, for any arbitrary signal transition (node) on the timing graph, we can start with a slew rate range assumption and propagate this through the circuit in an iterative fashion. Capacitance information is obtained from parasitics as well as the characterization information from the `.lib` file, so that the appropriate slew rate is propagated through gates. We keep propagating the slew rate ranges on the timing graph for each signal transition, until the ranges converge. Experimental results

show that slew rate ranges nearly always converge to very small intervals within several propagations, which provides an efficient way to calculate the accurate steady-state input slew rate for every signal transition.

3.1.4 Delay Calculation

Once slew rates are obtained for all signals, delay values for all the edges in the timing graph can be computed by using the slew rate and output load, combined with the cell characterization information in the `.lib` file. At this point in the flow, we will have a graph $G = (V, E)$ corresponding to the *RER* system constructed. In addition, each edge $e \in E$ has a weight $w(e)$ given by the delay values computed as above, and a tick value $\epsilon(e)$ that is either zero or one. Armed with this graph, we compute the cycle period $p^\star$ as described next.

3.1.5 Maximum Cycle Ratio Algorithm

The maximum (and minimum) cycle ratio problem has been explored in many previous papers. Burns' original *RER* formulation [5] uses linear programming as a direct method to compute $p^\star$, and recent work also uses linear programming as the baseline algorithm [27]. The performance of linear programming depends strongly on the solution technique and the quality of the solver. As we will show in Section VI, linear programming becomes much slower for large circuits.

Our implementation uses the Young-Tarjan-Orlin (YTO) algorithm [53], an asymptotically efficient implementation of Karp-Orlin's algorithm [25]. A previous survey has verified that these two algorithms are the fastest and most robust

maximum cycle ratio algorithms both in theory and practice [10]. Although their worst-case time complexity is $\mathcal{O}(|V||E|log|V|)$, the runtime for practical timing graphs is significantly faster than the worst-case complexity.

3.1.6 Slack Computation

Once we have determined the maximum cycle ratio ($p^\star$) and M, we can use standard timing propagation as described in Section III to compute the performance slack of each event, and the correctness slack for each timing fork. These values have to be computed for M versions of each transition, and the most conservative value is reported as the slack.

3.1.7 Power Calculation

The power of a circuit includes leakage power and dynamic power, and in particular, dynamic power consists of internal (short circuit) power and switching power (charging or discharging the capacitors). Given the cell characterization results in the .lib files, we compute for each cell i $P_{leakage_m}(i)$, the maximum leakage power over different input vectors; and for each signal transition j in the *RER* system $E_{internal_m}(j)$, the maximum internal energy per cycle for timing arcs joining at j. Then we report the pessimistic total power of the circuit if it runs at frequency $\frac{1}{p^\star}$: $P_{total} = P_{leak}(circuit) + P_{dyna}(circuit) = \Sigma_i P_{leakage_m}(i) + \frac{\alpha(j)}{p^\star}(\Sigma_j E_{internal_m}(j) + \Sigma_j \frac{1}{2} C_{load}(j) V_{DD}^2)$ where $C_{load}(j)$ is the driving load of the gate output pin corresponding to j and $\alpha(j)$ is the activity factor of j. For QDI circuits, $\alpha(j)$ is 1 since each signal transition occurs at frequency $p^\star$. For combinational logic in bundled-data circuits, simulation-based activity factors can be incorporated into

the dynamic power computed by Cyclone.

3.1.8 Parallelization Strategy

We leverage parallelization to achieve fast timing and power analysis. Since circuits are modeled by graphs in Cyclone analysis flow, we use the Galois framework, which supports parallelism for graph-based algorithms, and analyze the parallelism available in Cyclone with the operator formulation [48], a *data-centric* abstraction of algorithms. This part of the work is in collaboration with Yi-Shan Lu, a Ph.D. student in professor Pingali's research group from University of Texas, Austin.

The Operator Formulation

The operator formulation starts from identifying the data structures involved in the algorithm, e.g., a graph. *Active elements* capture where in the graph the computation needs to be done. An *operator* specifies the rules to update the graph, and it will be applied to active elements. Each application of an operator to an active element is called an *action*. An action may need to read from or write to a set of nodes and edges around the active element, which is termed the *neighborhood* of the action. Active elements become inactive once the actions are finished.

Algorithms can be categorized as *data-driven* or *topology-driven* based on the pattern of active elements. A data-driven algorithm begins with a set of initially active elements, generates new active elements on the fly, and terminates when there are no more active elements to be processed. In contrast, a topology-driven algorithm makes sweeps over all nodes/edges until certain convergence criteria is

reached.

Scheduling needs to be considered when there are multiple active elements at the same time. For *unordered* algorithms, processing active elements in any order gives the same answer. However, some ordering may be more efficient than the others. For *ordered* algorithms, active elements should appear to be processed in certain ordering for correctness.

Parallelism in graph algorithms can be exploited among actions with disjoint neighborhoods.

Available Parallelism

With the operator formulation of algorithms, we are ready to analyze the parallelism available in Cyclone, composed of deriving edge weights, maximum cycle ratio, timing propagation, power computation and timing constraint checking.

Deriving Edge Weights

As mentioned in Section 3.1.3, edge weights in a timing graph are computed by computing the steady-state slew rates first and then computing delays by table lookup.

Slew propagation is a topology-driven, unordered algorithm. It sweeps all nodes until the slew intervals are small enough. In one sweep, all nodes can be processed in parallel, since for each node n, it computes n's new slew interval based on n's predecessors' old slew intervals and n's successors' loads.

Delay computation is a topology-driven, unordered algorithm. It only makes

one sweep over all nodes. All nodes can be processed in parallel, since the table lookup for node n only reads from n's predecessors for slew rate, reads from n's successors for loads, and writes to n for delay.

Maximum Cycle Ratio

Recall from Section 3.1.5 that we use Young-Tarjan-Orlin algorithm for computing maximum cycle ratio. It contains two main pieces: longest-path tree construction, and edge swapping [53].

Longest-path tree construction is a data-driven, unordered algorithm. It starts from a synthetic source node, which is connected to all nodes in the timing graph. By using a pull-style operator, e.g., reading from the predecessors of node n and writing to the n itself, a node will activate its successors if its distance from the source increases. All active nodes can be processed in parallel, even if their neighborhoods overlap because (i) maximum function exhibits monotonicity; and (ii) no updates will be missed using the aforementioned activation scheme.

Edge swapping is a data-driven, ordered algorithm. At any point, only the edge with the largest key value is active. This means that there is no parallelism available here.

Timing Propagation

Timing propagation consists of computing arrival time and computing required time, both are data-driven, unordered algorithms. For arrival time, nodes on the critical cycle with incoming ticked edges are active initially, and other nodes become active when their predecessors' arrival times are updated. For required time,

nodes on the critical cycle are initially active, and other nodes will be activated once their successors' required times are updated. All active nodes can be processed in parallel, similar to longest-tree construction.

Power Computation

Computing leakage power for cells is a topology-driven, unordered algorithm, as it sweeps once over all cells, each of which reads from the cell libraries and writes to itself. Computing dynamic power is also a topology-driven, unordered algorithm, since nodes representing gate outputs are all active, only read from their neighboring nodes, and no two gate outputs can be neighbors.

Correctness Slack Computation

Correctness slacks are computed as follows. For a timing fork $\langle r, 0\rangle : \langle x, j\rangle < \langle y, k\rangle$ where $j, k \in \{0, 1\}$, $\langle r, 0\rangle$ is initially active with $delay(p_{\langle r,0\rangle,\langle r,0\rangle}) = 0$. We then compute $FO_{\langle r,0\rangle}$, which contains all the $\langle n, i\rangle$ reachable from $\langle r, 0\rangle$ using at most one ticked edge. Computing maximum/minimum delay from $\langle r, 0\rangle$ for nodes in $FO_{\langle r,0\rangle}$ is similar to propagating arrival times; as $FO_{\langle r,0\rangle}$ is always a directed acyclic graph (DAG), we can process its nodes in topological order. Finally, correctness slack is computed as defined in Section III.

Computing $FO_{\langle r,0\rangle}$ is a data-driven, unordered algorithm, as nodes can be marked in arbitrary order. Computing maximum/minimum delay from $\langle r, 0\rangle$ for all nodes in $FO_{\langle r,0\rangle}$ is a data-driven, ordered algorithm. $\langle r, 0\rangle$ is initially active in both algorithms. All timing forks can be processed in parallel provided that they are tracked separately.

Implementation in Galois

We implement Cyclone using the Galois framework [47, 29], a C++ library for parallel programming based on the operator formulation. The Galois framework (i) provides parallel data structures, and language constructs for highlighting parallelization opportunities; and (ii) supports dynamic work generation, load balance, resource management, and transactional execution of operators.

All sub-algorithms in asynchronous timing analysis are implemented as described in Section 3.1.8. Below we illustrate some special handling using the Galois framework.

- Sub-algorithms use pull-style operators.
- To have better locality for memory accesses, all variables only have one copy. To ensure proper synchronization, atomic instructions are used for all sub-algorithms except for delay computation and edge swapping.
- Steady-state slew calculation starts from $[0, \rho]$ and ends when the resultant interval is smaller than $\rho/1000$, where ρ is the default maximum transition time. The difference given by the final interval is $< 1ps$ in our experiments.
- In longest-path tree construction, instead of introducing a synthetic source node, we initialize the distances of all nodes to $-\tau = -(1 + \sum_{e \in G} |delay(e)|)$ similar to that in [25] and let all nodes be active initially. τ can be computed with `galois::GAccumulator`: each thread adds $|delay(e)|$ for its portion of edges to a thread-local sum, and then the master thread reduces the thread-local sums to the final answer.
- During edge swapping cases, any updates to distance and ticks for a subtree are parallelized.

- For computing correctness slacks, each node keeps a map from timing forks' roots to its maximum/minimum path delays in order to track multiple timing forks simultaneously. The maps are backed by Galois per-thread memory allocators to avoid serialization due to system calls for allocating memory.

3.2 Applications on Asynchronous + Synchronous Circuits

Our theory is also capable of modeling circuits that contain both synchronous and asynchronous components.[3] A simple free-running clock can be easily modeled as an *RER* system with two transitions $clk\uparrow$ and $clk\downarrow$, and two rule templates that relate the two transitions and whose delays correspond to the duration for which the clock is high and the clock is low. This transition can be used to determine propagation delays of other signals (the output of state-holding elements, for example) in the usual way. If part of the circuit described by the *RER* system is asynchronous and the clock is truly synchronous (i.e. not pausable/stretchable, etc.), there is no path from any event in the asynchronous circuit back to $clk\uparrow$ or $clk\downarrow$. Hence, most previous theory of periodicity does not apply to such an *RER* system because the collapsed constraint graph isn't strongly connected. However, if the cycle period for the clock is in fact the critical cycle period (which is typical), then our results apply as long as there is a path from $clk\uparrow$ or $clk\downarrow$ to all the other events in the system.

The immediate consequence of this is that if there is an asynchronous component whose primary inputs are driven by a clocked environment, and there is a path (in the *RER* system) from the clock transition to every other transition in

[3]This section is based on our work [21].

the system, then all signals in the entire *RER* system will be periodic with the same period as the external clock after finite signal transitions. In this scenario, we have $M = 1$ since there is only one critical cycle—the one that determines the clock period.

Table 3.1 demonstrates the synchronization of asynchronous systems with an external periodic input signal (a clock). In particular, we used linear arrays of FIFO in Figure 3.11 (denoted $L(a, b)$ with $a + 2b$ stages with b initial data values) implemented with weak-conditioned half buffers in Figure 3.2. In this section, without loss of generality, we simplify the analysis by assuming that the circuits are designed with symmetric timing characteristics for both true and false data values, and assume that each gate has delay of 10 time units. The input to the array is synchronized to a clock with fixed period of 200 time units (20 times gate delays in our simulations). We selected this period because it was longer than the cycle period of the free-running asynchronous circuit, so that the critical cycle in the system is the clock cycle. Additional asynchronous arithmetic circuit examples that we show consist of 2, 4, and 8 bit adders (*ADD*) and a 4-bit and 8-bit array multiplier core (*MUL*). In each case, the external data inputs are clocked with the clock cycle period shown, and the time at which the output of the asynchronous core is synchronized to this external clock is shown as well. The adder and multiplier circuits are implemented using pre-charged half-buffer asynchronous logic [36], also assuming symmetric sizing of data wires.

Theorem 3 implies that such a system will exhibit exact periodicity, and provides a bound on the time taken to reach this periodic state. The worst case bound is at least $8|E'|$, and in our examples $|E'|$ is quite large. Table 3.1 shows that in reality, the system typically enters the steady-state much faster than the bound, and

Table 3.1: Linear FIFOs and pre-charged adder and multiplier core completion time. The term in parentheses in the first completion column is the time divided by $p^\star$, providing an indication of the number of occurrences of transitions before periodicity is observed.

Structure	Clock Period	First Completion
L(1,0)	200	340 (1.7)
L(1,1)	200	380 (1.9)
L(5,0)	200	420 (2.1)
L(5,2)	200	500 (2.5)
L(10,6)	200	960 (4.8)
L(20,0)	200	720 (3.6)
L(50,8)	200	1640 (8.2)
L(100,6)	200	2560 (12.8)
L(100,20)	200	3120 (15.6)
ADD(2)	400	660 (1.7)
ADD(4)	400	700 (1.8)
ADD(8)	400	780 (1.9)
MUL(4)	400	800 (2.0)
MUL(8)	400	960 (2.4)

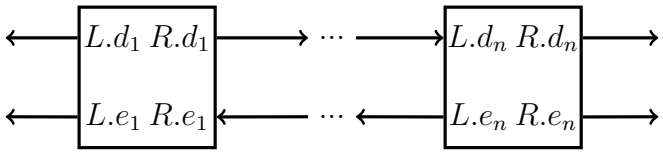

Figure 3.11: Linear FIFOs

asynchronous circuit outputs quickly become synchronous with respect to the external clock. Hence, an asynchronous circuit where all the inputs are synchronized to a single clock will have synchronous outputs after finite asynchronous outputs—and the outputs can therefore be read by a clocked circuit—*without metastability*, as implemented in Figure 3.12.

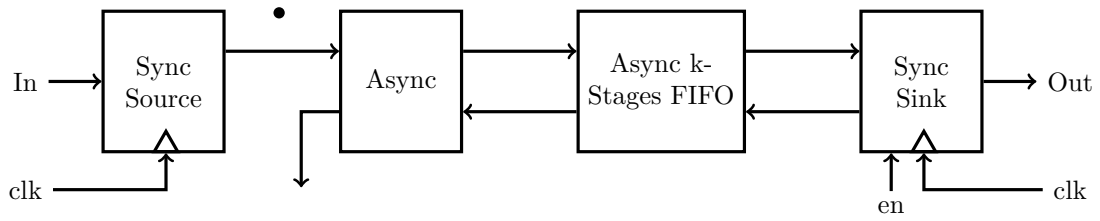

Figure 3.12: Synchronous to asynchronous metastability free interface

3.2.1 Synchronous to Asynchronous Metastability-Free Interface

In our implementation, the input of an asynchronous circuit (labelled "Async") is connected to the output of a synchronous circuit. Assume that the clock period is $p_1^\star$ and the asynchronous circuit has the largest cycle mean $p_2^\star$ (which determines the natural frequency of the asynchronous circuit). If $p_1^\star > p_2^\star$, the combined system will have $p^\star = p_1^\star$ and $p' = p_2^\star$. As we have established, the output of the asynchronous circuit will become synchronous to the external clock with fixed occurrence period $p^\star$ after finite occurrences of transitions. It is then possible to build a metastability-free interface between synchronous and asynchronous logic. To handle the initial transient when the asynchronous circuit is not periodic, a k-stage FIFO is attached to the output of the asynchronous circuit to hold the initial asynchronous tokens for synchronous reading. We assume that the handshake of a k-stage FIFO is fast enough and it will not change the critical cycle ratio $p^\star$ of the whole system (this is typical in our experience, since a FIFO is one of the fastest asynchronous circuits). The output of the FIFO is connected to another clocked component (with the same clock period $p^\star$) with an enable signal that is initially low.

Note that any finite part of a timing simulation can be easily computed using

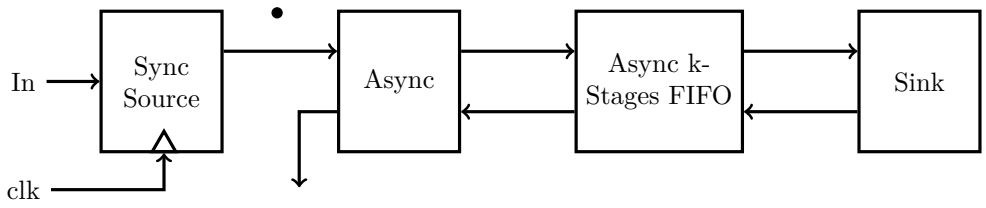

Figure 3.13: Clock + async + FIFOs with perfect sink

standard simulation tools. We proceed as follows.

Step 1. We connect a perfect sink—a circuit that acknowledges the data instantly—to the output of the FIFO, and simulate the circuit in Figure 3.13. In this case, the value of k does not matter and the token will be removed by the sink as soon as possible. The entire system becomes synchronous to the external clock after each transition occurs a certain number of times—denoted by m.

We record when the ith token arrives at the output of the asynchronous circuit (which is equivalent to ith occurrence of the output transition) and denote that time by t_i. Since the circuit becomes synchronous after each transition occurs m times, we know that $t_{m+j} = t_m + jp^\star$ for all $j \in \mathbb{N}$.

Each FIFO stage can hold one token.[4] Denote the minimum delay for a token at one FIFO stage to propagate to the next stage by d, assuming the rest of the FIFO is empty.

Step 2. We would like to replace the circuit from Figure 3.13 (that destroys the output of the asynchronous circuit) with Figure 3.12.

Suppose we change the enable from low to high after n ($n \in \mathbb{N}^+$) clock cycles (at time $t'_e = np^\star$). t'_e has to be large enough that by each time the clocked component connected to the output of the FIFO acknowledges a token, there is a token there

[4]If the FIFO is implemented using WCHBs, then a FIFO stage would require two WCHB stages.

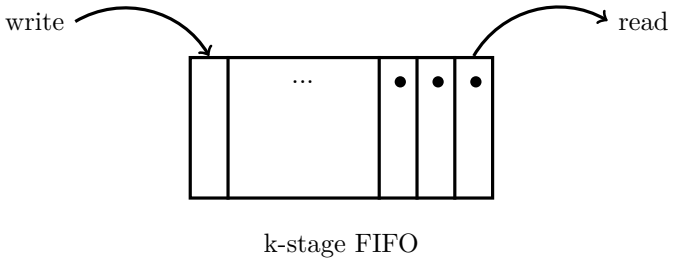

Figure 3.14: Asynchronous writing and reading

for acknowledgement. Also, k has to be large enough such that the FIFO can hold all the tokens that have been produced by the "Async" block but have not been acknowledged by the clocked component, without overflow. To simplify matters, we assume that the enable signal is changed at the negative edge of the clock (i.e. so the enable is asserted half-way through the nth clock cycle). Hence, the output circuit reads the FIFO data at times $t'_e + (i + \frac{1}{2})p^\star$, for $i \in \mathbb{N}$.

The first token ($i = 0$) is produced by the "Async" block at time t_1 (from Step 1), and it propagates through the FIFO with delay $k \times d$.[5] Generalizing, it is clear that if the time t'_e satisfies the constraints

$$t'_e + (i + \frac{1}{2})p^\star \geq t_{i+1} + kd \tag{3.1}$$

for all $i \leq m$, $i \in \mathbb{N}$, we are able to read the tokens stored in the k-stage FIFO with the same clock period $p^\star$ without metastability as in Figure 3.14. For the case when $i > m$, we know from our periodicity result that $t_{m+j} = t_m + jp^\star$, and hence equation (3.1) for $i = m$ is sufficient to show that the circuit continues to operate correctly for the rest of the data.

To characterize k, define a new parameter d' to be the delay that it takes for an

[5]This assumes that the FIFO is implemented by a linear array of one-place FIFOs. For FIFOs implemented in other ways (e.g. using a dual-ported memory), the delay is lower. Hence, our assumption is conservative.

empty stage to move from $(i+1)$th stage to ith stage. This parameter characterizes the process that a token at the $(i+1)$th stage is acknowledged and removed, and the token at the ith stage moves to the $(i+1)$th stage, effectively create an empty stage move from the $(i+1)$th stage to ith stage. Since it considers the time it takes for the next stage of FIFO to be ready to accept a new token, $d' > d$. We can also make a reasonable assumption that $d < d' \ll p^\star$. If

$$k \geq u \tag{3.2}$$

for any $u \in \mathbb{N}^+$ such that $t_u < t'_e + kd' + \frac{1}{2}p^\star$ and

$$k \geq u - \left(\left\lfloor \frac{t_u - (t'_e + kd' + \frac{1}{2}p^\star)}{p^\star} \right\rfloor + 1 \right) \tag{3.3}$$

for any $u \in \mathbb{N}^+$ such that $t_u \geq t'_e + kd' + \frac{1}{2}p^\star$, the FIFO will never overflow. To see this, first notice that at time t_u, the total number of tokens that have been generated is u. Meanwhile, we start to remove the tokens synchronously at time $t'_e + \frac{1}{2}p^\star$, and at time $t_u \geq t'_e + \frac{1}{2}p^\star$, $\left(\left\lfloor \frac{t_u - (t'_e + \frac{1}{2}p^\star)}{p^\star} \right\rfloor + 1 \right)$ tokens have been removed. Furthermore, the length of the FIFO might need to be larger to compensate the effect that when the token is removed, the empty stage is on the right most side, and it takes some more time for this empty stage to "propagate" to the left and effectively create a new space for tokens generated by the asynchronous circuit. Since it takes at most kd' time units for an empty space to propagate from one end of the FIFO to the other, at time $t_u \geq t'_e + kd' + \frac{1}{2}p^\star$, at least $\left(\left\lfloor \frac{t_u - (t'_e + kd' + \frac{1}{2}p^\star)}{p^\star} \right\rfloor + 1 \right)$ empty stages are actually created and available at the output of the asynchronous circuit. For simplicity, we get rid of the case when $t_u < t'_e + kd'$ and use (3.3) as a uniform expression of k. Notice that (3.3) is a sufficient condition of (3.2) when $t_u < t'_e + kd'$. Theoretically, such a simplification might result in unnecessarily large value of k in some unusual systems when the difference between t_{i+1} and t_i are many times of $p^\star$ for some i. We do not have to worry about this in most practical

circuits. Re-arranging the inequality, a simple version of sufficient condition of k such that the FIFO will never overflow is

$$k \geq \frac{u - \frac{t_u - (t'_e + \frac{1}{2}p^\star)}{p^\star}}{1 - \frac{d'}{p^\star}} \tag{3.4}$$

for any $u \in \mathbb{N}^+$. We do not actually have to verify infinite number of integer values u either, since again, $t_{m+j} = t_m + jp^\star$ for any $j \in \mathbb{N}$. If the inequality (3.4) holds for some $u_1 \geq m, u_1 \in \mathbb{N}^+$, it automatically holds for all $u \geq u_1$, $u \in \mathbb{N}^+$.

