# STOCHASTIC CYCLE PERIOD ANALYSIS IN TIMED CIRCUITS

by

Eric G. Mercer

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical Engineering

The University of Utah

Submission date is to soon to rationally cope with!

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

## SUPERVISORY COMMITTEE APPROVAL

of a thesis submitted by

Eric G. Mercer

This thesis has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

Chair: Chris J. Myers

Erik Brunvand

Yong-Bin Kim

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

# FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the thesis of Eric G. Mercer in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

| | |
|---|---|
| Date | Chris J. Myers<br>Chair, Supervisory Committee |

Approved for the Major Department

Om Gandhi
Chair/Dean

Approved for the Graduate Council

David S. Chapman
Dean of The Graduate School

# ABSTRACT

This thesis presents a method of deriving a performance metric for timed asynchronous circuits called a stochastic cycle period, which uses analytical techniques combined with simulation to capture the stochastic profile of the system. The stochastic cycle period is constructed by finding transition and steady-state probabilities in a reachability graph of the timed circuit. The transition and steady-state probabilities are used to obtain trigger probabilities in the circuit implementation. The trigger probabilities are employed in a timing simulation to construct the stochastic cycle period of the timed specification. Since this performance metric is a stochastic profile of the circuit behavior with regards to its individual components, synthesis optimization efforts can be focused on areas that significantly improve the expected cost of a cycle in the system. This thesis presents some case studies where the metric is used to evaluate and improve designs. The studies show the potential of the performance metric and its import in the design process.

To my loving toe, which functioned.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

## ACKNOWLEDGEMENTS

Many thanks to those who contributed to this work:

- Chris J. Myers who always made time for me.
- Shennon Mercer who never left me when I didn't show up on time.
- Office mates who didn't throw me out of the penthouse window when I got a little kooky.

# CHAPTER 1

# INTRODUCTION

Crucial to the correctness of most digital circuits is the notion of global synchronization. Each computational stage in a design must be completed before the fall of the mallet on the drum, signifying the clock pulse, designating the time to start on the next computation. This constant beat coordinates the global efforts of the design to complete its intended task by ensuring that all components are working in step with each other. The idea of a governing global clock orchestrating the movements of data in a system has been the predominately used approach in the design of digital circuits, but due to recent advances in the size and the complexity of these circuits, the realization of this ideology is being challenged.

Aspect ratios and design sizes have evolved at a phenomenal rate in digital circuit implementations. No longer is it feasible to propagate a synchronizing signal from one area of a chip to another in a negligible amount of time. Globally marching to the same beat is not something that can easily be achieved in many designs. Clock skew derived from wire delays has necessitated the revaluation of the traditional global synchronization approach to design.

Asynchronous design styles were engendered before many synchronous techniques, but were left to the wayside because of their perceived difficulty of implementation. Inherent to asynchronous design is the lack of a global synchronizing clock. Components in an asynchronous circuit operate as fast as they can and notify other components when they have completed their work. Since this type of design approach does not rely on a global clock to synchronize activities, it is being reconsidered as a possible solution to synchronous design challenges.

Many techniques exist for designing asynchronous circuits and each has its own merits [8]. However, the work presented in this thesis relies upon, and is an extension to, a specific class of asynchronous circuits called *timed asynchronous circuits* [12, 14, 16]. Important to this research is a general notion of a timed circuit and its position relative to other asynchronous methodologies. Timed circuits uphold a more realistic model of circuit behavior. Rather than restricting inputs and outputs to occur in tightly ordered bursts or to assume a possibly infinite delay, timed circuits portray signals as being able to happen within a finite bounded window of time from when they become enabled, thus accounting for variations in signal propagation and generation delay. These *timing assumptions* can be used to aggressively design a circuit, eliminating extra circuitry that might have been needed had the timing assumptions not been in effect. Timed circuits rely on the fact that a designer knows *a priori* the timing requirements and specifications of the circuit and its environment; and through an iterative process, can refine those assumptions, thus arriving at a completed design. `ATACS` is a tool designed to take a specification of a control system annotated with timing assumptions [26] and output the resulting circuit. This tool uses many detailed algorithms to consider and verify the effects of the timing assumptions on a final implementation [3, 22].