There are two technical details worth mentioning. First, we can always pick t'_e large enough so that equation (3.1) is satisfied by an appropriate choice of k so long as the "Async" circuit's output data token rate is approximately equal to its input data token rate.[6] The reason is that d is typically much smaller than $p^\star$, since an asynchronous FIFO is one of the fastest circuits one can design. The second technical detail has to do with the value of kd. d was chosen as the empty FIFO delay, but if $k > 1$ and we consider the ith token's propagation delay, the FIFO may not be empty. This could result in the ith token taking more time to propagate through the FIFO than kd. However, this stall would occur only because there are earlier tokens that have already reached the output of the FIFO. Hence, the stall is therefore a result of the "Async" block producing tokens *faster* than they can be read by the synchronous output. This scenario also implies that the output can be safely read via the synchronous interface without metastability. Hence, we need not consider this as a separate case and we simply have to verify equation (3.1). The property we have relied on is that for large enough k, the FIFO will not perturb the timing of the "Async" circuit.

[6]This means, for example, that the "Async" block cannot produce two data outputs for each data input. In this scenario, the synchronous output will not be able to keep up with the "Async" block.

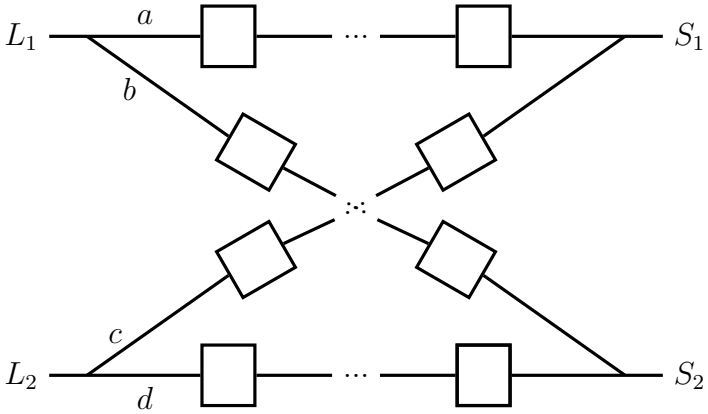

Figure 3.15: Cross-connected FIFOs

Table 3.2: Metastability-Free Interface Implementation

Async Circuit	Clock Period	k	t'_e
L(5,2)	200	3	200
L(10,6)	200	6	200
L(100,20)	200	20	800
F(2,3,2,3)	200	1	200
F(3,4,7,10)	200	2	400
F(4,7,9,16)	200	1	600
ADD(4)	400	2	400
ADD(8)	400	1	400
MUL(4)	400	1	400
MUL(8)	400	1	400

We used the method outlined above to implement the metastability-free interface to some of our asynchronous circuits examples. We simulated the timing of transitions using our asynchronous circuit simulation tools. Here we also included more complicated circuit structures: the cross-connected ("X") FIFO topologies $F(a, b, c, d)$ as in Figure 3.15, which are composed of four linear arrays of a, b, c, d buffers (with no initial data values) respectively, where the primary inputs L_1 and L_2 are copied to the FIFOs of length a, b and c, d respectively, and the output of the FIFOs are combined to produce the two primary outputs. We report the minimum k and the corresponding minimum t'_e that are valid for each different asynchronous circuit in Table 3.2, and that result in metastability-free operation.

The large values of k in Table 3.2 correspond to asynchronous circuits that can produce output tokens *without* receiving any inputs. Note that for most of the circuits, the value of k is either one or two; in the other cases, it is zero or one plus the number of initial tokens can be output by the "Async" block without receiving any external input. Therefore, we expect that the FIFOs needed in many practical cases (for example arithmetic units and datapath blocks) will be small.

3.2.2 Extensions to Bounded Delay Systems

Work on the problem of time separation of events (TSE) considered systems under bounded delay models [22, 38]. The analysis is essentially based on periodic properties under fixed delay models where delays of all the rules are set to the maximum or minimum of the range. Notice that if one permits even a tiny and arbitrary pertubation of a delay value $[\alpha, \alpha + r]$ on an edge on the critical cycle, the system is no longer periodic. This can be easily seen by examining a simple ring oscillator, where one of the inverters has variable delay. A perturbation of the delay value on an edge which is not on the critical cycle might also have effects on timing properties of the system. In reality, a clock is not exactly periodic either, because of jitter. However, the exact periodicity under fixed delay models reveals the core properties of such systems, and applications based on such properties can be readily generalized to bounded delay systems by similar methods as in previous work. We now examine how our design of the synchronous-asynchronous interface works under bounded delay models.

Theorem 4 (Monotone Property of Timing Simulations)**.** *Assume that the timing simulation of an arbitrary event $\langle f, n_f \rangle$ in an ER system $\langle E, R \rangle$, is $\hat{t}(\langle f, n_f \rangle)$. If the delay of one arbitrary rule r in the system is changed from α to $\alpha + w$ with*

$w > 0$, we will get a new timing simulation of $\langle f, n_f \rangle$, $\hat{t}'(\langle f, n_f \rangle)$. Then, we must have $\hat{t}(\langle f, n_f \rangle) \leq \hat{t}'(\langle f, n_f \rangle) \leq \hat{t}(\langle f, n_f \rangle) + w$.

Proof. Before we increase the delay of r, we denote a zero slack e-path from some initial event $\langle g_1, 0 \rangle$ to $\langle f, n_f \rangle$ as p_0. After the delay of r is increased by w, we denote a zero slack e-path from some initial event $\langle g_2, 0 \rangle$ to $\langle f, n_f \rangle$ as p_1. Thus, $\hat{t}(\langle f, n_f \rangle) = \delta(p_0)$ and $\hat{t}'(\langle f, n_f \rangle) = \delta(p_1)$. We also denote the path corresponding to p_0 (consisting of the same transitions and the same rules) after the delay increase as p_2 and that corresponding to p_1 before the delay increase as p_3. By Corollary 2, we have $\hat{t}(\langle f, n_f \rangle) \geq \delta(p_3)$ and $\hat{t}'(\langle f, n_f \rangle) \geq \delta(p_2)$. Since we only increase the delay of one rule r in the *ER* system by w, we also have $\delta(p_0) \leq \delta(p_2) \leq \delta(p_0) + w$ and $\delta(p_3) \leq \delta(p_1) \leq \delta(p_3) + w$. Altogether, we have $\hat{t}(\langle f, n_f \rangle) = \delta(p_0) \leq \delta(p_2) \leq \hat{t}'(\langle f, n_f \rangle) = \delta(p_1) \leq \delta(p_3) + w \leq \hat{t}(\langle f, n_f \rangle) + w$ as desired. □

Lemma 6. *Since we assumed that $p^\star \gg d' > d$, let $p^\star \geq qd$ where $q \in \mathbb{N}^+, q \gg 2$, and it is also reasonable to assume that $\frac{1}{2} < 1 - \frac{d'}{p^\star} < 1$. If (n_1, k_1) is a feasible solution to inequalities (3.1) and (3.4), $(n_1 + 1, k_1 + 2), ..., (n_1 + 1, k_1 + q)$ are also feasible solutions (assume that change of n_1 and k does not have effects on $t_i(t_u), p^\star, d'$ and d).*

Proof. For any specific $i \in \mathbb{N}$, if (3.1) holds for (n_1, k_1), then

$$n_1 p^\star + p^\star + (i + \frac{1}{2})p^\star \geq t_{i+1} + k_1 d + vd \tag{3.5}$$

for any $2 \leq v \leq q, v \in \mathbb{N}^+$. Similarly, if (3.4) holds for (n_1, k_1), then by the assumption,

$$\begin{aligned} k_1 + v \geq k_1 + 2 &\geq \frac{u - \frac{t_u - (n_1 p^\star + \frac{1}{2}p^\star)}{p^\star}}{1 - \frac{d'}{p^\star}} + 2 \\ &> \frac{u - \frac{t_u - (n_1 p^\star + \frac{1}{2}p^\star)}{p^\star} + 1}{1 - \frac{d'}{p^\star}} = \frac{u - \frac{t_u - ((n_1+1) p^\star + \frac{1}{2}p^\star)}{p^\star}}{1 - \frac{d'}{p^\star}} \end{aligned} \tag{3.6}$$

for any $2 \leq v \leq q, v \in \mathbb{N}^+$ as desired. $\square$

Lemma 6 immediately implies the following Corollary.

Corollary 6. *If two systems have solutions* (n_1, k_1) *and* (n_2, k_2) $(n_1, n_2, k_1, k_2 \in \mathbb{N}^+)$*, respectively, to the inequalities (3.1) and (3.4), then there exists a minimum common solution* (n_3, k_3) *where* $n_3 \geq \max\{n_1, n_2\}$*,* $k_3 \geq \max\{k_1, k_2\}$*, and* $n_3, k_3 \in \mathbb{N}^+$ *to the inequalities for both systems.*

Now, assume that the *RER* system $\langle E', R' \rangle$ correctly models the circuit structure in Figure 3.12 (as shown in Figure 3.16). For simplicity, we model the clocked reader with incoming edges from both transitions $clk\downarrow$ and $clk\uparrow$ to the reader because in our implementations, the enable signal changes from low to high at full clock cycle while we read the tokens at the rising edge of the clock. Define set $R'_1 \subset R'$ which includes two edges on the critical cycle, $R'_2 \subset R'$ which includes all the edges inside the async block (not shown in this Figure) and all the incoming edges to it, $R'_3 \subset R'$ includes all edges inside the k-stage FIFO block and all the incoming edges to it, and $R'_4 \subset R'$ includes the two edges labeled with delay $[D_1, D'_1]$ and $[D_2, D'_2]$.

There are several details that we are going to emphasize here. First, the delays on edges in Figure 3.16 are denoted by $[a, b]$, which means that each corresponding rule of the corresponding *ER* system can pick any delay value in the range $[a, b]$. For example, it is possible that $\hat{t}(\langle clk\uparrow, 2\rangle) = \hat{t}(\langle clk\downarrow, 2\rangle) + \beta_1$, but $\hat{t}(\langle clk\uparrow, 3\rangle) = \hat{t}(\langle clk\downarrow, 3\rangle) + \beta'_1$. Second, several previous assumptions still hold: In the worst case scenario when $r \in R'_1$ always pick the smallest possible delay values and $r \in R'_2$ always pick the largest possible delay values, the clock cycle period $p^\star$ is still the (only) critical cycle of the system. Similarly, we also assume that when

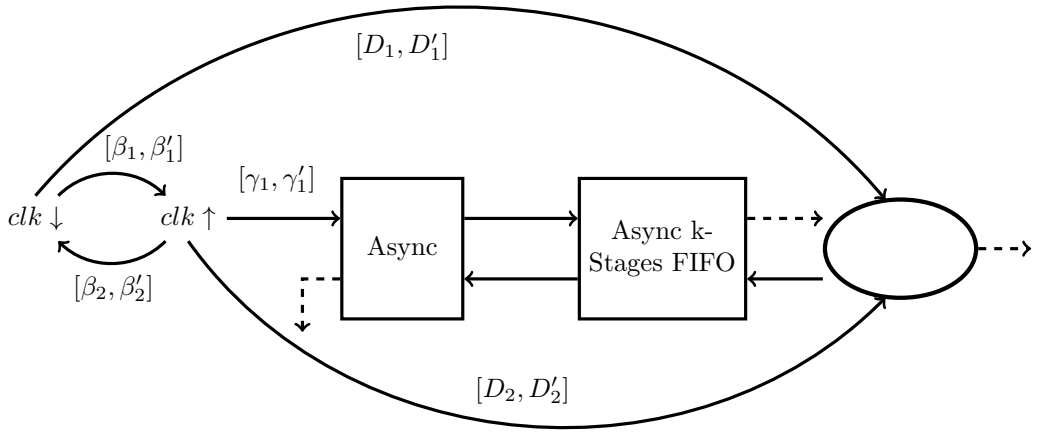

Figure 3.16: Synchronous to asynchronous metastability-free interface

$r \in R_3'$ always pick the largest possible delay values, the k-stage FIFO is still fast enough that it will not change $p^\star$ or perturb the timing of "Async" circuit. Third, the changes in delays on edges shown in dashed arrows will not result in metastability or FIFO overflow, because they do not have effects on any parameters which determine t_e' and k in (3.1) and (3.4).

Now, we are going to analyze how to determine the feasible solution (n, k) when delays of rules in each set change. In our implementation under fixed delay system, a token is removed exactly at the rising edge of the clock after n clock cycles, which means that $D_1 = D_1' = D_2 = D_2' = 0$. For simplicity, we will make these assumptions in all of the following analysis. It can be seen that effects of clock skew and nondeterministic delay of $r \in R_4'$ can be easily characterized by assigning appropriate values to $D_1, D_1', D_2, D_2', \gamma_1, \gamma_1'$ (and possibly the timing of the initial events to make sure that the values $D_1, D_1', D_2, D_2', \gamma_1, \gamma_1'$ are all nonnegative) and adding additional terms $\tilde{t}'s$ to $t_e' + (i + \frac{1}{2})p^\star$ in (3.1) and $t_u - (t_e' + \frac{1}{2}p^\star)$ in (3.4) to characterize delay changes on edges $r \in R_4'$. We will omit the details here because

it will be very similar to the analysis of delay changes of rules in R'_1, R'_2 and R'_3.

First, the following condition is an equivalent one to (3.4): For all $u \in \mathbb{N}^+$,

$$kp^\star - up^\star - (n + \frac{1}{2})p^\star \geq kd' - t_u. \tag{3.7}$$

Notice that $\frac{t_u - (t'_e + \frac{1}{2}p^\star)}{p^\star}$ in (3.4) is to find how many clock cycles elapsed in time $t_u - (t'_e + \frac{1}{2}p^\star)$ after the initial n and half clock cycles. Thus, $p^\star$ in $kp^\star - up^\star$ and $(n + \frac{1}{2})p^\star$, respectively, carries different physical meanings. In particular, delay increase by $\triangle$ on the edge of the corresponding *ER* system, $\langle clk\downarrow, n_c\rangle \overset{\delta(r_1)}{\mapsto} \langle clk\uparrow, n_c\rangle$, for example, will increase both $kp^\star$ and $up^\star$ (related to clock cycles after n and half initial ones) by $\triangle$, or $(n + \frac{1}{2})p^\star$ (related to initial n and half clock cycles) by $\triangle$, depending on whether $n_c > n$ or $n_c \leq n$. In the *RER* system, we denote the output transition of the asynchronous block to be f, and thus in the corresponding *ER* system, $\hat{t}(\langle f, u\rangle) = t_{u+1}$ (we denoted t_1 as the first token, but $\langle f, 0\rangle$ is the event of first occurrence of transition f).

Now, fix the delays of all rules except for those in R'_1. Without loss of generality, we increase the delay of an arbitrary edge (rule), $\langle clk\downarrow, n_c\rangle \overset{\delta(r_1)}{\mapsto} \langle clk\uparrow, n_c\rangle$ in the *ER* system, which corresponds to r_1 in the *RER* system, from some reference value $\delta(r_1) \in [\beta_1, \beta'_1]$, by $\triangle$, to $\delta(r_1) + \triangle \in [\beta_1, \beta'_1]$.

First, in (3.1), we can find a zero slack e-path p_i from some initial event $\langle g, 0\rangle$ to $\langle f, i\rangle$. Notice that a transition $\langle e, n_e\rangle$ with $n_e > i$ is never on p_i, which implies that for any $n_c > i$, t_{i+1} will not change. If $n_c \leq i$, the left-hand side of the inequality, the time after initial $n + i + \frac{1}{2}$ clock cycles, will increase by exactly $\triangle$, and by Theorem 4, the right-hand side of the inequality will increase by at most $\triangle$. Thus, the left-hand side of the inequality will always increase by at least the same amount as the right-hand side of the inequality, and $\delta(r_1) \equiv \beta_1$ for all $n_c \in \mathbb{N}$

will be the worst case scenario.

Then, in (3.7), $kp^\star$ and $up^\star$ will always change at the same time and by the same amount, which always results in a cancellation. Again by Theorem 4, the right-hand side of the inequality will decrease by at most $\triangle$. When $n_c \leq n$, the $(n + \frac{1}{2})p^\star$ will decrease by exactly $\triangle$, which implies a worst case scenario when $\delta(r_1) \equiv \beta_1'$ for all $n_c \leq n$. When $n_c > n$, $(n + \frac{1}{2})p^\star$ will remain the same, which implies a worst case scenario when $\delta(r_1) \equiv \beta_1$ for all $n_c > n$.

The same argument holds if we increase $\langle clk\uparrow, n_c - 1\rangle \overset{\delta(r_2)}{\mapsto} \langle clk\downarrow, n_c\rangle$ in the *ER* system, which corresponds to r_2 in the *RER* system. Define $p^\star_{min} = \beta_1 + \beta_2$, $p^\star_{max} = \beta_1' + \beta_2'$, and let d'_{max} correspond to the value of d' when each rule $r \in R_3'$ picks its maximum delay. If $2p^\star_{min} - p^\star_{max} \geq q'd'_{max}$, where $q' \gg 2$, which is typical in practical circuits, consider the system under the worst case scenario we discussed above, where $\delta(r_1) \equiv \beta_1'$, $\delta(r_2) \equiv \beta_2'$ for all $n_c \leq n_1$ and $\delta(r_1) \equiv \beta_1$, $\delta(r_2) \equiv \beta_2$ for all $n_c > n_1$. We can find a metastability-free interface solution (n_1, k_1) to it because when we increase n_1 by 1, t'_e will be increased by at least $p^\star_{min}$, and the total increase of delays of rules in the *ER* system will be $p^\star_{max} - p^\star_{min}$. Theorem 4 then guarantees that timing simulations of the output transition increase by at most $p^\star_{max} - p^\star_{min}$. Together with the similar idea to technical details we mentioned in the fixed delay system, we can always find n_1 and k_1 large enough that satisfy both (3.1) and (3.4) for the system.

Furthermore, $(n_1 + 1, k_1 + v)$ where $2 \leq v \leq q', v \in \mathbb{N}$ are all feasible solutions. Notice that Lemma 6 does not apply directly here because delays of rules depend on n_1. However, with the reasonable assumption we just made, similar result still holds. To see this, if (3.1) holds for (n_1, k_1), then for $(n_1 + 1, k_1 + v)$, $\delta(r_1) \equiv \beta_1'$, $\delta(r_2) \equiv \beta_2'$ for all $n_c \leq n_1 + 1$ and $\delta(r_1) \equiv \beta_1$, $\delta(r_2) \equiv \beta_2$ for all $n_c > n_1 + 1$.

The left-hand side of (3.1) increases by at least $p^\star_{min}$ and the right-hand side of it increases by at most $p^\star_{max} - p^\star_{min} + vd_{max}$ by Theorem 4, for all i, and the inequality still holds by the assumption. Similarly, (3.7) still holds as well because the left-hand-side of it increases by at least $vp^\star_{min} - p^\star_{max}$ and the right-hand side of it increases by at most vd'_{max} for all u.

If $\delta(r_1) \equiv \beta_1$, $\delta(r_2) \equiv \beta_2$ for all n_c, we can find another metastability-free interface solution (n_2, k_2). Corollary 6 applies here, and together with arguments above, we can find a common solution (n_3, k_3) to both systems. Thus, (n_3, k_3) is a solution to worst case scenario when delays of rules in the *ER* system corresponding to $r_1 \in R'_1$ are allowed to change.

Similar arguments can be made for all other edges, and it is also much easier than the analysis above because t_{i+1} (or d and d') are monotone increasing when we increase delays of rules corresponding to $r \in R'_2$ (or $r \in R'_3$), respectively, and other parameters will not change in either case. It is easy to check that when $r \in R'_2$, the worst case scenario for (3.1) is when each rule picks its maximum delay and that for (3.7) is when each rule picks its minimum delay. When $r \in R'_3$, the worst case scenario for both (3.1) and (3.7) is when each rule picks its maximum delay. A common solution to combinations of these worst case scenarios will be a valid solution for the bounded delay system.

Remark 6. *If q and q' are large, which is typical in practical systems, in Corollary 6, n_3 and k_3 of the minimum common solution (n_3, k_3) will not be much larger than $\max\{n_1, n_2\}$ and $\max\{k_1, k_2\}$, respectively, which is good for practical feasibility of our design under bounded delay assumptions.*

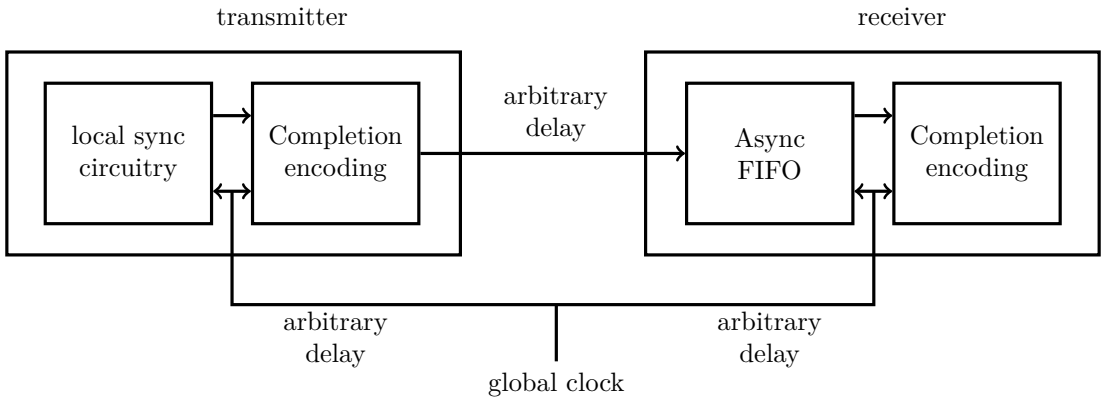

Figure 3.17: STARI communication

3.2.3 Existing Systems

"STARI", as is shown in Figure 3.17, is a novel technique for high-bandwidth communication proposed by Greenstreet [14]. It combines synchronous and self-timed design methods, and corresponds to a case in our design, where the "Async" component is a collection of wires. There are also at least two additional commercially developed systems that connect asynchronous components to a synchronous environment without any metastability. One of these systems is a digital FIR filter chip used in the read channel of a disk drive controller [52]. Another system is an asynchronous FPGA that provides a synchronous I/O interface and user model [31]. While both these systems relied on the empirical observation of periodicity of a specific class of asynchronous circuits, our results show that this is not a coincidence. The property of exact periodicity holds for a wide range of asynchronous circuits, and can be exploited for metastability-free interfaces between synchronous and asynchronous logic.

CHAPTER 4

CYCLONE ENGINE EVALUATION

4.1 Experimental Results

We implement Cyclone in C++ using the g++ 8.1 compiler, boost 1.67 libraries, and the Galois 5.0 framework for parallelization.[1] Cyclone can support multi-corner analysis, as well as extracted parasitics in the `.spef` file format. For the reported runtime, we use two corners (SSA and FFA), and ideal wire models. All experiments are conducted on a Linux machine with CentOS 7, 56 cores Intel Xeon Gold 5120 2.2GHz CPU and 187GB memory.

We study different configurations of asynchronous dataflow pipelines including cyclic structures and branching pipelines. The benchmark circuits are listed in Table 4.1. We use a small bundled-data benchmark (bd203) to show that Cyclone can analyze such circuits as well. Other circuits are converted automatically from synchronous benchmarks to their QDI analog (following [2]).

For each benchmark, we report (i) circuit properties: circuit name, number of gate pins, number of concurrent processes, number of timing constraints, $p^\star$, M, minimum correctness slack, and total power consumption; (ii) maximum cycle ratio runtime: best parallel runtime by YTO algorithm and best runtime by CPLEX, one of the fastest linear programming solver developed by IBM; and (iv) full static timing analysis (maximum cycle ratio + timing propagation + timing constraints

[1] This chapter is based on the "Experimental Results" section in our work [20].

Table 4.1: Analysis results of Cyclone. Large benchmarks are below the thick line.

Circuit Properties							
$Name$	$\#Pins$	$\#Proc$	$\#TC$	$p^{\star}(ns)$	M	$sl(ps)$	$P(mW)$
$bd203$	495	20	10	0.44	1	33.17	0.44
$s27$	817	31	164	2.02	1	21.27	0.22
$c2670$	22171	796	4540	1.65	5	21.48	7.54
$s1488$	37766	1502	9008	5.64	2	21.27	3.93
$c3540$	42772	1682	10016	7.50	1	21.48	3.34
$c7552$	60355	2278	13444	4.11	1	21.48	8.47
$c6288$	72621	2877	17260	7.90	2	21.48	5.49
$s5378$	88292	3595	20880	4.39	3	21.27	11.71
$s9234$	137723	5594	32724	7.82	1	21.27	10.31
wb_dma	212247	8593	49464	4.17	2	21.27	29.40
$tv80$	315219	12801	75352	9.10	2	21.27	20.3
$ac97\_ctrl$	650709	27215	154472	3.79	3	20.26	99.70
usb_funct	798895	32838	190020	8.17	1	21.27	57.06
$s38584$	807903	32975	192676	9.68	1	21.27	48.81
aes_core	1017817	40833	242616	8.61	1	21.27	69.49
vga_lcd	5689435	237059	1354068	7.05	1	21.27	473.20

checking) runtime: sequential and best runtime, and speedup. For best runtime, we also report the number of threads i used in parenthesis.

In full static timing analysis, the geometric mean of the best speedup across all benchmarks is 3.93, and that across large benchmarks ($> 200K$ pins) is 6.90.

By comparing columns 9 and 12 in Table 4.2, we notice that YTO algorithm always takes up a significant fraction of time in full static timing analysis after parallelization; for large benchmarks, it takes 35% to 70%. This is because timing propagation can be parallelized as discussed in Section V, but edge-swapping in YTO algorithm [53] is inherently sequential. Hence, it is reasonable that Cyclone achieves 6.90× speedup on average in full static timing analysis for large benchmarks.

When calculating the maximum cycle ratio, CPLEX runs as fast as YTO al-

Table 4.2: Analysis runtime of Cyclone. Large benchmarks are below the thick line.

Circuit Scales		Maximum Cycle Ratio		Full Static Timing Analysis		
$Name$	$\#Pins$	$YTO(i)(s)$	$LP(s)$	$seq(s)$	$best(i)(s)$	sp
$bd203$	495	0.01(1)	<0.01	0.03	0.03(1)	1.00
$s27$	817	<0.01(1)	0.01	0.02	0.02(1)	1.00
$c2670$	22171	0.44(1)	0.69	1.69	0.86(7)	1.97
$s1488$	37766	0.23(28)	1.80	3.09	0.60(28)	5.15
$c3540$	42772	0.32(21)	1.53	1.96	0.59(14)	3.32
$c7552$	60355	0.95(14)	2.01	2.87	1.24(14)	2.31
$c6288$	72621	1.95(14)	2.80	7.93	2.63(14)	3.02
$s5378$	88292	0.97(14)	2.83	9.10	2.01(14)	4.53
$s9234$	137723	0.81(28)	5.23	6.71	1.63(28)	4.12
$wbdma$	212247	2.08(21)	12.82	14.49	3.64(21)	3.98
$tv80$	315219	7.17(21)	18.41	65.20	11.06(21)	5.90
$ac97ctrl$	650709	8.87(21)	79.93	102.53	15.54(35)	6.60
$usbfunct$	798895	3.87(42)	96.49	58.02	9.09(35)	6.38
$s38584$	807903	3.92(56)	79.39	51.07	9.58(35)	5.33
$aescore$	1017817	4.76(49)	74.44	95.30	12.62(42)	7.55
$vgalcd$	5689435	105.48(56)	2959.01	2994.24	159.58(56)	18.76

gorithm for very small benchmarks, but much slower for large benchmarks. For example, when running on "vga_lcd" (with around 5.69M pins and 18.08M linear constraints), CPLEX can be more than 28× slower than YTO algorithm, since CPLEX is a general purpose linear programming solution package, but YTO algorithm is designed specifically for maximum cycle ratio problem.

Note that the runtime and speedup are not only related to the scale of the circuit, but also related to the circuit topology and the structure of the critical path. For large values of M, the effective number of pins is multiplied by M since there are M different times associated with each signal transition. This increases the time for the delay propagation phase accordingly.

CHAPTER 5

FAST ALGORITHM ON CYCLE TIME OF MIN-MAX SYSTEMS

5.1 OR-Causality Extensions

The current implementation of Cyclone Timing and Power Engine assumes AND-causality, which gives conservative timing and power analysis for asynchronous circuit. In this model, when there are multiple rules (timing constraints) have the same target event, the event can occur only when all of the constraints are satisfied, and the max operator is used when defining the timing simulation of events (see chapter 2). In real circuits, some events can occur when only one of the timing constraints is satisfied. It is more accurate to model this kind of behavior with min operator when calculating the actual time of events. The timing of events of a general system should have both min and max relations.

5.2 Cycle Time of Min-Max Systems

Lee first introduced extended repetitive event-rule (*XRER*) system in [28], which considered both max and min relations of timing. It turns out that the cycle time of the *XRER* system is a good indication of the system performance. The papers [16, 15] also studied the properties of min-max functions, a closely related problem in mathematics literature. Unfortunately, as we mentioned in the chapter 1, there is no known polynomial algorithm on cycle time of *XRER* systems in the

literature.

In this chapter, we consider the following equivalent version of the described problem: cycle time of min-max systems. We will begin with the definition of min-max systems and its cycle time, and provide a brand new algorithm on this problem. The algorithm was originally illuminated by Karp and Orlin's Algorithm on maximum cycle ratio [25], the special case of the problem. We will prove that our algorithm terminates and returns the correct result as desired, and show that it runs reasonably fast in practice.

Definition 11 (min-max system)**.** *A min-max system is a connected, weighted, ticked and directed graph $G = (V, E, w, t)$, where V is a non-empty set of vertices, E is a set of edges, $w\colon E \to \mathbb{R}$ assigns a weight to each edge, and $t\colon E \to \{0, 1\}$ assigns a tick value to each edge. The vertices are partitioned into min-vertices V_{min} and max-vertices V_{max}, and the edges are partitioned into sets $E_{max} \subseteq V \times V_{max}$ and $E_{min} \subseteq V \times V_{min}$. For an edge $e = (u, v) \in E$, the source $src(e) = u$ and target $tar(e) = v$.*

Without loss of generality, we make the following simplifying assumptions regarding min-vertices: (i) $t(e) = 0$ for all $e \in E_{min}$—this constraint can be accommodated by introducing a new max-vertex for any edge that violates this assumption and splitting the edge; (ii) the in-degree of any vertex in E_{min} is at least two—otherwise the vertex can be viewed as a max-vertex; and (iii) $E_{min} \subseteq V_{max} \times V_{min}$, i.e. every edge to a min-vertex is from a max-vertex—this can be accommodated by edge splitting as in (a); and (iv) there is at least one vertex in V_{min}.

Definition 12 (strategy)**.** *A strategy $\sigma_l = (V, E_l, w, t) \subset G$ is a subgraph of G obtained by selecting all the edges from E_{max}, and a subset of E_{min} such that every min-vertex has in-degree one. If G does not have $v \in V_{min}$, then we say that G only*

has one strategy $\sigma_0 = G$. A strategy can be switched to another strategy by deleting and inserting edges whose target are in V_{min}. A strategy switching step at $v \in V_{min}$ corresponds to deleting the edge with target v from the strategy, followed by inserting a different edge from E_{min} whose target is also v. The distance $d(\sigma_a, \sigma_b) \leq |V_{min}|$ is the minimum number of such steps needed to switch the strategy σ_a to σ_b.

Definition 13 (path)**.** *A path p consists of a set of consecutive vertices and edges starting at vertex v_0 and ending at vertex v_n, which is denoted as $p = (v_0, v_1, ...v_n)$ where $v_i \in V$ for all $i \leq n, i \in \mathbb{N}$. The length of a path p (denoted $l(p)$) is the total number of edges on that path. The sum of the weights and ticks of all the edges on the path p is denoted by $w(p)$ and $t(p)$, respectively. If the final vertex of path p_1 matches the first vertex of path p_2, then $p_1 + p_2$ denotes the path obtained by appending p_2 to p_1.*

For the following discussions, we add a single special vertex s to G such that $V = V_{max} \cup V_{min} \cup \{s\}$, and an edge $e = (s, v)$ for each $v \in V_{max}$. For each edge $e = (s, v)$, we set the weight $w(e) = 0$ and transit time $t(e) = 0$. Note that adding s does not introduce any additional cycles, and thus will not affect properties related to cycles, including maximum cycle ratio and cycle time introduced in the definitions below. We remark that there will always be a path from s to $v \in V$ in any strategy $\sigma_l \subset G$ because: (i) there is an edge from s to every $v \in V_{max}$ by construction, and (ii) for every $v \in V_{min}$, there is an edge from some $u \in V_{max}$ to v in every σ_l.

The set of paths from s to $v \in V$ in σ_l are denoted by $\pi(v, \sigma_l)$. If we consider a spanning tree T of G rooted at s, then the cardinality of $\pi(v, T)$ will always be one; in this case, we use the notation $p(v, T)$ to denote the unique element of $\pi(v, T)$.

Definition 14 (simple cycle)**.** *A path $p = (v_0, v_1, ...v_n)$ is a simple cycle C if $v_i \neq v_j$ for all $0 \leq i, j \leq n, i \neq j$, except $v_0 = v_n$ and $(v_i, v_{i+1}) \in E$ for all $0 \leq i < n$. We use $cyc(G)$ to denote the set of simple cycles in G.*

We assume that $t(C) > 0$ for all $C \in cyc(G)$, and hence also for any $C \in cyc(\sigma_l)$ for any strategy $\sigma_l \subset G$.

Definition 15 (maximum cycle ratio)**.** *The maximum cycle ratio of a strategy σ_l is defined to be $\hat{\lambda}(\sigma_l) = \max_{C \in cyc(\sigma_l)} \frac{w(C)}{t(C)}$. If there is no cycle in σ_l, we define $\hat{\lambda}(\sigma_l) = -\infty$.*

Definition 16 (cycle time)**.** *The cycle time of the graph G is defined to be*

$$\lambda^\star(G) = \min_l \hat{\lambda}(\sigma_l) = \min_l \max_{C \in cyc(\sigma_l)} \frac{w(C)}{t(C)}. \tag{5.1}$$

Definition 17 (parametric graph)**.** *Given a graph G, its parametric version $G(\lambda)$ is the weighted, directed graph $G(\lambda) = (V, E, w_\lambda)$ where $\lambda \in \mathbb{R}$ is called the parameter. The weight function w_λ is given by $w_\lambda(e) = w(e) - \lambda t(e)$. For the following discussions, we will consider both the parametric graph and its corresponding original graph frequently, and if we provide the definition of one, we will omit the definition of the other. When $\sigma_l \subset G$, their corresponding parametric graphs also satisfy $\sigma_l(\lambda) \subset G(\lambda)$.*

Corollary 7. *$\lambda = \hat{\lambda}(\sigma_l)$ if and only if there exists $C_m \in cyc(\sigma_l)$ such that $w_\lambda(C_m) = 0$ and for all $C \in cyc(\sigma_l)$, $w_\lambda(C) \leq 0$.*

Proof. If $\lambda = \hat{\lambda}(\sigma_l) = \max_{C \in cyc(\sigma_l)} \frac{w(C)}{t(C)}$, let $C_m = arg(\max_{C \in cyc(\sigma_l)} \frac{w(C)}{t(C)}) \in cyc(\sigma_l)$, then $\lambda = \frac{w(C_m)}{t(C_m)}$, which gives $w_\lambda(C_m) = 0$. Furthermore, for all $C \in cyc(\sigma_l)$, $\lambda \geq \frac{w(C)}{t(C)}$ by definition, which gives $w_\lambda(C) \leq 0$ for all $C \in cyc(\sigma_l)$ as desired.

If there exists $C_m \in cyc(\sigma_l)$ such that $w_\lambda(C_m) = 0$ and for all $C \in cyc(\sigma_l)$, $w_\lambda(C) \leq 0$, then $C_m = arg(\max_{C \in cyc(\sigma_l)} \frac{w(C)}{t(C)})$ and by definition, $\hat{\lambda}(\sigma_l) = \frac{C_m}{t_m}$. Thus, $w_{\hat{\lambda}(\sigma_l)}(C_m) = 0$, which gives $\lambda = \hat{\lambda}(\sigma_l)$ as desired. □

Definition 18 (max (longest) path)**.** *A longest path from p_0 from s to v in $\sigma_l(\lambda)$ is a path that satisfies*

$$w_\lambda(p_0) = w(p_0) - \lambda t(p_0) = \max_{p \in \pi(v, \sigma_l(\lambda))} w_\lambda(p) = \hat{w}_\lambda(v, \sigma_l(\lambda)). \tag{5.2}$$

There can be multiple longest paths that have the same parametric weight. To avoid ambiguity, we assume that there is an ordering on paths (to be defined later) that can be used to break ties. We define $\hat{p}(v, \sigma_l(\lambda))$ to be the unique longest path with maximum number of ticks from s to v in $\sigma_l(\lambda)$. If $\sigma_l(\lambda)$ has a non-negative weight cycle, then some longest paths from s to v in $\sigma_l(\lambda)$ will have infinite weight or infinite number of ticks. We say that a strategy $\sigma_l(\lambda)$ is well defined if and only if all (simple) cycles in $\sigma_l(\lambda)$ are negative and $\hat{p}(v, \sigma_l(\lambda))$ is well defined iff $w(\hat{p}(v, \sigma_l(\lambda))) = \hat{w}_\lambda(v, \sigma_l(\lambda)) < \infty$ and $t(\hat{p}(v, \sigma_l(\lambda))) < \infty$.

Corollary 8. *$\sigma_l(\lambda)$ is well defined if and only if $\hat{p}(v, \sigma_l(\lambda))$ is well defined (and acyclic) for all $v \in V$.*

Corollary 9. *If $\sigma_l(\lambda')$ is well-defined, $\sigma_l(\lambda)$ will be well defined for all $\lambda \geq \lambda'$.*

Corollary 10. *If $\sigma_l(\lambda)$ is not a well defined strategy, $\lambda \leq \hat{\lambda}(\sigma_l)$.*

Lemma 7 (sub-path optimality)**.** *Assume that $p(v, \sigma_l)$ is a longest path from s to v in σ_l, then any sub-path from $p(u, \sigma_l) \subset p(v, \sigma_l)$ from s to u where u is on $p(v, \sigma_l)$ is a longest path from s to u in σ_l.*

Proof. If otherwise, assume that u is the vertex closest to v such that $p(u, \sigma_l)$ is not a longest path from s to u, and we know that $p(v, \sigma_l) = p(u, \sigma_l) + p_0$ where

$p_0 \subset p(v, \sigma_l)$ is the path from u to v. Then there exists another path $p'(u, \sigma_l)$ such that $w_\lambda(p'(u, \sigma_l)) > w_\lambda(p(u, \sigma_l))$. Thus, we find a path $p_1 = p'(u, \sigma_l) + p_0$ from s to v in σ_l such that $w_\lambda(p_1) = w_\lambda(p'(u, \sigma_l)) + w_\lambda(p_0) > w_\lambda(p(u, \sigma_l)) + w_\lambda(p_0) = w_\lambda(p(v, \sigma_l))$, a contradiction to the assumption that $p(v, \sigma_l)$ is a longest path from s to v in σ_l. □

Definition 19. *It is straightforward that a spanning tree T of σ_l must be rooted at s. A tree T is a longest path tree in $\sigma_l(\lambda)$ iff it is a spanning tree of σ_l rooted at s, and for all $v \in V, v \neq s$, the corresponding path $p(v, T)$ is a longest path in $\sigma_l(\lambda)$. In other words,*

$$\forall v \in G : w_\lambda(p(v, T)) = \hat{w}_\lambda(v, \sigma_l(\lambda)) \tag{5.3}$$

Theorem 5. *Given a well defined strategy $\sigma_l(\lambda)$, there is always a longest path tree T in $\sigma_l(\lambda)$. T can be constructed in $\mathcal{O}(|V||E|)$ time using Bellman-Ford Algorithm.*

5.2.1 Min-max paths and trees

We now consider the paths that result from considering longest path trees across all strategies.

Definition 20 (min-max path)**.** *The path p_0 from s to v in $G(\lambda)$ is called a min-max path of $G(\lambda)$ (corresponding to strategy $\sigma_{min}(\lambda) \subset G(\lambda)$), iff it is a longest path in $\sigma_{min}(\lambda)$, and for all well-defined $\sigma_l(\lambda) \subset G$,*

$$w_\lambda(\hat{p}(v, \sigma_{min}(\lambda))) \leq w_\lambda(\hat{p}(v, \sigma_l(\lambda))). \tag{5.4}$$

That is,

$$\begin{aligned} c^\star_\lambda(v) = w_\lambda(\hat{p}(v, \sigma_{min}(\lambda))) &= \min_l w_\lambda(\hat{p}(v, \sigma_l(\lambda))) \\ &= \min_l \max_{p \in \pi(v, \sigma_l(\lambda))} w_\lambda(p). \end{aligned} \tag{5.5}$$

The min-max path for a $v \in V$ in $G(\lambda)$ is well defined if and only if at least one of the strategies $\sigma_l(\lambda) \subset G(\lambda)$ is well defined. If at least one of the strategies $\sigma_l(\lambda) \subset G(\lambda)$ is well-defined, there exists a well defined min-max path from s to v in $\sigma_l(\lambda)$ that is acyclic.

Definition 21 (min-max tree)**.** *It is straightforward that a spanning tree T of G must be rooted at s. Furthermore, T must be a subgraph of a unique strategy of G because every $v \in T$ has in-degree one; we use $\sigma_T \subset G$ to denote the unique strategy corresponding to T. A tree T is a min-max tree of $G(\lambda)$ iff it is a spanning tree of G rooted at s corresponding to some strategy $\sigma_{min} \subset G$ and for all $v \in V, v \neq s$, the corresponding path $p(v, T)$ is a min-max path. In other words,*

$$\forall v \in G : w_\lambda(p(v, T)) = c^\star_\lambda(v) \tag{5.6}$$

We will also need to study paths whose length is bounded. To this end, we examine longest paths of length $\leq k$.

Definition 22. *A max (longest) path from s to v with length $\leq k$ for strategy $\sigma_l(\lambda)$ is denoted as $\hat{p}(v, \sigma_l(\lambda), k)$. If there is no such path, we define $w_\lambda(\hat{p}(v, \sigma_l(\lambda), k)) = -\infty$. We also define $w_\lambda(\hat{p}(s, \sigma_l(\lambda), 0)) = 0$, and $w_\lambda(\hat{p}(v, \sigma_l(\lambda), 0)) = -\infty$ for all $v \neq s, v \in V$.*

5.2.2 Acyclic Min-Max Algorithm

By deleting every edge e satisfying $t(e) > 0$ from G, we obtain an acyclic subgraph $G' = (V, E', w, t)$ of G ($V' = V'_{max} \cup V'_{min}, E' = E'_{max} \cup E'_{min}$). This is a consequence of assuming that all cycles c of G have $t(c) > 0$, and t is a non-negative function. Since all edges e' in G' have $t(e') = 0$, we have $w(e') = w_\lambda(e') = w(e') - \lambda t(e')$

for $\lambda \in \mathbb{R}$; thus $G' = G'(\lambda)$ for $\lambda \in \mathbb{R}$. Notice that there is still a path from s to $v \in G'$ for every $v \in G', v \neq s$ because there is an edge from s to $v \in V_{max}$, and all edges e_{min} from E_{min} have $t(e_{min}) = 0$ and hence are in G'. Also, for all paths $p \subset G'$, we have $w(p) = w(p) - \lambda t(p) = w(p)$.

Definition 23 (s-distance)**.** *The s-distance of $v \in V$ in graph G', $D(v, G')$, is defined as: $D(v, G') = \max_{p \in G'}\{l(p(s, v))\}$. In particular, $D(s, G') = 0$, and $D(v, G') \leq |V|$ for all v.*

Assume that $v_0, v_1, ...v_n$ is a sequence of all the vertices $v \in V$ such that $D(v_i, G') \leq D(v_j, G')$ for all $0 \leq i < j \leq n$, $i, j \in \mathbb{N}$.

Definition 24 (Acyclic Min-Max Algorithm)**.** *We define $C(v, k)$ for each $v \in V$ by the following recursive relation: Initially, let $C(s, 0) = 0$, and $C(v, 0) = \perp$ (not defined) for $v \in V$, $v \neq s$. For all $k \leq |V|, k \in \mathbb{N}, v \in V$, if $v = s$, $C(v, k + 1) = C(s, k + 1) = 0$, and if $v \neq s$, define*

$$C(v, k+1) = \begin{cases} \max_{(v_j,v)\in E', C(v_j,k)\neq\perp}\{C(v_j, k) + w(v_j, v)\} & \text{for } v \in V_{max} \\ \min_{(v_j,v)\in E', C(v_j,k)\neq\perp}\{C(v_j, k) + w(v_j, v)\} & \text{for } v \in V_{min} \end{cases}$$

with the convention that if the min/max sets are empty, they evaluate to $\perp$.

The node key of $v \in V, v \neq s$: $nk(v, k)$ ($k \in \mathbb{N}$, $nk(v, 0) = \perp$ (not defined) for all $v \in V$) as source vertex of the edge that gives the maximum(minimum) values in the equations above. In other words, for all $k \in \mathbb{N}$,

$$nk(v, k+1) = \tag{5.7}$$

$$\begin{cases} nk(v, k) & C(v, k+1) = C(v, k) \\ src(\arg\max_{(v_j,v)\in E', C(v_j,k)\neq\perp}\{C(v_j, k) + w(v_j, v)\}) & C(v, k+1) \neq C(v, k), v \in V_{max} \\ src(\arg\min_{(v_j,v)\in E', C(v_j,k)\neq\perp}\{C(v_j, k) + w(v_j, v)\}) & C(v, k+1) \neq C(v, k), v \in V_{min} \end{cases}$$

We have constructed $nk(v,k)$ so that it only changes from k to $k+1$ if the function $C(v,k+1)$ differs from $C(v,k)$.

Lemma 8. *For all $k \leq |V|, k \in \mathbb{N}$, if $D(v,G') \leq k, v \in V$, $C(v,i) = C(v,i+1) \neq \perp$ for all $k \leq i \leq |V|, i \in \mathbb{N}$.*

Proof. We prove the statement by induction on k. For the base case, $k = 0$, if $D(v,G') \leq 0$, then $v = s$, and $C(s,i) = C(s,i+1) = 0 \neq \perp$ for all $0 \leq i \leq |V|, i \in \mathbb{N}$ by definition. Let $k = m \leq |V| - 1, k \in \mathbb{N}$, assume that the statement is true. When $1 \leq k = m+1 \leq |V|, k \in \mathbb{N}$, if $D(v,G') \leq m+1, v \in V$, we first consider the case when $D(v,G') \leq m, v \in V$, and $C(v,i) = C(v,i+1) \neq \perp$ for all $m \leq i \leq |V|, i \in \mathbb{N}$ by the induction hypothesis. If $1 \leq D(v,G') = m+1, v \in V$, then $v \neq s$, and for all $(v_j, v) \in E'$, $D(v_j,G') \leq m, v_j \in V$, and $C(v_j,i) = C(v_j,i+1) \neq \perp$ for all $m \leq i \leq |V|, i \in \mathbb{N}$ by the induction hypothesis, and $C(v,i) = C(v,i+1) \neq \perp$ for all $m+1 \leq i \leq |V|, i \in \mathbb{N}$ directly by Definition 24 as desired. □

Since $D(v,G') \leq |V|$ for all $v \in V$, the following Corollary directly follows:

Corollary 11. *$v \in V, v \neq s$, $nk(v,|V|) \neq \perp$ and $C(v,|V|) = C(v,|V|+1) \neq \perp$.*

Consider graph $T_0 = (V, E_0)$ where $E_0 = \{(nk(v,|V|), v) | v \in V\}$.

Corollary 12. *T_0 is acyclic and the in-degree of each $v \in V$ in T_0 is 1.*

Proof. By definition, for each $e \in E_0$, $e \in E'$, and then T_0 is acyclic because G' is acyclic. The in-degree of each $v \in V$ in T_0 is 1 because $nk(v,|V|)$ exists by Corollary 11 together with the fact that it is unique. □

We start from an arbitrary $v_0 \in V$, and for all $k \in \mathbb{N}, k \leq |V|, v_k \in V, v_k \neq s$, define $v_{k+1} = nk(v_k, |V|)$, we will get $v_0, v_1, ..., v_n$ $(n \in \mathbb{N}, n \leq |V|)$.

Corollary 13. $v_n = s$.

Proof. If otherwise, the definition implies that $n = |V|$, and $(v_{k+1}, v_k) \in T_0$ for all $k \in \mathbb{N}, k \leq n-1$, and we must have $v_a = v_b$ for some $0 \leq a < b \leq |V|, a, b \in \mathbb{N}$, a contradiction to the fact that T_0 is acyclic. □

Lemma 9. *T_0 is a spanning tree of $G(\lambda)$ rooted at s.*

Proof. By Corollary 13, there is a path from s to v in T_0 for all $v \in V$. Together with the fact that the in-degree of each $v \in V$ in T_0 is 1 by Corollary 12, we know that T_0 is a spanning tree of G' (and also $G(\lambda)$ for all $\lambda \in \mathbb{R}$ since G' and $G(\lambda)$ have the same vertices and $G' \subset G(\lambda)$). □

Lemma 10. *For all $v \in V$, $w(v, T_0) = C(v, |V|)$.*

Proof. Let $v = v_0$ be an arbitrary vertex in V and consider the sequence $v_0, v_1, ...v_n = s$ constructed above. By definition and Corollary 11, we have $C(v_0, |V|) = C(v_0, |V|+1) = C(nk(v_0, |V|), |V|) + w(nk(v_0, |V|), v_0) = C(v_1, |V|) + w(v_1, v_0) = C(v_1, |V|+1) + w(v_1, v_0)$. By similarly recursive substitution, we will have

$$\begin{aligned} C(v_0, |V|) &= C(v_0, |V|+1) \\ &= C(v_n, |V|+1) + w(v_1, v_0) + ... + w(v_n, v_{n-1}) \\ &= C(s, |V|+1) + w(v, T_0) = w(v, T_0) \end{aligned} \tag{5.8}$$

as desired. □

Theorem 6. *Let $\lambda_0 = 1 + \sum_{e \in E} |w(e)|$. Then*

$$\begin{cases} w_{\lambda_0}(u, T_0) + w_{\lambda_0}(u, v) - w_{\lambda_0}(v, T_0) \leq 0 & \text{for } (u, v) \in E_{max} \\ w_{\lambda_0}(u, T_0) + w_{\lambda_0}(u, v) - w_{\lambda_0}(v, T_0) \geq 0 & \text{for } (u, v) \in E_{min} \end{cases}$$

Proof. Since we assumed that every edge in E_{min} has zero tick, and G' is got by deleting all ticked edges in G, we know that if $E'_{min} = E_{min}$.

If $(u, v) \in E'_{min} = E_{min}$, $t(u, v) = 0$ and

$$\begin{aligned} C(v, |V|) &= C(v, |V| + 1) \\ &= \min_{(v_j, v) \in E', C(v_j, |V|) \neq \perp} \{C(v_j, |V|) + w(v_j, v)\} \\ &\leq C(u, |V|) + w(u, v). \end{aligned} \tag{5.9}$$

By Lemma 10, together with fact that all paths in $T_0 \subset G'$ have zero ticks, we have

$$\begin{aligned} 0 &\leq w(u, T_0) + w(u, v) - w(v, T_0) \\ &= w_{\lambda_0}(u, T_0) + w_{\lambda_0}(u, v) - w_{\lambda_0}(v, T_0) \end{aligned} \tag{5.10}$$

as desired.

If $(u, v) \in E'_{max}$, by definition, $t(u, v) = 0$ and

$$\begin{aligned} C(v, |V|) &= C(v, |V| + 1) \\ &= \max_{(v_j, v) \in E', C(v_j, |V|) \neq \perp} \{C(v_j, |V|) + w(v_j, v)\} \\ &\geq C(u, |V|) + w(u, v). \end{aligned} \tag{5.11}$$

By Lemma 10, together with fact that all paths in $T_0 \subset G'$ have zero ticks, we have

$$\begin{aligned} 0 &\geq w(u, T_0) + w(u, v) - w(v, T_0) \\ &= w_{\lambda_0}(u, T_0) + w_{\lambda_0}(u, v) - w_{\lambda_0}(v, T_0) \end{aligned} \tag{5.12}$$

as desired. If $(u, v) \in E_{max} - E'_{max}$, by definition, (u, v) has one tick, and

$$\begin{aligned} & w_{\lambda_0}(u, T_0) + w_{\lambda_0}(u, v) - w_{\lambda_0}(v, T_0) \\ &= w(u, T_0) + w(u, v) - \lambda_0 - w(v, T_0) \\ &= \sum_{e \in p_1, e \notin p_2} w(e) - \sum_{e \in p_2, e \notin p_1} w(e) - \lambda_0 \\ &\leq \sum_{e \in p_1, e \notin p_2} |w(e)| + \sum_{e \in p_2, e \notin p_1} |w(e)| - \lambda_0 \\ &\leq \sum_{e \in E} |w(e)| - \lambda_0 = -1 < 0 \end{aligned} \tag{5.13}$$

where $p_1 = p(u, T_0) + p(u, v)$ and $p_2 = p(v, T_0)$ are two paths from s to v in $G(\lambda_0)$ as desired. □

Remark 7. *Computing T_0 of $G(\lambda_0)$ where $\lambda_0 = M$ takes $\mathcal{O}(|V||E|)$ time as indicated in Definition 24. Since the graph is acyclic, the time complexity of computing T_0 can be further reduced to $\mathcal{O}(|E|)$ with the help of topological sort of G', which will be very important to make the algorithm scalable in practice. We call $w_{\lambda_0}(u, T_0) + w_{\lambda_0}(u, v) - w_{\lambda_0}(v, T_0) \leq 0$ the max vertex inequality and $w_{\lambda_0}(u, T_0) + w_{\lambda_0}(u, v) - w_{\lambda_0}(v, T_0) \geq 0$ min vertex inequality.*

5.2.3 λ Progression

We have constructed a spanning tree of G: T_0 that satisfies both max vertex inequality and min vertex inequality for all $v \in V, v \neq s$ for $\lambda = \lambda_0$. Now we examine what happens as we decrease the value of λ.

Definition 25 (edge key)**.** *Given a spanning tree T_0 of $G(\lambda_0)$ that satisfies both max vertex inequality and min vertex inequality, and edges $e = (u, v) \in E$, define T_{i+1} by performing λ Progression on T_i for all $i \in \mathbb{N}$ as following:*

1. *Define two values:*

$$\Delta w(e, T_i) = w(u, T_i) + w(u, v) - w(v, T_i) \tag{5.14}$$

and

$$\Delta t(e, T_i) = t(u, T_i) + t(u, v) - t(v, T_i) \tag{5.15}$$

2. *Define the edge keys $K(e, T_i)$ of all the edges $e = (u, v) \in E$ as following:*

$$K(e, T_i) = \begin{cases} -\infty & \Delta t(e) \leq 0, v \in V_{max} \\ -\infty & \Delta t(e) \geq 0, v \in V_{min} \\ \frac{\Delta w(e,T_i)}{\Delta t(e,T_i)} & otherwise \end{cases}$$

3. *Define*

$$\lambda_{i+1} = \max\{K(e, T_i)\}; \tag{5.16}$$

if $\lambda_{i+1} = -\infty$, return $-\infty$.

4. *Let $e^\star = (u', v')$ be the edge that gives the maximum value of $K(e^\star, T_i)$. If there are multiple candidates for $e^\star$, pick an edge whose target is in V_{min} if possible. Define T_{i+1} as the graph obtained by deleting the edge $e = (u, v')$ in T_i and replacing it with (u', v').*

5. *Define*

$$w(v, T_{i+1}) = \begin{cases} w(v, T_i) + \Delta w(e^\star, T_i) & v \in T_i^s(v') \\ w(v, T_{i+1}) = w(v, T_i) & v \notin T_i^s(v') \end{cases} \tag{5.17}$$

$$t(v, T_{i+1}) = \begin{cases} t(v, T_i) + \Delta t(e^\star, T_i) & v \in T_i^s(v') \\ t(v, T_i) & v \notin T_i^s(v') \end{cases} \tag{5.18}$$

where $T_i^s(v')$ denotes the subtree of T_i rooted at v' (the updated vertices)

6. *If T_{i+1} is a spanning tree of G, and T_{i+1} is different from T_j for all $j \leq i, j \in \mathbb{N}$ (in later section of the proof, we will show that it is always the case and there is no need to check the repetition of trees), repeat the λ Progression on T_{i+1}; if otherwise, stop and we will perform Cycle Break Steps on T_i as instructed later.*

The following Corollary follows directly from the definition of the spanning tree:

Corollary 14. *If T_i is a spanning tree of G, T_{i+1} is also a spanning tree of G if and only if $u' \notin T_i^s(v')$.*

Given a spanning tree of G: T_0 that satisfies both max inequality and min inequality and perform λ Progression as described above. Here we assume that λ Progression never returns $-\infty$ (see the remark in the end of this section). We will have distinct spanning trees of G: $T_0, T_1, ..., T_k, ...$ in $\sigma_{T_0}, \sigma_{T_1}, ..., \sigma_{T_k}, ...$, associated with $\lambda_0, \lambda_1, ..., \lambda_k, \lambda_{k+1}, ...$ $(k \in \mathbb{N})$.

Lemma 11. *If $\lambda_{a+1} \neq \lambda_{b+1}$ $(0 \leq a < b \leq k)$, then $T_a \neq T_b$*

Proof. We prove the contrapositive of the statement. If T_a and T_b are two spanning trees of G such that $T_a = T_b$, then $\lambda_{a+1} = \lambda_{b+1} = \max\{K(e, T_a)\}$ directly by definition. □

Lemma 12. $k < 2^{|E|}$.

Proof. If otherwise, we will have distinct spanning trees of G: $T_0, T_1, ..., T_{2^{|E|}}$, a contradiction because there are at most $2^{|E|}$ different subgraphs containing all the vertices in V, and thus at most $2^{|E|}$ different spanning trees of G. □

Thus, we can assume that when the λ Progression stops, we have distinct spanning trees of G: $T_0, T_1, ..., T_c$ in $\sigma_{T_0}, \sigma_{T_1}, ..., \sigma_{T_c}$, associated with λ_0, λ_1, ..., λ_c, λ_{c+1}, ... ($c \in \mathbb{N}, c < 2^{|E|}$).

Lemma 13. *If T_i is a tree in σ_{T_i} such that for all $(u, v) \in E_{max}$, $w_\lambda(u, T_i) + w_\lambda(u, v) - w_\lambda(v, T_i) \leq 0$, then $\sigma_{T_i}(\lambda)$ does not have a positive simple cycle. In particular, if whenever $w_\lambda(u, T_i) + w_\lambda(u, v) - w_\lambda(v, T_i) = 0$, $t(u, T_i) + t(u, v) - t(v, T_i) \leq 0$, then $\sigma_{T_i}(\lambda)$ has all negative simple cycles.*

Proof. Given an arbitrary simple cycle in $\sigma_{T_i}(\lambda)$: $C = (v_0, ..., v_n = v_0)$ ($n \in \mathbb{N}^+, n \leq |V|$). Then by Lemma 16, for all $k \in \mathbb{N}^+, k \leq n, v_k \in V_{max}$, $w_\lambda(v_k, T_i) \geq w_\lambda(v_{k-1}, T_i) + w_\lambda(v_{k-1}, v_j)$ for all $k \in \mathbb{N}^+, j \leq n$; for all $k \in \mathbb{N}^+, k \leq n, v_k \in V_{min}$, (v_{k-1}, v_k) is the unique edge in strategy σ_{T_i}, and thus $w_\lambda(v_k, T_i) = w_\lambda(v_{k-1}, T_i) + w_\lambda(v_{k-1}, v_j)$. By recursive substitution, we would have:

$$\begin{aligned} w_\lambda(v_n, T_i) &\geq w_\lambda(v_0, T_i) + w_\lambda(v_0, v_1) + ... + w_\lambda(v_{n-1}, v_n) \\ &= w_\lambda(v_n, T_i) + w(C), \end{aligned} \tag{5.19}$$

which gives $w(C) \leq 0$. Thus, $\sigma_{T_i}(\lambda)$ does not have a positive simple cycle.

In particular, if C is a zero cycle, all the inequalities above hold. Thus, $w_\lambda(v_k, T_i) = w_\lambda(v_{k-1}, T_i) + w_\lambda(v_{k-1}, v_j)$ for all $k \in \mathbb{N}^+, j \leq n$. If whenever $w_\lambda(u, T_i) + w_\lambda(u, v) - w_\lambda(v, T_i) = 0$, $t(u, T_i) + t(u, v) - t(v, T_i) \leq 0$, then

$$\begin{aligned} t(v_n, T_i) &\geq t(v_0, T_i) + t(v_0, v_1) + ... + t(v_{n-1}, v_n) \\ &= t(v_n, T_i) + t(C), \end{aligned} \tag{5.20}$$

which gives $t(C) \leq 0$, a contradiction to the assumption that C has at least one tick, which means that in this case, $\sigma_{T_i}(\lambda)$ has all negative simple cycles. □

Lemma 14. *For all* $i \in \mathbb{N}, i \leq c+1$,

$$\begin{cases} w_{\lambda_i}(u, T_i) + w_{\lambda_i}(u, v) - w_{\lambda_0}(v, T_i) \leq 0 & \text{for } (u, v) \in E_{max} \\ w_{\lambda_i}(u, T_i) + w_{\lambda_i}(u, v) - w_{\lambda_0}(v, T_i) \geq 0 & \text{for } (u, v) \in E_{min} \end{cases}$$

Proof. We will prove this by induction on $i \in \mathbb{N}, i \leq c+1$. For the base case, $i = 0$, and the statement is true by Theorem 6. Assume that when $i = k \in \mathbb{N}, i \leq c$, the statement is true. Let $i = k+1$. Assume that $e_0 = (u_0, v_0)$ such that $\lambda_{i+1} = \max\{K(e, T_i)\} = K(e_0, T_i)$. By definition, if $v \notin T_i^s(v_0)$, where $T_i^s(v_0)$ denotes the subtree of T_i rooted at v_0, $p(v, T_{i+1}) = p(v, T_i)$, and thus $w_{\lambda_{i+1}}(v, T_{i+1}) = w_{\lambda_{i+1}}(v, T_i)$; if $v \in T_i^s(v_0)$, we have $w(v, T_{i+1}) = w(v, T_i) + \triangle w(e_0, T_i)$ and $t(v, T_{i+1}) = t(v, T_i) + \triangle t(e_0, T_i)$, where $\lambda_{i+1} = \frac{\triangle w(e_0, T_i)}{\triangle t(e_0, T_i)}$.

Thus, $w_{\lambda_{i+1}}(v, T_{i+1}) = w(v, T_{i+1}) - \lambda_{i+1} t(v, T_{i+1}) = w(v, T_i) + \triangle w(e_0, T_i) - \lambda_{i+1}(t(v, T_i) + \triangle t(e_0, T_i)) = w_{\lambda_{i+1}}(v, T_i)$ as well. Thus, the statement is true by the induction hypothesis, which completes the induction. □

Lemma 15. *For all* $i \in \mathbb{N}, i \leq c$, *1.* $\lambda_{i+1} \leq \lambda_i$. *2. If for all* $(u, v) \in E_{max}$, *whenever* $w_{\lambda_i}(u, T_i) + w_{\lambda_i}(u, v) - w_{\lambda_i}(v, T_i) = 0$, $t(u, T_i) + t(u, v) - t(v, T_i) \leq 0$; *for all* $(u, v) \in E_{min}$, *whenever* $w_{\lambda_i}(u, T_i) + w_{\lambda_i}(u, v) - w_{\lambda_i}(v, T_i) = 0$, $t(u, T_i) + t(u, v) - t(v, T_i) \geq 0$, *then the inequality 1 is strict.*

Proof. We first prove Statement 1. If otherwise, assume that $\lambda_{i+1} > \lambda_i$ for some $i = k \in \mathbb{N}$. By the assumption,

$$\lambda_{i+1} = \max_{(u,v) \in E} \{K((u, v), T_i)\} = \frac{w(u, T_i) + w(u, v) - w(v, T_i)}{t(u, T_i) + t(u, v) - t(v, T_i)}. \tag{5.21}$$

Thus, there exists u_0, v_0 and an edge (u_0, v_0) such that

$$\frac{w(u_0, T_i) + w(u_0, v_0) - w(v_0, T_i)}{t(u_0, T_i) + t(u_0, v_0) - t(v_0, T_i)} > \lambda_i. \tag{5.22}$$

If $v_0 \in V_{max}$, by definition, we have $t(u_0, T_i) + t(u_0, v_0) - t(v_0, T_i) > 0$. Thus,

$$w(u_0, T_i) + w(u_0, v_0) - w(v_0, T_i) > \lambda_i(t(u_0, T_i) + t(u_0, v_0) - t(v_0, T_i)), \tag{5.23}$$

a contradiction to Lemma 14. Similarly, if $v_0 \in V_{min}$, by definition, we have $t(u_0, T_i) + t(u_0, v_0) - t(v_0, T_i) = t(u_0, T_i) - t(v_0, T_i) < 0$ and

$$w(u_0, T_i) + w(u_0, v_0) - w(v_0, T_i) < \lambda_i(t(u_0, T_i) + t(u_0, v_0) - t(v_0, T_i)), \tag{5.24}$$

again a contradiction to Lemma 14 as desired.

To prove Statement 2., we prove the contrapositive of the statement: assume that $\lambda_{i+1} = \lambda_i$. Then there exists u_0, v_0 and an edge (u_0, v_0) such that

$$\frac{w(u_0, T_i) + w(u_0, v_0) - w(v_0, T_i)}{t(u_0, T_i) + t(u_0, v_0) - t(v_0, T_i)} = \lambda_i. \tag{5.25}$$

We will either have $v_0 \in V_{max}$, $w_{\lambda_i}(u_0, T_i) + w_{\lambda_i}(u_0, v_0) - w_{\lambda_i}(v_0, T_i) = 0$ and $t(u_0, T_i) + t(u_0, v_0) - t(v_0, T_i) > 0$, or $v_0 \in V_{min}$, $w_{\lambda_i}(u_0, T_i) + w_{\lambda_i}(u_0, v_0) - w_{\lambda_i}(v_0, T_i) = 0$ and $t(u_0, T_i) + t(u_0, v_0) - t(v_0, T_i) < 0$ as desired. □

Lemma 16. *For all* $i \in \mathbb{N}, i \leq c + 1$, $\lambda_{i+1} \leq \lambda \leq \lambda_i$,

$$\begin{cases} w_{\lambda_i}(u, T_i) + w_{\lambda_i}(u, v) - w_{\lambda_i}(v, T_i) \leq 0 & \text{for } (u, v) \in E_{max} \\ w_{\lambda_i}(u, T_i) + w_{\lambda_i}(u, v) - w_{\lambda_i}(v, T_i) \geq 0 & \text{for } (u, v) \in E_{min} \end{cases}$$

Proof. The case when $\lambda_{i+1} = \lambda = \lambda_i$ is trivial by Lemma 14. We assume that $\lambda_{i+1} < \lambda_i$. If otherwise, there exists $i \in \mathbb{N}, i \leq c+1$, $\lambda_{i+1} \leq \lambda' < \lambda_i$ and (u_0, v_0) such that the statement is violated. If $v_0 \in V_{max}$, $w_{\lambda'}(u_0, T_i) + w_{\lambda'}(u_0, v_0) - w_{\lambda'}(v_0, T_i) > 0$, and by Lemma 14, $w_{\lambda_i}(u_0, T_i) + w_{\lambda_i}(u_0, v_0) - w_{\lambda_i}(v_0, T_i) \leq 0$. Thus,

$$(\lambda_i - \lambda')(t(u_0, T_i) + t(u_0, v_0)) > (\lambda_i - \lambda')(t(v_0, T_i)), \tag{5.26}$$

which gives

$$t(u_0, T_i) + t(u_0, v_0) > t(v_0, T_i). \tag{5.27}$$

Thus,

$$\begin{aligned}\lambda' &< \frac{w(u_0, T_i) + w(u_0, v_0) - w(v_0, T_i)}{t(u_0, T_i) + t(u_0, v_0) - t(v_0, T_i)} \\ &\leq \max_{e \in E}\{K(e, T_i)\} \\ &= \lambda_{i+1},\end{aligned} \tag{5.28}$$

a contradiction. If $v_0 \in V_{min}$, $w_{\lambda'}(u_0, T_i) + w_{\lambda'}(u_0, v_0) - w_{\lambda'}(v_0, T_i) < 0$. By Lemma 14, $w_{\lambda_i}(u_0, T_i) + w_{\lambda_i}(u_0, v_0) - w_{\lambda_i}(v_0, T_i) \geq 0$. Thus,

$$(\lambda_i - \lambda')(t(u_0, T_i) + t(u_0, v_0)) < (\lambda_i - \lambda')(t(v_0, T_i)), \tag{5.29}$$

which gives

$$t(u_0, T_i) + t(u_0, v_0) < t(v_0, T_i). \tag{5.30}$$

Thus,

$$\begin{aligned}\lambda' &< \frac{w(u_0, T_i) + w(u_0, v_0) - w(v_0, T_i)}{t(u_0, T_i) + t(u_0, v_0) - t(v_0, T_i)} \\ &\leq \max_{e \in E}\{K(e, T_i)\} \\ &= \lambda_{i+1},\end{aligned} \tag{5.31}$$

a contradiction again, as desired. □

The following two Corollaries directly follow from Lemma 13 and Lemma 16:

Corollary 15. *For all $i \in \mathbb{N}, i \leq c+1$ $\sigma_{T_i}(\lambda)$ does not have positive cycles for all $\lambda_{i+1} \leq \lambda \leq \lambda_i$, which further implies that $\lambda_{c+1} \geq \lambda^\star(G)$.*

Lemma 17. *T_{c+1} is never a spanning tree of G.*

Proof. We prove the statement by contradiction. If otherwise, by definition, $T_{c+1} = T_i$ for some $i \in \mathbb{N}, i \leq c$, which implies that $\lambda_{i+1} = \lambda_{c+2} = \max\{K(e, T_i)\}$ directly

by definition. Since $T_{c+1} = T_i$ is a tree, Lemma 15 can be readily extended to the case when $i \in \mathbb{N}, i \leq c+1$ and we have $\lambda_{i+1} \leq \lambda_{i+2} \leq ... \leq \lambda_{c+2}$, which further implies that $\lambda_{i+1} = \lambda_{i+2} = ... = \lambda_{c+2} = \lambda$. Assume that we get $T_{i+1}, ..., T_{c+1}$ by deleting edge $(u'_i, v_i) \in T_i$ and appending $e_i = (u_i, v_i)$, ..., deleting edge $(u'_c, v_c) \in T_c$ and appending (u_c, v_c). For all $v_a \in V_{max}, i \leq a \leq c$, pick the v_a that $t(v_a, T_a)$ is minimized. When there are multiple such vertices, pick one of them that $D(v_a, T_a)$ is also minimized. For all $v_b \in V_{min}, i \leq b \leq c$, pick the v_b that $t(v_b, T_{b+1})$ is minimized. When there are multiple such vertices, pick one of them that $D(v_b, T_{b+1})$ is also minimized. Finally, define

$$v_j = \arg \min_{v_j \in \{v_a, v_b\}} \{\langle t(v_a, T_a), D(v_a, T_a)\rangle, \langle t(v_b, T_{b+1}), D(v_b, T_{b+1})\rangle\}, \tag{5.32}$$

where comparison of $\langle x, y\rangle$ follows lexicographic order, which first compare x, and in case of equality, compare y.

If $v_j \in V_{max}$, we prove that

$$\langle t(v_j, T_i), D(v_j, T_i)\rangle = \langle t(v_j, T_{c+1}), D(v_j, T_{c+1})\rangle > \langle t(v_j, T_j), D(v_j, T_j)\rangle. \tag{5.33}$$

If otherwise, $\langle t(v_j, T_{c+1}), D(v_j, T_{c+1})\rangle \leq \langle t(v_j, T_j), D(v_j, T_j)\rangle$. Since we know that $t(v_j, T_{j+1}) = t(v_j, T_j) + \triangle t(e_j, T_j) > t(v_j, T_j)$, we can find the smallest $j+1 < k \leq c+1$ such that

$$\langle t(v_j, T_k), D(v_j, T_k)\rangle \leq \langle t(v_j, T_j), D(v_j, T_j)\rangle < \langle t(v_j, T_{k-1}), D(v_j, T_{k-1})\rangle. \tag{5.34}$$

Then $v_{k-1} \in V_{min}$ and $v_j \in {T_{k-1}}^s(v_{k-1}) = {T_k}^s(v_{k-1})$, which gives

$$\langle t(v_{k-1}, T_k), D(v_{k-1}, T_k)\rangle < \langle t(v_j, T_k), D(v_j, T_k)\rangle \leq \langle t(v_j, T_j), D(v_j, T_j)\rangle. \tag{5.35}$$

Thus, $v_{k-1} \neq v_j$ should have been picked as v_j by definition, a contradiction.

Then, based on Inequality 5.33, we can again find the smallest $i < y \leq j$ such that

$$\langle t(v_j, T_y), D(v_j, T_y)\rangle \leq \langle t(v_j, T_j), D(v_j, T_j)\rangle < \langle t(v_j, T_{y-1}), D(v_j, T_{y-1})\rangle. \tag{5.36}$$

Then, $v_{y-1} \in V_{min}$ and $v_j \in {T_{y-1}}^s(v_{y-1}) = {T_y}^s(v_{y-1})$, which gives

$$\langle t(v_{y-1}, T_y), D(v_{y-1}, T_y)\rangle < \langle t(v_j, T_y), D(v_j, T_y)\rangle \leq \langle t(v_j, T_j), D(v_j, T_j)\rangle. \quad (5.37)$$

Thus, $v_{y-1} \neq v_j$ should have been picked as v_j by definition, again a contradiction.

If $v_j \in V_{min}$, we prove that

$$\begin{aligned} \langle t(v_j, T_i), D(v_j, T_i)\rangle &= \langle t(v_j, T_{c+1}), D(v_j, T_{c+1})\rangle \\ &> \langle t(v_j, T_{j+1}), D(v_j, T_{j+1})\rangle. \end{aligned} \quad (5.38)$$

If otherwise, $\langle t(v_j, T_i), D(v_j, T_i)\rangle \leq \langle t(v_j, T_{j+1}), D(v_j, T_{j+1})\rangle$. Since we know that $t(v_j, T_{j+1}) = t(v_j, T_j) + \triangle t(e_j, T_j) < t(v_j, T_j)$, we can find the smallest $i < k \leq j$ such that

$$\langle t(v_j, T_k), D(v_j, T_k)\rangle > \langle t(v_j, T_{j+1}), D(v_j, T_{j+1})\rangle \geq \langle t(v_j, T_{k-1}), D(v_j, T_{k-1})\rangle. \quad (5.39)$$

Then, $v_{k-1} \in V_{max}$ and $v_j \in {T_{k-1}}^s(v_{k-1}) = {T_k}^s(v_{k-1})$, which gives

$$\begin{aligned} \langle t(v_{k-1}, T_{k-1}), D(v_{k-1}, T_{k-1})\rangle &< \langle t(v_j, T_{k-1}), D(v_j, T_{k-1})\rangle \\ &\leq \langle t(v_j, T_{j+1}), D(v_j, T_{j+1})\rangle. \end{aligned} \quad (5.40)$$

Thus, $v_{k-1} \neq v_j$ should have been picked as v_j by definition, a contradiction.

Then, based on Inequality 5.38, we can again find the smallest $j+1 < y \leq c+1$ such that

$$\langle t(v_j, T_y), D(v_j, T_y)\rangle > \langle t(v_j, T_{j+1}), D(v_j, T_{j+1})\rangle \geq \langle t(v_j, T_{y-1}), D(v_j, T_{y-1})\rangle. \quad (5.41)$$

Then, $v_{y-1} \in V_{max}$ and $v_j \in {T_{y-1}}^s(v_{y-1}) = {T_y}^s(v_{y-1})$, which gives

$$\begin{aligned} \langle t(v_{y-1}, T_{y-1}), D(v_{y-1}, T_{y-1})\rangle &< \langle t(v_j, T_{y-1}), D(v_j, T_{y-1})\rangle \\ &\leq \langle t(v_j, T_{j+1}), D(v_j, T_{j+1})\rangle. \end{aligned} \quad (5.42)$$

Thus, $v_{y-1} \neq v_j$ should have been picked as v_j by definition, again a contradiction.

□

Remark 8. *The Lemma implies that there is no need to check the repetition of trees explored in the λ Progression, because every tree explored is different.*

Remark 9. *One can show with similar arguments as above that the case when the algorithm returns $-\infty$ is trivial – it corresponds to the case when at least one strategy in G is acyclic, because $\lambda_{c+1} = -\infty$ and $\sigma_{T_c}(\lambda)$ does not have positive cycles for all $\lambda \leq \lambda_c$, and thus does not have a cycle. For the discussions in this chapter, we will assume that the algorithm never returns $-\infty$.*

5.2.4 Cycle Break

Definition 26. *T_i is called a Boundary Tree (denoted as B-Tree) in $\sigma_{T_i}(\lambda)$ if it is a spanning tree of $\sigma_{T_i}(\lambda)$ that satisfies the following properties:*

1. For some $(u', v') \in E_{max}$, $w_\lambda(u', T_i) + w_\lambda(u', v') - w_\lambda(v', T_i) \geq 0$. In particular, if $w_\lambda(u', T_i) + w_\lambda(u', v') - w_\lambda(v', T_i) = 0$, $t(u', T_i) + t(u', v') - t(v', T_i) > 0$.

2. For all $(u, v) \in E_{min}$, $w_\lambda(u, T_i) + w_\lambda(u, v) - w_\lambda(v, T_i) \geq 0$. In particular, if $w_\lambda(u, T_i) + w_\lambda(u, v) - w_\lambda(v, T_i) = 0$, $t(u', T_i) + t(u', v') - t(v', T_i) \geq 0$.

Corollary 16. *T_c is a B-Tree in $\sigma_{T_c}(\lambda_{c+1})$.*

Proof. By the assumption, T_c is a tree but T_{c+1} is not a tree. Thus, assume that T_{c+1} is got by deleting $(u_0, v') \in T_c$ and appending the edge (u', v'). We must have $u' \in T_c^s(v')$, and $C = p + (u', v')$ is a cycle, where p is the path from v' to u' in T_c,

which implies that

$$0 < t(C) = t(p) + t(u', v') = t(u', T_c) - t(v', T_c) + t(u', v'). \tag{5.43}$$

Also, by definition,

$$\lambda_{c+1} = \frac{w(u', T_c) + w(u', v') - w(v', T_c)}{t(u', T_c) + t(u', v') - t(v', T_c)} > -\infty. \tag{5.44}$$

Thus, $v \in V_{max}$ and $w_{\lambda_{c+1}}(u', T_c) + w_{\lambda_{c+1}}(u', v') - w_{\lambda_{c+1}}(v', T_c) = 0$ and T_c satisfies the Property 1. of B-Tree definition. Lemma 16 indicates that for all $(u, v) \in E_{min}$, $w_{\lambda_{c+1}}(u, T_c) + w_{\lambda_{c+1}}(u, v) - w_{\lambda_{c+1}}(v, T_c) \geq 0$. In particular, if $w_{\lambda_{c+1}}(u, T_c) + w_{\lambda_{c+1}}(u, v) - w_{\lambda_{c+1}}(v, T_c) = 0$ and $t(u, T_c) + t(u, v) - t(v, T_c) < 0$,

$$\frac{w(u', T_c) + w(u', v') - w(v', T_c)}{t(u', T_c) + t(u', v') - t(v', T_c)} = \lambda_{c+1} = \frac{w(u, T_c) + w(u, v) - w(v, T_c)}{t(u, T_c) + t(u, v) - t(v, T_c)}, \tag{5.45}$$

and by the assumption, $(u, v) \in E_{min}$ has higher priority than (u', v'). Then, the λ progression should have picked (u, v) instead of (u', v') for the edge appending, a contradiction. Thus, if $w_{\lambda_{c+1}}(u, T_c) + w_{\lambda_{c+1}}(u, v) - w_{\lambda_{c+1}}(v, T_c) = 0$, $t(u, T_c) + t(u, v) - t(v, T_c) \geq 0$, and T_c also satisfies the Property 2. of B-Tree definition as desired. □

5.2.5 Cycle Break Steps

Assume that we are given a B-Tree T'_0 in $\sigma_{T'_0}(\lambda)$. We will first consider the case when $u' \in T'^s_0(v')$. In this case, $p(u', T'_0) = p(v', T'_0) + p'$, $t(u', T'_0) + t(u', v') - t(v', T'_0) > 0$, and we define "cycle break steps".

Corollary 17. *There exists $C \subset \sigma_{T'_0}$ such that $w_\lambda(C) \geq 0$.*

Proof. By the assumption, $p(u', T'_0) = p(v', T'_0) + p'$ for some $u' \in V, v' \in V_{max}, (u', v') \in$

E_{max}. Consider the cycle $C = (u', v') + p' \subset \sigma_{T_0'}$, we have

$$\begin{aligned} 0 &\leq w_\lambda(u', T_0') + w_\lambda(u', v') - w_\lambda(v', T_0') \\ &= w_\lambda(v', T_0') + w_\lambda(p') + w_\lambda(u', v') - w_\lambda(v', T_0') \\ &= w_\lambda(C) \end{aligned} \tag{5.46}$$

as desired. □

Corollary 18. *$\sigma_{T_0'}(\lambda)$ is not a well-defined strategy.*

Denote such a non-negative cycle $C = (u', v') + p'$ as above. A recursive Cycle Break Step on C is to define T_{k+1}' based on T_k' ($k \in \mathbb{N}$) as following:

1. If there is no edge $(u, v) \in E_{min}$ such that $v \in {T_k'}^s(v'), u \notin {T_k'}^s(v')$, let $T_{k+1}' = T_k'$ and return the current T_k', λ, and a false (Cycle Break Steps fail).

2. If there is an edge $(u, v) \in E_{min}$ such that $v \in {T_k'}^s(v'), u \notin {T_k'}^s(v')$, find an arbitrary (u, v) such that $\triangle w_k = w_\lambda(u, T_k') + w_\lambda(u, v) - w_\lambda(v, T_k')$ is minimized. We denote the (u, v) we pick as (u_k, v_k) and assume that T_{k+1}' is got by deleting the edge (u, v_k) in T_k' and appending the edge (u_k, v_k), and denote that $\triangle t_k = t(u, T_k') + t(u, v) - t(v, T_k')$.

3. If $v_k \notin C$, repeat another Cycle Break Step (the arguments above) on T_{k+1}'; if $v_k \in C$, stop and we will perform other steps as instructed later.

Corollary 19. *For each T_{k+1}' defined given T_k', T_{k+1}' is a tree and*

$$|\{v : v \in {T_{k+1}'}^s(v'), v \in V_{min}\}| \leq |\{v : v \in {T_k'}^s(v'), v \in V_{min}\}| - 1. \tag{5.47}$$

Corollary 20. *The total number of cycle break steps on C is no more than $|V|+1$.*

We will assume that the cycle break steps does not return a false in the following

discussions except the termination theorem or otherwise specified. Assume that we defined $T'_1, T'_2, ..., T'_{n+1}$ $(n \leq |V|)$.

Corollary 21. $\triangle w_0 \geq 0$. *In particular, if* $\triangle w_0 = 0$, $\triangle t_0 \geq 0$.

Corollary 22. *For all* $k \in \mathbb{N}, k \leq n$, *if* $v \in {T'_{k+1}}^s(v')$, *then* $v \in {T'_k}^s(v')$ *and* $v \notin {T'_k}^s(v_k)$; *if* $u \notin {T'_{k+1}}^s(v')$ *and* $u \notin {T'_k}^s(v_k)$, *then* $u \notin {T'_k}^s(v')$.

Corollary 23. *For all* $j, k \leq n+1, j, k \in \mathbb{N}, j \neq k$, ${T_k}'^s(v_j) \cap {T_k}'^s(v_k) = \emptyset$.

Corollary 24. *For all* $k \in \mathbb{N}, k \leq n$, *if* $v \in {T'_k}^s(v_k)$, $w_\lambda(v, T'_{k+1}) = w_\lambda(v, T'_k) + \triangle w_k$, $t(v, T'_{k+1}) = t(v, T'_k) + \triangle t_k$; *if* $v \notin {T'_k}^s(v_k)$, $p(v, T'_{k+1}) = p(v, T'_k)$ *and thus* $w_\lambda(v, T'_{k+1}) = w_\lambda(v, T'_k)$ *and* $t(v, T'_{k+1}) = t(v, T'_k)$.

Corollary 25. *For all* $k \in \mathbb{N}, k \leq n+1$, *if*

1. $v \in {T'_k}^s(v')$, $(u, v) \in E_{min}$ *or*

2. $(u, v) = (u', v')$,

then $w_\lambda(u, T'_k) + w_\lambda(u, v) - w_\lambda(v, T'_k) \geq 0$. *In particular, if* $w_\lambda(u, T'_k) + w_\lambda(u, v) - w_\lambda(v, T'_k) = 0$, $t(u, T'_k) + t(u, v) - t(v, T'_k) \geq 0$

Proof. We prove the statements by induction on $k \in \mathbb{N}$. For the base case, $k = 0$, the statements directly follow by the assumption that T'_0 is a B-Tree. When $k = m \leq n$, and assume that the statements is true. Let $k = m + 1 \leq n + 1$. By the induction hypothesis, for all $v \in {T'_m}^s(v'), (u, v) \in E_{min}$, $w_\lambda(u, T'_m) + w_\lambda(u, v) - w_\lambda(v, T'_m) \geq 0$. Thus, $\triangle w_m \geq 0$ by definition, and thus

$$w_\lambda(u, T'_m) \leq w_\lambda(u, T'_{m+1}) \leq w_\lambda(u, T'_m) + \triangle w_m \tag{5.48}$$

and

$$w_\lambda(v, T'_m) \leq w_\lambda(v, T'_{m+1}) \leq w_\lambda(v, T'_m) + \triangle w_m \tag{5.49}$$

for all $v \in V$.

If $v \in {T'_m}^s(v')$, by Corollary 22, $v \notin {T'_m}^s(v_m)$, and thus $w_\lambda(v, T'_{m+1}) = w_\lambda(v, T'_m)$. We have

$$\begin{aligned} & w_\lambda(u, T'_{m+1}) + w_\lambda(u, v) - w_\lambda(v, T'_{m+1}) \\ & \geq w_\lambda(u, T'_m) + w_\lambda(u, v) - w_\lambda(v, T'_m) \\ & \geq 0 \end{aligned} \tag{5.50}$$

by the induction hypothesis. In particular, if $w_\lambda(u, T'_m) + w_\lambda(u, v) - w_\lambda(v, T'_m) = 0$, $t(u, T'_m) + t(u, v) - t(v, T'_m) \geq 0$ and all the arguments above follow by replacing w_λ by t.

If $(u, v) = (u', v')$, we know that $v' \notin {T'_m}^s(v_m)$. Thus $w_\lambda(v', T'_{m+1}) = w_\lambda(v', T'_m)$ and $t(v', T'_{m+1}) = t(v', T'_m)$

$$\begin{aligned} & w_\lambda(u', T'_{m+1}) + w_\lambda(u', v') - w_\lambda(v', T'_{m+1}) \\ & \geq w_\lambda(u', T'_m) + w_\lambda(u', v') - w_\lambda(v', T'_m) \\ & \geq 0 \end{aligned} \tag{5.51}$$

by the induction hypothesis. In particular, if $w_\lambda(u', T'_m) + w_\lambda(u', v') - w_\lambda(v', T'_m) = 0$, $t(u', T'_m) + t(u', v') - t(v', T'_m) \geq 0$ by the induction hypothesis as well, which completes the induction. □

Corollary 26. *For all $k \in \mathbb{N}, k \leq n-1$, $\triangle w_{k+1} \geq \triangle w_k$.*

Proof. By definition, given $(u_{k+1}, v_{k+1}) \in E_{min}$ such that $v_{k+1} \in {T'_{k+1}}^s(v')$, $u_{k+1} \notin {T'_{k+1}}^s(v')$, by Corollary 22, $v_{k+1} \notin {T'_k}^s(v_k)$ and $v_{k+1} \in {T'_k}^s(v')$. Thus $w_\lambda(v_{k+1}, T'_{k+1}) = w_\lambda(v_{k+1}, T'_k)$.

If $u_{k+1} \in {T'_k}^s(v_k)$, $w_\lambda(u_{k+1}, T'_{k+1}) = w_\lambda(u_{k+1}, T'_k) + \triangle w_k$. Since $v_{k+1} \in {T'_k}^s(v')$, by Corollary 24,

$$\begin{aligned} \triangle w_{k+1} &= w_\lambda(u_{k+1}, T'_{k+1}) + w_\lambda(u_{k+1}, v_{k+1}) - w_\lambda(v_{k+1}, T'_{k+1}) \\ &= w_\lambda(u_{k+1}, T'_k) + \triangle w_k + w_\lambda(u_{k+1}, v_{k+1}) - w_\lambda(v_{k+1}, T'_k) \\ &= \triangle w_k. \end{aligned} \tag{5.52}$$

If $u_{k+1} \notin {T'_k}^s(v_k)$, again by Corollary 22, $u_{k+1} \notin {T'_k}^s(v')$ and thus $w_\lambda(u_{k+1}, T'_{k+1}) = w_\lambda(u_{k+1}, T'_k)$.

$$\begin{aligned} \triangle w_{k+1} &= w_\lambda(u_{k+1}, T'_{k+1}) + w_\lambda(u_{k+1}, v_{k+1}) - w_\lambda(v_{k+1}, T'_{k+1}) \\ &= w_\lambda(u_{k+1}, T'_k) + w_\lambda(u_{k+1}, v_{k+1}) - w_\lambda(v_{k+1}, T'_k) \\ &\geq w_\lambda(u_k, T'_k) + w_\lambda(u_k, v_{k+1}) - w_\lambda(v_k, T'_k) \\ &= \triangle w_k \end{aligned} \tag{5.53}$$

as well because (u_k, v_k) is the $(u, v) \in E_{min}, v \in {T'_k}^s(v'), u \notin {T'_k}^s(v')$ that minimizes $w_\lambda(u, T'_k) + w_\lambda(u, v) - w_\lambda(v, T'_k)$.

□

For all $k \in \mathbb{N}, k \leq n+1$, given an arbitrary strategy $\sigma' \subset G$ and vertex $x_1 \in {T_k}^s(v')$, define x_{m+1} if $m \in \mathbb{N}^+, m \leq |V|, x_m \in {T'_k}^s(v')$ as following:

if $x_m \in V_{max}$ and $x_m = v'$, then let $x_{m+1} = u' \in {T'_k}^s(v')$;

if $x_m \in V_{max}$ and $x_m \neq v'$, then find x_{m+1} such that (x_{m+1}, x_m) is in T_k ($x_{m+1} \in {T_k}^s(v')$ as well);

if $x_m \in V_{min}$, find x_{m+1} such that (x_{m+1}, x_m) is the unique edge whose target is x_m in σ'.

We would then get a sequence of vertices: $x_1, ..., x_j$ $(2 \leq j \leq |V| + 1)$ and (x_m, x_{m-1}) is in σ' for all $m \in \mathbb{N}^+, m \leq j$.

Lemma 18.

$$w_\lambda(x_{m+1}, T'_k) + w_\lambda(x_{m+1}, x_m) \geq w_\lambda(x_m, T'_k) \tag{5.54}$$

for all $m \leq j, m \in \mathbb{N}^+$.

Proof. If $x_m \in V_{max}$, $w_\lambda(x_{m+1}, T'_k) + w_\lambda(x_{m+1}, x_m) = w_\lambda(x_m, T'_k)$, except when $x_m = v'$, in which case we let $x_{m+1} = u'$, $w_\lambda(u', T'_k) + w_\lambda(u', v') - w_\lambda(v', T'_k) \geq 0$ by Corollary 25. If $x_m \in V_{min}$, since $x_m \in {T'_k}^s(v')$, Corollary 25 again implies that $w_\lambda(x_{m+1}, T'_k) + w_\lambda(x_{m+1}, x_m) \geq w_\lambda(x_m, T'_k)$ as desired. □

Lemma 19. *If* $\sigma'(\lambda)$ *is a well-defined strategy, then* $j \leq |V|$.

Proof. If otherwise, $j = |V| + 1$. Since ${T'_k}^s(v')$ has at most $|V|$ vertices, $x_a = x_b$ for some $0 \leq a < b \leq |V| + 1, a, b \in \mathbb{N}$. By Lemma 18, we have

$$\begin{aligned} w_\lambda(x_a, T'_k) &\leq w_\lambda(x_{a+1}, T'_k) + w_\lambda(x_{a+1}, x_a) \leq ... \\ &\leq w_\lambda(x_b, T'_k) + w_\lambda(x_b, x_{b-1}) + ... + w_\lambda(x_{a+1}, x_a) \\ &= w_\lambda(x_b, T'_k) + w(C) \end{aligned} \tag{5.55}$$

where $C = (x_b, x_{b-1}) + ... + (x_{a+1}, x_a)$ is a cycle in σ'. Thus, $w_\lambda(C) \geq 0$, a contradiction to the assumption that $\sigma'(\lambda)$ is a well defined strategy. □

Corollary 27. *If* $\sigma'(\lambda)$ *is a well-defined strategy,* $x_j \notin {T'_k}^s(v'), x_j \in V_{min}$ *and* $x_m \in {T'_k}^s(v')$ *for all* $m \in \mathbb{N}^+, m \leq j - 1$.

Theorem 7 (Termination Criterion)**.** *If a false is returned,* $\lambda \leq \lambda^\star(G)$.

Proof. Given an arbitrary strategy σ' and a vertex $x_1 \in {T_n}^s(v')$, consider the sequence of vertices $x_1, ...x_j$ constructed as described earlier. We know that $x_j \in$

${T'_n}^s(v')$, because if otherwise, $x_j \notin {T'_n}^s(v')$, $x_{j-1} \in {T'_n}^s(v')$, $(x_j, x_{j-1}) \in E_{min}$ by definition, a contradiction to the assumption that a false is returned, and there is no edge $(u, v) \in E_{min}$ such that $v \in {T'_n}^s(v'), u \notin {T'_n}^s(v')$. The contrapositive of Corollary 27 implies that $\sigma'(\lambda)$ is not a well defined strategy. By Corollary 10 $\lambda \leq \hat{\lambda}(\sigma')$, and since σ' is arbitrary,

$$\lambda \leq \min_l\{\hat{\lambda}(\sigma_l)\} = \lambda^\star(G) \tag{5.56}$$

as desired. □

We assume that T'_{n+2} is got by deleting edge (u, v') in T'_{n+1} and appending (u', v'). Assume that $\triangle w_{n+1} = w_\lambda(u', T'_{n+1}) + w_\lambda(u', v') - w_\lambda(v', T'_{n+1})$ and $\triangle t_{n+1} = t(u', T'_{n+1}) + t(u', v') - t(v', T'_{n+1})$.

Corollary 28. *$u' \notin {T'_{n+1}}^s(v')$, and T'_{n+2} is a tree.*

Corollary 29. *If $v \in {T'_{n+1}}^s(v')$, $w_\lambda(v, T'_{n+2}) = w_\lambda(v, T'_{n+1}) + \triangle w_{n+1}$, $t(v, T'_{n+2}) = t(v, T'_{n+1}) + \triangle t_{n+1}$; if $v \notin {T'_{n+1}}^s(v')$, $w_\lambda(v, T'_{n+2}) = w_\lambda(v, T'_{n+1})$ and $t(v, T'_{n+2}) = t(v, T'_{n+1})$.*

Corollary 30. *$\triangle w_{n+1} \geq \triangle w_n$.*

Proof. By convention, we know $u' \in {T'_n}^s(v_n)$ and $v' \notin {T'_n}^s(v_n)$. Then, by Corollary 24 and Corollary 25, $\triangle w_{n+1} = w_\lambda(u', T'_{n+1}) + w_\lambda(u', v') - w_\lambda(v', T'_{n+1}) = w_\lambda(u', T'_n) + \triangle w_n + w_\lambda(u', v') - w_\lambda(v', T'_n) \geq \triangle w_n$ as desired. □

Corollary 31. *If T'_k is a tree, $u \in {T'_k}^s(v_1), u \notin {T'_k}^s(v_2)$, then $v_1 \notin {T'_k}^s(v_2)$.*

Proof. Since $u \in {T'_k}^s(v_1)$, there is a path p_1 from v_1 to u in T'_k. If $v_1 \in {T'_k}^s(v_2)$, there is a path p_2 from v_2 to v_1 in T'_k, which means that there is a path $p_2 + p_1$ from v_2 in T'_k, and then $u \in {T'_k}^s(v_2)$, a contradiction. □

The following Corollary directly follows from Corollary 21, 23, 24, 26 and 30

Corollary 32. *For all $v \in V$ and $k \in \mathbb{N}, k \leq n+2$,*

$$0 \leq w_\lambda(v, T'_k) - w_\lambda(v, T'_0) \leq \max_{0 \leq m \leq k-1} \{\triangle w_m\} = \triangle w_{k-1}. \tag{5.57}$$

Corollary 33. *If $w_\lambda(v, T'_{n+2}) - w_\lambda(v, T'_0) = 0$ for all $v \in V$, then $t(v, T'_{n+2}) - t(v, T'_0) \geq 0$ for all $v \in V$ and in particular $t(v', T'_{n+2}) - t(v', T'_0) > 0$*

Proof. If $w_\lambda(v, T'_{n+2}) - w_\lambda(v, T'_0) = 0$ for all $v \in V$, there is no cost increase in any of the cycle break step and $\triangle w_k = 0 = w_\lambda(u_k, T'_k) + w_\lambda(u_k, v_k) - w_\lambda(v_k, T'_k)$ for all $k \in \mathbb{N}, k \leq n+1$. Since $v_k \in {T'_k}^s(v')$ and $(u_k, v_k) \in V_{min}$ for all $k \in \mathbb{N}, k \leq n$, by Corollary 25, $t(u_k, T'_k) + t(u_k, v_k) - t(v_k, T'_k) = \triangle t_k \geq 0$ for all $k \in \mathbb{N}, k \leq n$, which further implies that $t(v, T'_{n+1}) \geq t(v, T'_0)$ for all $v \in V$. Furthermore, since $v' \in {T'_{n+1}}^s(v')$, Corollary 24 implies that $v' \in {T'_k}^s(v')$ and $v' \notin {T'_k}^s(v_k)$ for all $k \in \mathbb{N}, k \leq n$. Thus, $t(v', T'_{n+1}) = t(v', T'_0)$. Then,

$$\begin{aligned} \triangle t_{n+1} &= t(u', T'_{n+1}) + t(u', v') - t(v', T'_{n+1}) \\ &\geq t(u', T'_0) + t(u', v') - t(v', T'_0) \\ &> 0 \end{aligned} \tag{5.58}$$

by the assumption. Thus,

$$t(v, T'_0) \leq t(v, T'_{n+1}) \leq t(v, T'_{n+2}) \leq t(v, T'_{n+1}) + \triangle t_{n+1} \tag{5.59}$$

and in particular,

$$\begin{aligned} t(v', T'_{n+2}) &= t(v', T'_{n+1}) + \triangle t_{n+1} \\ &> t(v', T'_{n+1}) \\ &= t(v', T'_0) \end{aligned} \tag{5.60}$$

as desired. □

Lemma 20. *After performing the Cycle Break Steps of a B-Tree T'_0, the final tree T'_{n+2} will satisfy that $w_\lambda(v, T'_{n+2}) \geq w_\lambda(v, T'_0)$ for all $v \in V$. In particular, if $w_\lambda(v, T'_{n+2}) = w_\lambda(v, T'_0)$ for all $v \in V$, $t(v, T'_{n+2}) \geq t(v, T'_0)$ and there exists $v' \in V_{max}$ such that $t(v', T'_{n+2}) > t(v', T'_0)$.*

Proof. We have already shown that $\triangle w_k \geq 0$ for all $k \in \mathbb{N}, k \leq n+1$. Thus, $w_\lambda(v, T'_{n+2}) \geq w_\lambda(v, T'_0)$ for all $v \in V$. In particular $w_\lambda(v, T'_{n+2}) = w_\lambda(v, T'_0)$ for all $v \in V$, Corollary 33 implies that $t(v, T'_{n+2}) \geq t(v, T'_0)$ and there exists v' such that $t(v, T'_{n+2}) > t(v, T'_0)$ as desired. □

Corollary 34. *For all $k \in \mathbb{N}, k \leq n+2$, if $(u, v) \in E_{min}, u \in {T'_k}^s(v)$, $w_\lambda(u, T'_k) + w_\lambda(u, v) - w_\lambda(v, T'_k) \geq 0$.*

Proof. We prove this by induction on k. For the base case, $k = 0$ and the statement is true by the assumption. Let $k = m \leq n+1$, and assume that the statement is true. When $k = m+1 \leq n+2$, for all $(u, v) \in E_{min}, u \in {T'_{m+1}}^s(v)$, if $u \notin {T'_{m+1}}^s(v_m)$ (and thus $u \notin {T'_m}^s(v_m)$), by Corollary 31, then $v \notin {T'_{m+1}}^s(v_m)$ (and thus $v \notin {T'_m}^s(v_m)$). By Corollary 24, $p(u, T'_{m+1}) = p(u, T'_m)$ and $p(v, T'_{m+1}) = p(v, T'_m)$ and

$$\begin{aligned} & w_\lambda(u, T'_{m+1}) + w_\lambda(u, v) - w_\lambda(v, T'_{m+1}) \\ &= w_\lambda(u, T'_m) + w_\lambda(u, v) - w_\lambda(v, T'_m) \\ &\geq 0 \end{aligned} \tag{5.61}$$

by the induction hypothesis; if $u \in {T'_{m+1}}^s(v_m)$ (and thus $u \notin {T'_m}^s(v_m)$),

$$\begin{aligned} & w_\lambda(u, T'_{m+1}) + w_\lambda(u, v) - w_\lambda(v, T'_{m+1}) \\ &= w_\lambda(u, T'_m) + \triangle w_m + w_\lambda(u, v) - w_\lambda(v, T'_{m+1}). \end{aligned} \tag{5.62}$$

By Corollary 32, $w_\lambda(u, T'_m)+\triangle w_m \geq w_\lambda(u, T'_0)+\triangle w_m$ and $w_\lambda(v, T'_{m+1}) \leq w_\lambda(v, T'_0)+\triangle w_m$, we have

$$\begin{aligned} & w_\lambda(u, T'_{m+1}) + w_\lambda(u, v) - w_\lambda(v, T'_{m+1}) \\ & \geq w_\lambda(u, T'_0) + \triangle w_m + w_\lambda(u, v) - w_\lambda(v, T'_0) - \triangle w_m \\ & \geq 0 \end{aligned} \tag{5.63}$$

by the assumption that T'_0 is a B-Tree, which completes the induction. □

Corollary 35. *For all* $(u, v) \in E_{min}$ *or* $(u, v) \in T'_{n+2}$, $w_\lambda(u, T'_0) + w_\lambda(u, v) - w_\lambda(v, T'_0) \geq 0$.

Proof. Since T'_0 is a B-Tree, for all $(u, v) \in E_{min}$, $w_\lambda(u, T'_0)+w_\lambda(u, v)-w_\lambda(v, T'_0) \geq 0$ by definition. If $(u, v) \in E_{max}$ and $(u, v) \in T'_{n+2}$, by definition, $(u, v) \in T'_0$ and $w_\lambda(u, T'_0)+w_\lambda(u, v)-w_\lambda(v, T'_0) = 0$, except when $(u, v) = (u', v')$, and by definition, $w_\lambda(u, T'_0) + w_\lambda(u, v) - w_\lambda(v, T'_0) \geq 0$ as well. □

5.2.6 Min-Edge Optimization Steps

Given a spanning tree T'_1 of $\sigma_{T'_1}$, such that for all $(u, v) \in E_{min}, u \in T'^s_1(v)$, $w_\lambda(u, T'_1) + w_\lambda(u, v) - w_\lambda(v, T'_1) \geq 0$, and assume that there exists a spanning tree T'_0 of some $\sigma_{T'_0}$ such that for all $(u, v) \in E_{min}$ or $(u, v) \in T'_1$, $w_\lambda(u, T'_0) + w_\lambda(u, v) - w_\lambda(v, T'_0) \geq 0$, A recursive Min-Edge Optimization Step is to define T'_{k+1} based on T'_k ($k \in \mathbb{N}^+$) as following:

1. If there exists $(u, v) \in E_{min}$ such that $\triangle w_k = w_\lambda(u, T'_k) + w_\lambda(u, v) - w_\lambda(v, T'_k) \leq 0$, and in particular, if $w_\lambda(u, T'_k) + w_\lambda(u, v) - w_\lambda(v, T'_k) = 0$, $\triangle t_k = t(u, T'_k)+t(u, v)-t(v, T'_k) < 0$, find an arbitrary such edge that minimizes $D(u, T'_k)$

and denote it as (u_k, v_k). Define the T'_{k+1} by deleting (u, v_k) in T'_k and appending (u_k, v_k), and repeat another Min-Edge Optimization Step (the arguments above) on T'_{k+1}.

2. If otherwise, return the tree T'_k;

Corollary 36. *For all $k \in \mathbb{N}^+$, if T'_k is defined, it is a spanning tree of some $\sigma_{T'_k} \subset G$ and for all $(u, v) \in E_{min}, u \in {T'_k}^s(v)$, $w_\lambda(u, T'_k) + w_\lambda(u, v) - w_\lambda(v, T'_k) \geq 0$.*

Proof. We prove this by induction on k. For the base case, $k = 1$, and the statement is true by the assumption. Let $k = m \in \mathbb{N}^+$, and assume that the statement is true. When $k = m + 1 \in \mathbb{N}^+$, if T'_{m+1} is defined, by definition, T'_{m+1} is got by deleting (u, v_m) in T'_m and appending $(u_m, v_m) \in E_{min}$, where $w_\lambda(u_m, T'_m) + w_\lambda(u_m, v_m) - w_\lambda(v_m, T'_m) \leq 0$ and in particular, if $w_\lambda(u_m, T'_m) + w_\lambda(u_m, v_m) - w_\lambda(v_m, T'_m) = 0, t(u_m, T'_m) + t(u_m, v_m) - t(v_m, T'_m) \leq 0$. If $w_\lambda(u_m, T'_m) + w_\lambda(u_m, v_m) - w_\lambda(v_m, T'_m) < 0$, the contrapositive of the induction hypothesis implies that $u_m \notin {T'_m}^s(v_m)$; if $w_\lambda(u_m, T'_m) + w_\lambda(u_m, v_m) - w_\lambda(v_m, T'_m) = 0$ and $t(u_m, T'_m) + t(u_m, v_m) - t(v_m, T'_m) \leq 0$, then $u_m \notin {T'_m}^s(v_m)$ as well, because otherwise, assume the simple cycle $C = p + (u_m, v_m)$ where $p = (v_m, ..., u_m) \in T'_m$, we have $t(u_m, T'_m) + t(u_m, v_m) - t(v_m, T'_m) = t(C) > 0$, a contradiction. That $u_m \notin {T'_m}^s(v_m)$ further implies that T'_{m+1} is a spanning tree of some $\sigma_{T'_{m+1}} \subset G$.

Assume that $(u, v) \in E_{min}, u \in {T'_{m+1}}^s(v)$, and we can then find a path p from $v = x_0$ to $u = x_j (j \leq |V|, j \in \mathbb{N}^+, x_0 \neq x_j) : p = (x_0, ..., x_j)$ in T'_{m+1}. For all $0 \leq a \leq j - 1$, if $(x_a, x_{a+1}) \in E_{max}$, we have $(x_a, x_{a+1}) \in T'_1$ since only $e \in E_{min}$ is deleted or appended from T'_1 to T'_2 to ... to T'_{m+1} by definition and we have $w_\lambda(x_a, x_{a+1}) \geq w_\lambda(x_{a+1}, T'_0) - w_\lambda(x_a, T'_0)$ by the assumption. If $(x_a, x_{a+1}) \in E_{min}$, we have $w_\lambda(x_a, x_{a+1}) \geq w_\lambda(x_{a+1}, T'_0) - w_\lambda(x_a, T'_0)$ directly by the assumption again.

Thus,

$$\begin{aligned} & w_\lambda(u, T'_{m+1}) - w_\lambda(v, T'_{m+1}) \\ &= w_\lambda(p) \\ &= w_\lambda(x_0, x_1) + ... + w_\lambda(x_{j-1}, x_j) \\ &\geq w_\lambda(x_1, T'_0) - w_\lambda(x_0, T'_0) + ... + w_\lambda(x_j, T'_0) - w_\lambda(x_{j-1}, T'_0) \\ &= w_\lambda(u, T'_0) - w_\lambda(v, T'_0) \\ &\geq -w_\lambda(u, v) \end{aligned} \tag{5.64}$$

by the assumption, which completes the induction. □

Corollary 37. *For all $k \in \mathbb{N}^+$, if T'_{k+1} is defined, if $v \in {T'_k}^s(v_k)$, $w_\lambda(v, T'_{k+1}) = w_\lambda(v, T'_k) + \triangle w_k \leq w_\lambda(v, T'_k)$; in particular, if $w_\lambda(v, T'_{k+1}) = w_\lambda(v, T'_k) + \triangle w_k = w_\lambda(v, T'_k)$, $t(v, T'_{k+1}) = t(v, T'_k) + \triangle t_k < t(v, T'_k)$. If $v \notin {T'_k}^s(v_k)$, $w_\lambda(v, T'_{k+1}) = w_\lambda(v, T'_k)$ and $t(v, T'_{k+1}) = t(v, T'_k)$.*

Corollary 38. *For all $k \in \mathbb{N}^+$, if T'_k is defined, $u_k \notin {T'_k}^s(v_k)$, and thus $D(u_k, T'_{k+1}) = D(u_k, T'_k)$ and $D(v_k, T'_{k+1}) = D(u_k, T'_{k+1}) + 1 = D(u_k, T'_k) + 1$.*

Corollary 39. *If T'_{k+1} is defined, then $D(u_{k+1}, T'_{k+1}) \geq D(u_k, T'_k)$.*

Proof. By Corollary 36, T'_{k+1} is a tree, and by Corollary 38, $D(u_k, T'_{k+1}) = D(u_k, T'_k)$. If $u_{k+1} \in {T'_k}^s(v_k)$, we have $D(u_{k+1}, T'_{k+1}) \geq D(v_k, T'_{k+1}) = D(u_k, T'_k) + 1$ as desired.

If $u_{k+1} \notin {T'_k}^s(v_k)$, $w_\lambda(u_{k+1}, T'_{k+1}) = w_\lambda(u_{k+1}, T'_k)$. Since Corollary 37 indicates that $w_\lambda(v_{k+1}, T'_{k+1}) \leq w_\lambda(v_{k+1}, T'_k)$, and in particular, if $w_\lambda(v_{k+1}, T'_{k+1}) = w_\lambda(v_{k+1}, T'_k)$, $t(v_{k+1}, T'_{k+1}) \leq t(v_{k+1}, T'_k)$ we have

$$\begin{aligned} & w_\lambda(u_{k+1}, T'_k) + w_\lambda(u_{k+1}, v_{k+1}) - w_\lambda(v_{k+1}, T'_k) \\ &\leq w_\lambda(u_{k+1}, T'_{k+1}) + w_\lambda(u_{k+1}, v_{k+1}) - w_\lambda(v_{k+1}, T'_{k+1}) \\ &\leq 0, \end{aligned} \tag{5.65}$$

and in particular, if $w_\lambda(u_{k+1}, T'_k) + w_\lambda(u_{k+1}, v_{k+1}) - w_\lambda(v_{k+1}, T'_k) = 0$, $t(u_{k+1}, T'_k) + t(u_{k+1}, v_{k+1}) - t(v_{k+1}, T'_k) < 0$. By the assumption, (u_k, v_k) is picked among all such edges (u, v) that $D(u, T'_k)$ is minimized, we have $D(u_k, T'_k) \leq D(u_{k+1}, T'_k) = D(u_{k+1}, T'_{k+1})$ as desired. □

The following stronger Corollary follows directly from Corollary 39.

Corollary 40. *If T'_{k+1} is defined, then $D(u_b, T'_b) \geq D(u_a, T'_a)$ for all $a \leq b \leq k+1, a, b \in \mathbb{N}$.*

Lemma 21. *The total number of Min-Edge Optimization Steps is bounded by $n = |V||E_{min}|$.*

Proof. We prove the statement by contradiction. If otherwise, assume that (x_1, y_1), ... $(x_{|E_{min}|}, y_{|E_{min}|})$ are all different edges in E_{min} and let $S_1, ..., S_{|E_{min}|}$ be sets of edges. For all $j \in \mathbb{N}, j \leq n$, we allocate (u_j, v_j) to S_m if $(u_j, v_j) = (x_m, y_m)$ (there exists a unique $m \leq |E_{min}|, m \in \mathbb{N}^+$ that the equality holds). Assume that $S_{h_1}, ..., S_{h_f}$ $(f \leq |E_{min}|)$ are all sets with cardinalities of at least 2.

By the assumption, consider the $n+1$ Min-Edge Optimization Steps, and we have spanning trees $T'_0, ..., T'_n, T'_{n+1}$ of $\sigma_{T'_0}, ..., \sigma_{T'_n}, \sigma_{T'_{n+1}}$, and edges appended are $(u_0, v_0), (u_1, v_1), ..., (u_n, v_n) \in E_{min}$. The total number of elements in $S_{h_1}, ..., S_{h_f}$ will be then at least $|V||E_{min}| + 1 - (|E_{min}| - f) = f + (|V| - 1)|E_{min}| + 1$. Thus, there exists one set with at least $p = |V_{min}| + 1$ elements. Without loss of generality, assume that this set has elements $(u_{z_1}, v_{z_1}), ..., (u_{z_p}, v_{z_p}), ...; z_1 < ... < z_p \leq n, z_1, ..., z_p \in \mathbb{N}$, and we have $(u_{z_1}, v_{z_1}) = ... = (u_{z_p}, v_{z_p}) = (x_m, y_m), m \leq |E_{min}|, m \in \mathbb{N}^+$ by construction.

If $(u_{z_a}, v_{z_a}) = (u_{z_b}, v_{z_b}) = (x_m, y_m)$ $(a < b \leq p, m \leq |E_{min}|, a, b, m \in \mathbb{N}^+)$, by Corollary 38, $x_m \notin {T'_{z_a}}^s(v_{z_a}), x_m \notin {T'_{z_b}}^s(v_{z_b})$. If $x_m \notin {T'_{z_o}}^s(v_{z_o})$ for all $z_a \leq z_o \leq z_b$,

Corollary 37 indicates that $w_\lambda(x_m, T'_{z_a}) = w_\lambda(x_m, T'_{z_{a+1}}) = ... = w_\lambda(x_m, T'_{z_b})$. Then Corollary 37 also implies that

$$\begin{aligned} \triangle w_{z_b} &= w_\lambda(x_m, T'_{z_b}) + w_\lambda(x_m, y_m) - w_\lambda(y_m, T'_{z_b}) \\ &= w_\lambda(x_m, T'_{z_{a+1}}) + w_\lambda(x_m, y_m) - w_\lambda(y_m, T'_{z_b}) \\ &\geq w_\lambda(x_m, T'_{z_{a+1}}) + w_\lambda(x_m, y_m) - w_\lambda(y_m, T'_{z_{a+1}}) \\ &= 0 \end{aligned} \tag{5.66}$$

and in particular, if $\triangle w_{z_b} = 0$, we will get the similar arguments by replacing w_λ by t, which gives $\triangle t_{z_b} > 0$, a contradiction to the assumption that $\triangle w_{z_b} \leq 0$, and whenever $\triangle w_{z_b} = 0$, $\triangle t_{z_b} < 0$. Thus, the argument implies that $x_m \in {T'_{z_o}}^s(v_{z_o})$ for some $z_a < z_o < z_b$ (and thus $z_b \geq z_a + 2$). Together with Corollary 40 and Corollary 38, we have $D(x_m, T'_{z_b}) \geq D(x_m, T'_{z_o+1}) \geq D(v_{z_o}, T'_{z_o+1})D(u_{z_o}, T'_{z_o}) + 1 \geq D(u_{z_a}, T'_{z_a}) + 1 = D(x_m, T'_{z_a}) + 1$. Similar arguments will give: $D(x_m, T'_{z_p}) \geq D(x_m, T'_{z_{p-1}}) + 1 \geq ... \geq D(x_m, T'_{z_1}) + p - 1 = D(x_m, T'_{z_1}) + |V|$, which is a contradiction to the fact that tree T'_{z_1} and tree T'_{z_p} satisfy: $0 \leq D(x_m, T'_{z_1}) \leq |V| - 1$ and $0 \leq D(x_m, T'_{z_p}) \leq |V| - 1$ as desired. □

Remark 10. *In our algorithm, given a B-Tree T'_0, we first perform Cycle Break Steps and get T'_1, which will satisfy the condition we stated at the beginning of Min-Edge Optimization Steps. Then, we perform Min-Edge Optimization Steps on T'_1 and get T'_n where $n \leq |V||E_{min}| + 1$. All of the Corollaries and Lemmas above hold. Furthermore, we will prove the following Lemma:*

Lemma 22. *For all $k \in \mathbb{N}^+, k \leq n$, $w_\lambda(v, T'_k) \geq w_\lambda(v, T'_0)$ for all $v \in V$. In particular, if $w_\lambda(v, T'_k) = w_\lambda(v, T'_0)$ for all $v \in V$, then $t(v, T'_k) \geq t(v, T'_0)$ for all $v \in V$ and there exists $v' \in V$ such that $t(v', T'_k) > t(v', T'_0)$.*

Proof. We prove this by induction on k. For the base case, $k = 1$, the statement is true by Lemma 20. Let $k = m \in \mathbb{N}^+, k \leq n - 1$, and assume that the statement is

true. When $k = m+1, k \in \mathbb{N}^+$, assume that we get T'_{m+1} by deleting $(u_m, v'_m) \in T'_m$ and appending $(u'_m, v'_m) \in E_{min}$. We denote $\triangle w_m = w_\lambda(u'_m, T'_m) + w_\lambda(u'_m, v'_m) - w_\lambda(v'_m, T'_m)$ and $\triangle t_m = t(u'_m, T'_m) + t(u'_m, v'_m) - w_\lambda(v'_m, T'_m)$. By the induction hypothesis, $w_\lambda(v, T'_m) \geq w_\lambda(v, T'_0)$ for all $v \in V$.

We first prove that $w_\lambda(v, T'_{m+1}) \geq w_\lambda(v, T'_0)$ for all $v \in V$. If otherwise, there exists $v \in V$ such that $w_\lambda(v, T'_{m+1}) < w_\lambda(v, T'_0)$. It further implies that $\triangle w_m < 0$ and $v \in {T'_m}^s(v_m)$. Find such a v that is closest to v_m, denote it and its predecessor in T'_{m+1} as v_0 and u_0, respectively. We have $w_\lambda(v_0, T'_0) > w_\lambda(v_0, T'_{m+1}) = w_\lambda(u_m, T'_{m+1}) + w_\lambda(u_0, v_0) \geq w_\lambda(u_0, T'_0) + w_\lambda(u_0, v_0)$. If $(u_0, v_0) \in E_{max}$, since Min-Edge Optimization Steps do not break any edge in E_{max}, $(u_0, v_0) \in T'_0$ and $w_\lambda(v_0, T'_0) = w_\lambda(u_0, T'_0) + w_\lambda(u_0, v_0)$, a contradiction. If $(u_0, v_0) \in E_{min}$, since T'_0 is a B-Tree, $w_\lambda(u_0, T'_0) + w_\lambda(u_0, v_0) \geq w_\lambda(v_0, T'_0)$, a contradiction as well.

We then prove that if $w_\lambda(v, T'_{m+1}) = w_\lambda(v, T'_0)$ for all $v \in V$, $t(v, T'_n) \geq t(v, T'_0)$ for all $v \in V$ by replacing w_λ and with t in the arguments above.

Finally, we prove that if $w_\lambda(v, T'_{m+1}) = w_\lambda(v, T'_0)$, there exists $v' \in V$ such that $t(v', T'_n) > t(v', T'_0)$. If otherwise, $w_\lambda(v, T'_{m+1}) = w_\lambda(v, T'_0)$ and $t(v, T'_{m+1}) = t(v, T'_0)$ for all $v \in V$, which further implies that $\triangle w_m = 0$ and $\triangle t_m = 0$. By the induction hypothesis, $t(v', T'_m) > t(v', T'_0)$, but $t(v', T'_{m+1}) = t(v', T'_0)$, which implies that $\triangle t_m < 0$, a contradiction as desired. □

Corollary 41. *$\Sigma_v w_\lambda(v, T'_n) \geq \Sigma_v w_\lambda(v, T'_0)$ and if $\Sigma_v w_\lambda(v, T'_n) = \Sigma_v w_\lambda(v, T'_0)$, $\Sigma_v t(v, T'_n) > \Sigma_v t(v, T'_0)$.*

Proof. By Lemma 22, $w_\lambda(v, T'_n) \geq w_\lambda(v, T'_0)$ for all $v \in V$. Thus, $\Sigma_v w_\lambda(v, T'_n) \geq \Sigma_v w_\lambda(v, T'_0)$. In particular, if $\Sigma_v w_\lambda(v, T'_n) = \Sigma_v w_\lambda(v, T'_0)$, $w_\lambda(v, T'_n) = w_\lambda(v, T'_0)$ for all $v \in V$ and $t(v, T'_n) \geq t(v, T'_0)$ for all $v \in V$, and there exists $v' \in V$ such that

$t(v', T'_n) > t(v', T'_0)$. Thus, $\Sigma_v t(v, T'_n) > \Sigma_v t(v, T'_0)$ as desired. □

The following Corollary directly follows from the definition:

Corollary 42. *T'_n satisfies Property 2. of B-Tree definition.*

5.2.7 B-Tree Update Steps

Assume that we are given a tree T'_0 that satisfies the Property 2. of the B-Tree definition. If there is an edge $(u, v) \in E_{max}$ such that $\triangle w_k = w_\lambda(u, T'_k) + w_\lambda(u, v) - w_\lambda(v, T'_k) > 0$, or $\triangle w_k = w_\lambda(u, T'_k) + w_\lambda(u, v) - w_\lambda(v, T'_k) = 0$ and $\triangle t_k = t_\lambda(u, T'_k) + t_\lambda(u, v) - t_\lambda(v, T'_k) > 0$, then T'_0 also satisfies the Property 1. of the B-Tree definition, and is thus a B-Tree. If T'_k is a B-Tree in $\sigma_{T'_k}(\lambda)$, a recursive B-Tree update step is to define T'_{k+1} given T'_k as following: pick one such edge (u, v); if there are multiple such edges find the one that minimize $D(u, T'_k)$. Denote the picked (u, v) as (u_k, v_k).

1. If $u_k \notin T'^s_k(v_k)$, define T''_k by deleting (u'_k, v_k) in T'_k and appending (u_k, v_k). Define T'_{k+1} to be the final tree after performing Min-Edge Optimization Steps on T''_k, and repeat the arguments above on T'_{k+1}.

2. If $u_k \in T'^s_k(v_k)$, return to Cycle Break steps with T'_k; if there is no such edge (u_k, v_k), T'_k is not a B-Tree, and return to λ Progression with T'_k.

Lemma 23. *For all $(u, v) \in E_{min}$, $u \in T''^s_0(v)$, $w_\lambda(u, T''_0) + w_\lambda(u, v) - w_\lambda(v, T''_0) \geq 0$.*

Proof. By the assumption, T'_0 satisfies Property 2. of B-Tree definition, and thus for all $(u, v) \in E_{min}$, $w_\lambda(u, T'_0) + w_\lambda(u, v) - w_\lambda(v, T'_0) \geq 0$. If $u \in T''^s_0(v)$, there are

three possibilities: 1. $u, v \in T_0'^s(v_0)$. 2. $u, v \notin T_0'^s(v_0)$. 3. $u \in T_0'^s(v_0), v \notin T_0'^s(v_0)$. (Notice that if $v \in T_0'^s(v_0)$, then $u \in T_0''^s(v) \subseteq T_0'^s(v) \subseteq T_0'^s(v_0)$, so it is not possible that $u \notin T_0'^s(v_0), v \in T_0'^s(v_0)$). The first case indicates that

$$\begin{aligned} & w_\lambda(u, T_0'') + w_\lambda(u, v) - w_\lambda(v, T_0'') \\ & = w_\lambda(u, T_0') + \triangle w_0 + w_\lambda(u, v) - w_\lambda(v, T_0') - \triangle w_0 \\ & = w_\lambda(u, T_0') + w_\lambda(u, v) - w_\lambda(v, T_0') \\ & \geq 0 \end{aligned} \tag{5.67}$$

as desired. The second case indicates that

$$\begin{aligned} & w_\lambda(u, T_0'') + w_\lambda(u, v) - w_\lambda(v, T_0'') \\ & = w_\lambda(u, T_0') + w_\lambda(u, v) - w_\lambda(v, T_0') \\ & \geq 0 \end{aligned} \tag{5.68}$$

as desired. The third case indicates that

$$\begin{aligned} & w_\lambda(u, T_0'') + w_\lambda(u, v) - w_\lambda(v, T_0'') \\ & = w_\lambda(u, T_0') + \triangle w_0 + w_\lambda(u, v) - w_\lambda(v, T_0') \\ & \geq w_0 \\ & \geq 0 \end{aligned} \tag{5.69}$$

as desired. □

Lemma 24. *For all* $(u, v) \in E_{min}$ *or* $(u, v) \in T_0''$, $w_\lambda(u, T_0') + w_\lambda(u, v) + w_\lambda(v, T_0') \geq 0$

Proof. Since T_0' satisfies Property 2. of B-Tree definition, for all $(u, v) \in E_{min}$, $w_\lambda(u, T_0') + w_\lambda(u, v) - w_\lambda(v, T_0') \geq 0$. Also, T_0'' is got by deleting edge $(u_0', v_0) \in T_0'$ and appending edge (u_0, v_0), every edge in T_0'' is also in T_0', except the edge (u_0, v_0).

Thus, for all $(u,v) \neq (u_0,v_0)$, $w_\lambda(u,T_0') + w_\lambda(u,v) + w_\lambda(v,T_0') = 0$. Finally, if $(u,v) = (u_0,v_0)$, we have $\triangle w_k = w_\lambda(u,T_k') + w_\lambda(u,v) - w_\lambda(v,T_k') \geq 0$ directly by definition of B-Tree Update Steps as desired. □

Corollary 43. *For all $k \in \mathbb{N}$ such that T_k' and T_k'' are defined, 1. For all $(u,v) \in E_{min}$, $u \in T_k''^s(v)$, $w_\lambda(u,T_k'') + w_\lambda(u,v) - w_\lambda(v,T_k'') \geq 0$. 2. For all $(u,v) \in E_{min}$ or $(u,v) \in T_k''$, $w_\lambda(u,T_k') + w_\lambda(u,v) + w_\lambda(v,T_k') \geq 0$*

Proof. Lemma 23 and Lemma 24 guarantee that T_0' and T_0'' satisfy the conditions required for performing Min-Edge Optimization Steps on T_0'' (by replacing T_0' and T_1' in Min-Edge Optimization Steps section with T_0' and T_0'' in this section) and all the Lemmas and Corollaries hold in that section. The final output T_1' will then satisfy Property 2. of the B-Tree definition by Corollary 42, and the statements follow by the same arguments on T_k' for all $k \in \mathbb{N}^+$. □

Remark 11. *In our algorithm, we are given a B-Tree T_{i_0} in $\sigma_{T_{i_0}}(\lambda)$ to perform Cycle Break Steps, followed by Min-Edge Optimization Steps to get T_{i_1}. If T_{i_1} is still a B-Tree, perform either one B-Tree Update Step followed by another Min-Edge Optimization Steps, or Cycle Break Steps followed by another Min-Edge Optimization Steps, depending on whether the picked edge in $(u,v) \in E_{max}$ satisfies that $u \in {T_{i_1}}^s(v)$ until we get some tree T_{i_n} that is no longer a B-Tree in $\sigma_{T_{i_n}}(\lambda)$, which we continue the λ Progression on (see the Appendix).*

Corollary 44. $n < 2^{|E|}$.

Proof. If otherwise, we will have spanning trees of G: $T_{i_0}, ..., T_{i_{2^{|E|}}}, ...$ and Corollary 41 implies that for all $k \in \mathbb{N}, k \leq 2^{|E|} - 1$, $\Sigma_v w_\lambda(v,T_{i_{k+1}}') \geq \Sigma_v w_\lambda(v,T_{i_k}')$. In particular, if $\Sigma_v w_\lambda(v,T_{i_{k+1}}') = \Sigma_v w_\lambda(v,T_{i_k}')$, $\Sigma_v t(v,T_{i_{k+1}}') > \Sigma_v t(v,T_{i_k}')$. Thus, We know that $T_{i_0}, ..., T_{i_{2^{|E|}}}$ are all distinct. However, there are at most $2^{|E|}$ different

subgraphs containing all the vertices in V, and thus at most $2^{|E|}$ spanning trees of G, a contradiction. □

Corollary 45.

$$\lambda_{i_{n+1}} = \max\{K(e, T_{i_n})\} < \lambda \tag{5.70}$$

Proof. We know that T_{i_n} satisfies the Property 2. of B-Tree definition. Since it is not a B-Tree in $\sigma_{T_{i_n}}(\lambda)$, it does not satisfy the Property 1 of B-Tree definition. We prove the statement by contradiction. If otherwise, $\lambda_{i_{n+1}} = \max\{K(e, T_{i_n}\} = K(e_0, T_{i_n}) \geq \lambda$. By definition, if $e_0 = (u, v) \in E_{min}$, then $t(u, T_{i_n}) + t(u, v) - t(v, T_{i_n}) < 0$ and $w_\lambda(u, T_{i_n}) + w_\lambda(u, v) - w_\lambda(v, T_{i_n}) \leq 0$, a contradiction to the Property 2. of B-Tree definition; if $e_0 = (u, v) \in E_{max}$, then $t(u, T_{i_n}) + t(u, v) - t(v, T_{i_n}) > 0$ and $w_\lambda(u, T_{i_n}) + w_\lambda(u, v) - w_\lambda(v, T_{i_n}) \geq 0$, in which case T_{i_n} satisfies the Property 1 of B-Tree definition, a contradiction as well. □

5.2.8 Complete Algorithm

Putting all of the pieces together, we describe the final complete algorithm: we start from T_0 corresponding to $\sigma_{T_0}(\lambda_0)$, which is a spanning tree of G and satisfies both max vertex inequality and min vertex inequality, perform λ Progression on T_0, and get T_1 (and λ_1). If T_1 is a tree, continue λ Progression on T_1; if not, perform Cycle Break Steps on T_0 (including Min-Edge Optimization Steps and B-Tree Update Steps). If Cycle Break Steps are not performed successfully (return a false), terminate and return the current tree T_0 and the current λ_1 as $\lambda^\star(G)$. If Cycle Break Steps are performed successfully, assign T_1 as the output tree of Cycle Break Steps (λ_1 does not change). T_1 will be a spanning tree of G that satisfies Property 2. of the B-Tree definition, but not Property 1. of the B-

Tree definition, which means that it satisfies both max vertex inequality and min vertex inequality. In this case, we continue λ Progression on T_1, and repeat the steps above in this case. (Lemma 44 guarantees that Cycle Break Steps including Min-Edge Optimization Steps and B-Tree Update Steps terminates, either returns a true on a success, or a false on a failure in finite steps). The complete algorithm is documented in the Appendix. Assume that we have $T_0, ..., T_n, ...$ corresponding to $\sigma_{T_0}(\lambda_0), ..., \sigma_{T_n}(\lambda_n), ...$ and $\lambda_0, ..., \lambda_{n+1}$.

Lemma 25. $n < 2^{|E|}$.

Proof. We prove the Lemma by contradiction. If otherwise, and we will have $T_0, ..., T_{2^{|E|}}, ...$ corresponding to $\sigma_{T_0}(\lambda_0), \sigma_{T_1}(\lambda_1), ..., \sigma_{T_{2^{|E|}}}(\lambda_{2^{|E|}}), ...$ Assume that among trees $T_0, ..., T_{2^{|E|}}$, $T_{i_1}, ..., T_{i_k}$ are trees got by performing Cycle Break Steps on $T_{i_1-1}, ..., T_{i_k-1}$ ($k \in \mathbb{N}^+, i_1 < ... < i_k \leq 2^{|E|}, i_1, ..., i_k \in \mathbb{N}^+$), and let $i_0 = 0, i_{k+1} = 2^{|E|} + 1$.

By definition of λ Progression, given any $j \in \mathbb{N}, j \leq k$, $T_{i_j}, ...T_{i_{j+1}-1}$ are distinct spanning trees of G. We will prove that $T_0, ..., T_{2^{|E|}}$ are all different. If otherwise, there exists some $a < b \leq k, a, b \in \mathbb{N}, i_a \leq x \leq i_{a+1} - 1, i_b \leq y \leq i_{b+1} - 1$ such that $T_x = T_y$. Lemma 15 implies that given any $j \in \mathbb{N}, j \leq k$, $\lambda_p \leq \lambda_q$ for all $i_j \leq q \leq p \leq i_{j+1}, p, q \in \mathbb{N}$. Furthermore, Corollary 45 implies that $\lambda_{i_j+1} < \lambda_{i_j}$, for all $j \in \mathbb{N}, j \leq k$. Altogether, we have

$$\lambda_{y+1} \leq \lambda_{i_b+1} < \lambda_{i_b} \leq ... \leq \lambda_{i_{a+1}} \leq \lambda_{x+1}, \tag{5.71}$$

and Lemma11 then gives that $T_x \neq T_y$, a contradiction. □

Theorem 8. *Our algorithm terminates and stops at the final tree T_f. Furthermore, the final value $\lambda_{f+1} = \lambda^\star(G)$ is returned.*

Remark 12. *Lemma 25 (together with Lemma 44) provides a trivial upper bound and states that our algorithm terminates. When the algorithm terminates at the final tree T_f, (and T_{f+1} is not a tree by definition), by Corollary 15, $\lambda_{f+1} \geq \lambda^\star(G)$. Furthermore, by definition, the Cycle Break Steps on T_f failed, and by Theorem 7, $\lambda_{f+1} \leq \lambda^\star(G)$. Thus, $\lambda_{f+1} = \lambda^\star(G)$.*

5.2.9 Runtime Evaluation in Practice

In this section, we show the runtime of our algorithm on random graphs of different scales. Our input graph assumes that every edge has 1 tick, with random edges added between vertices and $U(0, 20)$ random weight assigned to each edge. Our algorithm has a preprocessing phase, which converts the input graph to the min-max systems that satisfy (i) $E_{min} \subseteq V_{max} \times V_{min}$ and (ii) $t(e) = 0$. As a side note, we utilize the acyclic nature of G' and implement an $\mathcal{O}(|E|)$ version of the acyclic min-max algorithm as mentioned in Remark 7 (instead of the $\mathcal{O}(|V||E|)$ version documented in the proof section and Appendix) with the help of topological sort of G', to make our whole algorithm more scalable; also, our algorithm is implemented in C++ and our current implementation uses a "map" (instead of a "priority queue" documented in the Appendix) to store the edge keys. The experiment in this section is conducted on a Linux machine with 28 cores Intel Xeon E5-2660V4 2.0GHz CPU and 251GB memory.

For each random graph g_i, we report number of max nodes $|V_{max}|$, number of min nodes $|V_{min}|$, number of max edges $|E_{max}|$ and number of min edges $|E_{min}|$ in the original graphs; number of max nodes $|V_{max}|_c$, number of min nodes $|V_{min}|_c$, number of max edges $|E_{max}|_c$ and number of max edges $|E_{min}|_c$ in the converted min-max systems; the min-max cycle time $\lambda^\star$ that our proposed min-max cycle

Table 5.1: Scales of input random graphs

$Name$	$\|V_{max}\|$	$\|V_{min}\|$	$\|E_{max}\|$	$\|E_{min}\|$
$g1$	20	20	53	29
$g2$	10	50	81	394
$g3$	100	50	769	304
$g4$	200	200	2127	2035
$g5$	500	500	5518	4942
$g6$	2000	1000	4755	1364
$g7$	100	10000	287	21162
$g8$	100	10000	1104	98955
$g9$	10000	100	108938	1021
$g10$	1000	50000	10868	490093
$g11$	50000	10000	300517	49910
$g12$	10000	50000	59761	250655
$g13$	10000	100000	55503	455677
$g14$	100000	100000	1099218	1000747
$g15$	200000	200000	2700855	2499064
$g16$	1000000	1000000	3582011	2581407
$g17$	1000000	3000000	2222974	3666730

time algorithm returns and its total (sequential) runtime t_s (including that of the preprocessing phase, but not including that of the random graph construction phase, which is only a tiny portion of the total runtime).

As shown in Table 5.1 and Table 5.2, the algorithm runs reasonably fast on most large graphs in practice. The runtime depends heavily on the graph structure. We also noticed that the algorithm spends majority of the time on Cycle Break Steps (including Min-Edge Optimization Steps and B-Tree Update Steps), and very tiny portion on the λ Progression. For graphs with large ratio of number of min nodes to that of max nodes (and large ratio of number of min edges to that of max edges) in the original graph, the min-max cycle time is generally smaller due to large number of possible strategies (remember that the cycle time is to take the minimum of maximum cycle ratios for each strategy). In this case, the corresponding runtime is also generally higher, because the algorithm can potentially spend more time on

Table 5.2: Scales of converted min-max systems and final runtime of the algorithm

$Name$	$\|V_{max}\|_c$	$\|V_{min}\|_c$	$\|E_{max}\|_c$	$\|E_{min}\|_c$	$\lambda^\star$	$t_s(s)$
$g1$	49	20	111	29	12.700	0.002
$g2$	404	50	869	394	11.238	0.011
$g3$	769	304	1377	304	18.617	0.014
$g4$	2235	200	6197	2035	18.237	0.082
$g5$	5442	500	15402	4942	18.266	0.217
$g6$	3364	1000	7483	1364	14.042	0.139
$g7$	21262	10000	42611	21162	4.174	24.891
$g8$	99055	10000	199014	98955	0.855	224.404
$g9$	11021	100	110980	1021	19.164	1.758
$g10$	491093	50000	991054	490093	0.484	5428.394
$g11$	99910	10000	400337	49910	18.126	10.298
$g12$	260655	50000	561071	250655	1.940	2099.301
$g13$	465677	100000	966857	455677	1.500	7440.381
$g14$	1100747	100000	3100712	1000747	19.789	63.998
$g15$	2699064	200000	7698983	2499064	19.441	170.969
$g16$	3581407	1000000	8744825	2581407	14.788	585.886
$g17$	4666730	3000000	9556434	3666730	12.384	491.630

Cycle Break Steps (including Min-Edge Optimization Steps and B-Tree Update Steps) to explore those large number of possible strategies. We conjecture that the current ordering of edge picking in Min-Edge Optimization Steps and B-Tree Update Steps is bad and expensive, and some slight modifications on those parts might further improve the time efficiency of our algorithm significantly, or even lead to breakthrough in theoretical time complexity of the algorithm.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this thesis, we consider the asynchronous static timing analysis problem.[1] We give a graph-based rigorous proof of the exact periodicity property of asynchronous systems under AND-causality and fixed delays assumptions. In particular, we show that regardless of initial configuration, after finite number of each signal transition in the system $k^\star$, which we provide an analytical bound on, the system will enter steady-state, where each signal transition occurs with a fixed a fixed time period $Mp^\star$ and occurrence period M. We establish our result on a weaker assumption about the system connectivity, which is stronger than those of previous work, and provides a theoretical foundation for the exact periodicity property to be applied and exploited in more general asynchronous circuits containing bundled data logic or synchronous components. Based on this enhanced theoretical work, and the concept of performance slack we introduced in the graph-based proof, we present and implement Cyclone, which is, to our best knowledge, the first comprehensive description and implementation of timing and power analysis engine capable of handling large asynchronous circuits. By leveraging existing theory and efficient algorithms, and using Galois framework for parallelization, Cyclone is fast and accurate, as demonstrated by the experimental results.

[1] This chapter is partially based on the "Conclusion" section in our work [21] and the "Conclusions and Potential Future Work" section in our work [20].

Specifically, we address several major issues in asynchronous timing and power analysis: (i) timing graph construction; (ii) slew propagation and delay calculations for annotating timing graph; (iii) fast maximum cycle ratio algorithm implementation; (iv) asynchronous notions of arrival time, required time, and performance slack, and correctness slack of timing forks; (v) the power calculation; and (vi) parallelization using Galois to achieve 6.90× speedup for large benchmarks in full static timing analysis.

We also show that it is possible to interface asynchronous circuits to synchronous logic without metastability, provided all inputs to the asynchronous circuits are clocked, and provide an example of one such interface.

Finally, our Cyclone timing and power engine assumes AND-causality, which gives conservative timing and power analysis for asynchronous circuits. This thesis also considers the problem of cycle time of min-max system, one of the important algorithmic problems that we will encounter when extending Cyclone engine to more general systems that contain OR-causality. We provide a brand new algorithm on finding cycle time of min-max system. We prove that our algorithm terminates and returns the correct result, and show that the algorithm runs reasonably fast in practice. It is still unknown that whether our algorithm or a similar version will terminate in polynomial time in the worst case, but we hope that the thesis can give readers intuitions on new directions to solve the problem.

6.2 Potential Future Work

We hope that our work can contribute to both asynchronous circuit design support and theory of algorithmic complexity. Our research opens doors for lots of interest-

ing future work. In our current Cyclone timing and power engine, the correctness slack uses delay values based on steady-state input slew rate. In order to give more accurate correctness slack report, one needs to consider transient slew rate during the initialization period of asynchronous circuits. Also, the non-linear delay model we adopt in Cyclone engine starts to lose its accuracy as the technology scales down, and we need more accurate models – composite current source model, for example, which has been adopted in industry, to get more accurate timing and power analysis results. Other possible works include hierarchical timing analysis and incremental timing analysis to improve the efficiency of Cyclone; the top-k critical cycles report, the counterpart of top-k critical paths report in synchronous static timing analysis domain to improve the robustness of Cyclone.

Another important topic of robustness enhancement of Cyclone is, as mentioned, to remove the pessimism and extend timing and power analysis to more general systems with both AND-causality and OR-causality. Cycle time of min-max system is only one of the difficulties on extending Cyclone engine to more general systems. Notice that the *XRER* system introduced in [28] can only handle certain forms of OR-causality, and cannot handle common scenarios that arises such as dual-rail encoded data. Thus, another difficulty is to generate the correct timing graph for systems containing both AND-causality and general OR-causality.

Finally, this thesis still does not answer the question that whether the problem of cycle time of min-max systems is solvable in polynomial time. Although the thesis proposed a new algorithm that runs reasonably fast in practice, there is no proved polynomial bound on the worst case time complexity for this algorithm or other similar methods. This problem extends far beyond the asynchronous static timing analysis, and an efficient solution to the problem will lead to both important

applications and profound impacts on theory of computing complexity.

APPENDIX A

MIN-MAX CYCLE TIME ALGORITHM DOCUMENTATIONS

Algorithm 2 Min-Max Cycle Time Algorithm

Min_Max_Cycle_Time(G) ▷ $G = (V, E, w, t)$

```
for v ∈ V do
    c(v) := ⊥, D(v) := ⊥, τ(v) := ⊥
end for
pq.insert(⟨(−∞, 0, ⊥), ⊥⟩)        ▷ Priority queue pq follows the lexicographic order
T := Acyclic(G, pq, c, τ, D, nk, Key)                          ▷ T = (V, E_T)
while true do
    ⟨(λ, i_0, v_0), u_0⟩ := pq.max()                           ▷ i_0 ∈ {0, 1}
    if λ = −∞ then
        return λ                          ▷ There is an acyclic strategy, λ = −∞
    else if u_0 ∈ T^s(v_0) then           ▷ T^s(v_0) is the sub-tree of T rooted at v_0
        C := p(v_0, ..., u_0) + (u_0, v_0) where path p ∈ T
        if Cycle_Break(G, T, C, u_0, v_0, pq, λ, c, τ, D, nk, Key) = false then
            return λ                      ▷ The cycle cannot be broken, λ = λ*(G)
        end if
    else
        Next_Tree(G, T, pq, c, τ, D, nk, Key, u_0, v_0)                ▷ λ Progression
    end if
end while
```

Algorithm 3 Acyclic Min-Max Tree Algorithm

Acyclic$(G, pq, c, \tau, D, nk, Key)$
  $C(s,0) := 0, D(s,0) := 0, E_T := \emptyset$
  $T := (V, E_T)$
  for $v \in V, v \neq s$ **do**
    $C(v,0) := \perp$
    $nk(v,0) := \perp$
    $Key(v) := -\infty$
  end for
  for $k \leftarrow 1$ to $|V|$ **do**
    $u := \perp$
    if $v \in V_{max}$ **then**
      $C(v,k) := \max_{(v_j,v)\in E, t(v_j,v)=0, C(v_j,k-1)\neq\perp}\{C(v_j,k-1) + w(v_j,v)\}$
      $u := v_j$
    else if $v \in V_{min}$ **then**
      $C(v,k) := \min_{(v_j,v)\in E, t(v_j,v)=0, C(v_j,k-1)\neq\perp}\{C(v_j,k-1) + w(v_j,v)\}$
      $u := v_j$
    end if
    if $C(v,k) \neq C(v,k-1)$ **then**
      $nk(v,k) := u$
      $D(v,k) := D(u,k-1) + 1$
    else
      $nk(v,k) := nk(v,k-1)$
      $D(v,k) := D(v,k-1)$
    end if
  end for
  for $v \in V, v \neq s$ **do**
    $E_T := E_T \cup (nk(v,|V|), v)$
    $c(v) := C(v,|V|)$
    $\tau(v) := 0$
    $D(v) := D(v,|V|)$
    $nk(v) := nk(v,|V|)$
    for $(u,v) \in E$ **do**
      $key :=$ Find_Key$(e = (u,v), c, \tau)$
      if $key \geq Key(v)$ **then**
        $Key(v) := key$
        $nk(v) := u$
      end if
    end for
    if $v \in V_{min}$ **then**
      $pq.insert(\langle(Key(v), 1, v), nk(v)\rangle)$
    else
      $pq.insert(\langle(Key(v), 0, v), nk(v)\rangle)$
    end if
  end for
  return T

Algorithm 4 Cycle Break Steps

Cycle_Break$(G, T, C, u_0, v_0, pq, \lambda, c, \tau, D, nk, Key)$

```
while true do
    v' := ⊥
    u' := ⊥
    △cost := ∞
    for v ∈ V_min ∩ T^s(v_0) do
        for (u, v) ∈ E_min, u ∉ T^s(v_0) do
            △c := c(u) + w(u, v) − c(v)
            △τ := τ(u) + t(u, v) − τ(v)
            if △c − λ△t < △cost then
                △cost := △c − λ△t
                u' := u
                v' := v
            end if
        end for
    end for
    if △cost < ∞ then
        Next_Tree(G, T, pq, c, τ, D, nk, Key, u', v')
        if v' ∈ C then
            Next_Tree(G, T, pq, c, τ, D, nk, Key, u_0, v_0)
            Min_Edge_Optimization(G, T, pq, c, τ, D, nk, Key)
            if B_Tree_Update(G, T, C, u_0, v_0, pq, c, τ, D, nk, Key) = true then
                return true
            end if
        end if
    else
        return false                          ▷ Cycle Break Steps fail, λ = λ*(G)
    end if
end while
```

Algorithm 5 Update the Tree Structure and Corresponding Parameters

Next_Tree$(G, T, pq, c, \tau, D, nk, Key, u_0, v_0)$

$e := (u_0', v_0) \in E_T$
$E_T := E_T - e$
$E_T := E_T \cup (u_0, v_0)$
$\triangle c := c(u_0) + w(u_0, v_0) - c(v_0)$
$\triangle \tau := \tau(u_0) + t(u_0, v_0) - \tau(v_0)$
$\triangle D := D(u_0) + 1 - D(v_0)$
for $x \in T^s(v_0)$ **do**
  $c(x) := c(x) + \triangle c$
  $\tau(x) := \tau(x) + \triangle \tau$
  $D(x) := D(x) + \triangle D$
end for
for $x \in T^s(v_0)$ **do**
  $K_{orix} := Key(x)$
  $Key(x) := -\infty$
  for $(u, x) \in E$ **do**
    $key :=$ Find_Key$(e = (u, x), c, \tau)$
    if $key \geq Key(x)$ **then**
      $Key(x) := key$
      $nk(x) := u$
    end if
  end for
  if $x \in V_{min}$ **then**
    $pq.erase((K_{orix}, 1, x))$
    $pq.insert(\langle (Key(x), 1, x), nk(x) \rangle)$
  else
    $pq.erase((K_{orix}, 0, x))$
    $pq.insert(\langle (Key(x), 0, x), nk(x) \rangle)$
  end if
  for $(x, v) \in E$ **do**
    $K_{oriv} := Key(v)$
    $Key(v) := -\infty$
    for $(u, v) \in E$ **do**
      $key :=$ Find_Key$(e = (u, v), c, \tau)$
      if $key \geq Key(v)$ **then**
        $Key(v) := key$
        $nk(v) := u$
      end if
    end for
    if $v \in V_{min}$ **then**
      $pq.erase((K_{oriv}, 1, v))$
      $pq.insert(\langle (Key(v), 1, v), nk(v) \rangle)$
    else
      $pq.erase((K_{oriv}, 0, v))$
      $pq.insert(\langle (Key(v), 0, v), nk(v) \rangle)$
    end if
  end for
end for

Algorithm 6 Min-Edge Optimization Steps

Min_Edge_Optimization($G, T, pq, c, \tau, D, nk, Key$)

```
for k ← 1 to |V| − 1 do
    for u : D(u) = k do
        for (u, v) ∈ E_min do
            △c := c(u) + w(u, v) − c(v)
            △τ := τ(u) + t(u, v) − τ(v)
            △cost := △c − λ△τ
            if △cost < 0 or (△cost = 0 and △τ < 0) then
                Next_Tree(G, T, pq, c, τ, D, nk, Key, u, v)
            end if
        end for
    end for
end for
```

Algorithm 7 B-Tree Update Steps

B_Tree_Update($G, T, C, u_0, v_0, pq, c, \tau, D, nk, Key$)

```
while true do
    i := B_Tree_Update_Inner_Loop(G, T, C, u_0, v_0, pq, c, τ, D, nk, Key)
    if i = 0 then
        return false                    ▷ Cycle detected, go back to cycle break
    else if i = 1 then
        return true                     ▷ Not a B-Tree, go back to Key Update Steps
    end if
end while
```

Algorithm 8 B-Tree Update Steps (Inner Loop)

B_Tree_Update_Inner_Loop$(G, T, C, u_0, v_0, pq, c, \tau, D, nk, Key)$

```
for k ← 1 to |V| − 1 do
    for u : D(u) = k do
        for (u, v) ∈ E_max do
            △c := c(u) + w(u, v) − c(v)
            △τ := τ(u) + t(u, v) − τ(v)
            △cost := △c − λ△τ
            if △cost > 0 or (△cost = 0 and △τ > 0) then
                if u ∈ T^s(v) then
                    C := p(v, ..., u) + (u, v) where path p ∈ T
                    v_0 := v
                    u_0 := u
                    return 0                    ▷ Cycle detected, go back to cycle break
                else
                    Next_Tree(G, T, pq, c, τ, D, nk, Key, u, v)
                    Min_Edge_Optimization(G, T, pq, c, τ, D, nk, Key)
                    return 2                    ▷ Finish 1 B-Tree Update Step, continue
                end if
            end if
        end for
    end for
end for
return 1                                        ▷ Not a B-Tree, go back to Key Update Steps
```

Algorithm 9 Edge Key Evaluation

Find_Key$(e = (u, v), c, \tau)$

```
△τ := τ(u) + t(u, v) − τ(v)
if (v ∈ V_max and △τ > 0) or (v ∈ V_min and △τ < 0) then
    △c := c(u) + w(u, v) − c(v)
    return △c/△t
else
    return −∞
end if
```

BIBLIOGRAPHY

[1] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. Truenorth: Design and tool flow of a 65mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, October 2015.

[2] Filipp Akopyan, C Tadeo, O Otero, and R Manohar. Hybrid synchronous-asynchronous tool flow for emerging vlsi design. In *IEEE International Workshop on Logic Synthesis*, 2016.

[3] Henrik Björklund and Sergei Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 155(2):210–229, 2007.

[4] Lubos Brim, Jakub Chaloupka, Laurent Doyen, Raffaella Gentilini, and Jean-François Raskin. Faster algorithms for mean-payoff games. *Formal methods in system design*, 38(2):97–118, 2011.

[5] Steven M. Burns. *Performance analysis and optimization of asynchronous circuits*. PhD thesis, California Instititute of Technology, 1991.

[6] Cristian S Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasi-polynomial time. *SIAM Journal on Computing*, (0):STOC17–152, 2020.

[7] Bernadette Charron-Bost, Matthias Fugger, and Thomas Nowak. New transience bounds for long walks. *E-print arXiv:1209.3342*, 2012.

[8] Frederic Commoner, Anatol W Holt, S Even, and A Pnueli. Marked directed graphs. *Journal of Computer System and Sciences*, 5:511–523, 1971.

[9] Asa Dan, Rajit Manohar, and Yoram Moses. On using time without clocks via zigzag causality. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 241–250. ACM, 2017.

[10] Ali Dasdan. Experimental analysis of the fastest optimum cycle ratio and mean algorithms. *Transactions on Design Automation of Electronic Systems*, 9(4):385–418, 2004.

[11] Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979.

[12] Stephane Gaubert and Carlos Klimann. Rational computation in dioid algebra and its application to performance evaluation of discrete event systems. *Algebraic computer in control*, 1991. Number 165 in Lecture Notes in Computer Science, Springer.

[13] Gregoire Gimenez, Abdelkarim Cherkaoui, Guillaume Cogniard, and Laurent Fesquet. Static timing analysis of asynchronous bundled-data circuits. *24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2018.

[14] Mark R. Greenstreet. *STARI: A technique for high-bandwidth communication.* PhD thesis, Princeton University, January 1993.

[15] Jeremy Gunawardena. Timing analysis of digital circuits and the theory of min-max functions. *International workshop on timing issues in the specifications and synthesis of digital systems (TAU)*, September 1993.

[16] Jeremy Gunawardena. Cycle times and fixed points of min-max functions. *11th International Conference on Analysis and Optimization of Systems Discrete Event Systems*, 199:266–272, 1994.

[17] Vladimir A Gurvich, Alexander V Karzanov, and LG Khachivan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *USSR Computational Mathematics and Mathematical Physics*, 28(5):85–91, 1988.

[18] D. Hanson. On the product of the primes. *Canad. Math. Bull. 15*, pages 33–37, 1972.

[19] Mark Hartman and Cristina Arguelles. Transience bounds for long walks. *Mathematics of Operational Research*, 24(2):414–439, 1999.

[20] Wenmian Hua, Yi-Shan Lu, Keshav Pingali, and Rajit Manohar. Cyclone: a static timing and power engine for asynchronous circuits. In *2020 26th IEEE 19th International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 11–19. IEEE, 2020.

[21] Wenmian Hua and Rajit Manohar. Exact timing analysis for asynchronous

systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(1):203–216, 2017.

[22] Henrik Hulgaard, Steven M. Burns, Tod Amon, and Gaetano Borriello. An algorithm for exact bounds on the time separation of events in concurrent systems. Technical Report 94-02-02, University of Washington, Department of Compter Science and Engineering, 1994.

[23] Marcin Jurdziński. Deciding the winner in parity games is in up∩co-up. *Information Processing Letters*, 68(3):119–124, 1998.

[24] Robert Karmazin, Stephen Longfield, Carlos Tadeo Ortega Otero, and Rajit Manohar. Timing driven placement for quasi delay-insensitive circuit. *Proceedings of 21st IEEE International Symposium on Asynchronous Circuits and Systems*, pages 45–52, 2015.

[25] Richard M. Karp and James B. Orlin. Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Applied Mathematics*, 3:37–45, 1981.

[26] Chris LaFrieda, Benjamin Hill, and Rajit Manohar. An asynchronous FPGA with two-phase enable scaled routing. In *IEEE International Symposium on Asynchronous Circuits and Systems*, May 2010.

[27] Yi-Hsiang Lai, Chi-Chuan Chuang, and Jie-Hong R.Jiang. A general framework for efficient performance analysis of acyclic asynchronous pipelines. *Proceedings of International Conference on Computer-Aided Design*, pages 736–743, 2015.

[28] Tak Kwan Lee. *A general approach to performance analysis and optimization of asynchronous circuits*. PhD thesis, Caltech, 1995.

[29] Andrew Lenharth, Donald Nguyen, and Keshav Pingali. Parallel graph analytics. *Commun. ACM*, 59(5):78–87, April 2016.

[30] R Manohar and A Martin. Slack elasticity in concurrent computing. *Lecture Notes in Computer Science*, Jan 1998.

[31] Rajit Manohar. Systems and methods for performing automated conversion of representations of synchronous circuit designs to and from representations of asynchronous circuit designs, 2009. US Patent 7,610,567.

[32] Rajit Manohar and Alain J. Martin. Quasi-delay-insensitive circuits are turing complete. *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1996.

[33] Rajit Manohar and Yoram Moses. Analyzing isochronic forks with potential causality. In *Asynchronous Circuits and Systems (ASYNC), 2015 21st IEEE International Symposium on*, pages 69–76. IEEE, 2015.

[34] Rajit Manohar and Yoram Moses. The eventual c-element theorem for delay-insensitive asynchronous circuits. In *2017 23rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 102–109. IEEE, 2017.

[35] Jotham Vaddaboina Manoranjan and Kenneth S. Stevens. Qualifying relative timing constraints for asynchronous circuits. *22nd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 91–98, 2016.

[36] A Martin, A Lines, R Manohar, M Nystrom, P Penzes, R Southworth, U Cummings, and Tak Kwan Lee;. The design of an asynchronous MIPS R3000 microprocessor. In *Proceedings of the 17th Conference on Advanced Research in VLSI*, pages 164–181, Sep 1997.

[37] Alain J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In *Proceedings of the sixth MIT conference on Advanced research in VLSI*, AUSCRYPT '90, pages 263–278, Cambridge, MA, USA, 1990. MIT Press.

[38] Peggy B. McGee and Steven M. Nowick. An efficient algorithm for time separation of events in concurrent systems. *IEEE/ACM International Conference on Computer-Aided Design*, pages 180–187, 2007.

[39] Glenn Merlet, Thomas Nowak, and Sergei Sergeev. Weak csr expansions and transience bounds in max-plus algebra. *Linear Algebra and its Applications*, 461:163–199, 2014.

[40] Gabriele Miorandi, Marco Balboni, Steven M Nowick, and Davide Bertozzi. Accurate assessment of bundled-data asynchronous nocs enabled by a predictable and efficient hierarchical synthesis flow. In *2017 23rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 10–17. IEEE, 2017.

[41] Matheus T. Moreira, Michel Arendt, Adriel Ziesemer, Ricardo Reis, and Ney Laert Vilar Calazans. Automated synthesis of cell libraries for asynchronous circuits. *27th Symposium on Integrated Circuits and Systems Design*, pages 1–7, 2014.

[42] Hervé Moulin. Extensions of two person zero sum games. *Journal of Mathematical Analysis and Applications*, 55(2):490–508, 1976.

[43] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[44] Mehrdad Najibi and Peter A. Beerel. Deriving performance bounds for conditional asynchronous circuits using linear programing. *18th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 9–16, 2012.

[45] Mehrdad Najibi and Peter A Beerel. Performance bounds of asynchronous circuits with mode-based conditional behavior. In *2012 IEEE 18th International Symposium on Asynchronous Circuits and Systems*, pages 9–16. IEEE, 2012.

[46] A. Neckar, S. Fok, B. V. Benjamin, T. C. Stewart, N. Oza, A. R. Voelker, C. Eliasmith, R. Manohar, and K. Boahen. Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model. *Proceedings of the IEEE*, 107(1):144–164, Jan 2019.

[47] Donald Nguyen, Andrew Lenharth, and Keshav Pingali. A lightweight infrastructure for graph analytics. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP, pages 456–471, 2013.

[48] Keshav Pingali, Donald Nguyen, Milind Kulkarni, Martin Burtscher, M. Amber Hassaan, Rashid Kaleem, Tsung-Hsien Lee, Andrew Lenharth, Roman Manevich, Mario Méndez-Lojo, Dimitrios Prountzos, and Xin Sui. The TAO of parallelism in algorithms. In *Proc. ACM SIGPLAN Conf. Programming Language Design and Implementation*, PLDI '11, pages 12–25, 2011.

[49] Nicolai N Pisaruk. Mean cost cyclical games. *Mathematics of Operations Research*, 24(4):817–828, 1999.

[50] Ernst S. Selmer. On the linear diophantine problem of frobenius. *J.-Reine-Angew.-Math.*, (293/294):1–17, 1977.

[51] Sergei Sergeev and Hans Schneider. CSR expansions of maxtrix powers in max algebra. *Transactions of AMS*, 364:5696–5994, 2012. E-print arXiv:0912.2534.

[52] Montek Singh, Steven M. Nowick, Jose A. Tierno, Sergey Rylov, and Alexander Rylyakov. An adaptively-pipelined mixed synchronous-asynchronous digital fir filter chip operating at 1.3 gigahertz. In *Proceedings of the 8th International Symposium on Asynchronus Circuits and Systems*, ASYNC '02. IEEE Computer Society, 2002.

[53] Neal E. Young, Robert E. Tarjan, and James B. Orlin. Faster parametric shortest path and minimum-balance algorithms. *Networks*, 21(2):205–221, 1991.

[54] Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1-2):343–359, 1996.