Pivotal to any *computer aided design* (CAD) tool is the ability to analyze the relative merit of different design incarnations at different levels of the design process. `ATACS` provides many methods of verifying and synthesizing timed circuits, but previously did not provide any type of performance metric for evaluating design alternatives. Traditional methods of performance analysis involve the simulation of the design at different levels of the design process. Simulation is often slow, not comprehensive, and only shows an overall performance of the design without indicating the relative performance of specific system components. Without this information, it is difficult to know where to improve an unoptimized circuit or which design alternative to use when confronted with several choices.

This research presents a method of deriving a performance metric for timed asynchronous circuits which does not rely solely on simulation, but uses analytical techniques combined with simulation to capture the stochastic profile of the system. Not only does this metric evaluate the relative overall computational performance of the specification, but it shows the contribution of individual components to the overall performance metric. This type of information aids the designer in filtering out less efficient solutions and helps expose the effects of local synchronization points in highly concurrent systems. Since the performance metric shows a stochastic form of the circuit behavior with regards to its individual components, optimization efforts can be focused in highly probable areas of the circuit, yielding increased computational performance in the average case.

## 1.1 Performance Analysis

Measuring performance in an asynchronous circuit is an elusive task. The delay in a timed asynchronous circuit is a complex function of process variation, data, and operating environment. The multitudinous dependencies in the circuit make it convenient to use a stochastic model. With a stochastic model, transitions are presupposed to occur within a designer specified time window and the actual firing time is a random variable whose tendencies can be described with basic probability theory. Therefore, a good performance metric is one that can be applied at several levels of the design process, shows how much work or progress an implementation is making relative to some performance criteria (speed, area, power, etc.), and brings to light the effects of the stochastic personality of the system.

This thesis merges the cycle period metric from [5] and the Markovian analysis from [2] to define the *stochastic cycle period* performance metric, which can be used for the analysis of timed asynchronous systems with choice and bounded delay. The stochastic cycle period is a measure of the expected value of the delay between two consecutive occurrences of any transition in a cycle of the specification and can be applied to any class of timed circuits where such a transition can be identified. The metric is composed of weighted delays on trigger signals in the actual gate implementation of the specification. This detailed information targets areas in circuits where resources can be invested to gain significant improvements in the average-case or expected performance of the implementation. The stochastic cycle period metric can be directly applied in many different facets such as comparing protocols, finding better state variable insertion points [9], pin reordering, transistor sizing, guiding decomposition, etc. In order to legitimate the metric, this thesis presents case studies showing how the stochastic cycle period can be used to evaluate designs and guide implementation decisions. The case studies demonstrate the ability of the performance metric to correctly identify better implementations of timed circuit specifications.

## 1.2 Related Work

A plethora of research exists studying various methods of performance analysis for many classes of asynchronous circuits. The worked described in this thesis relies largely upon results from the network world [10, 18] and key methods developed in [5, 24]. Burns in [5] legitimates the cycle period and latency functions as good performance metrics for quasi-delay-insensitive and speed-independent circuits. Burns presents a series of definitions and proofs that relate the *minimum timing simulation* to the cycle period or latency of the asynchronous system through a *linear timing function.* He proposes methods and algorithms for finding the cycle period in choice-free *event-rule* systems. Burns proves the cycle period metric to be a convex function and proposes an algorithm for optimizing the function to increase the speed of the circuit subject to a variety of constraints. Burns models delay as being fixed and does not consider delay variation in the circuit. All performance metrics are, therefore, upper bounds on the worst-case performance of the circuit and do not take into account the nature of delay variation and choice behavior indigenous to asynchronous systems when analyzing and optimizing an implementation.

Beerel in [2] presents a method of optimizing the decomposition and technology mapping of burst-mode circuits using information derived from stochastic analysis of the system. By modeling the circuit as a Markov process, a *long-term* or *steady-state* behavior of the system can be discovered through analysis. A signal on a gate that is the latest arriving signal most of the time (also known as a *trigger* or *causal* signal), can be placed near the output of the gate, causing the gate to switch faster. This work establishes stochastic analysis as a viable source of information to guide circuit optimization in burst-mode circuits. Further work in [24, 25] presents implicit and string compression techniques for managing memory requirements and increasing convergence rates when analyzing large Markovian systems. Unique to the work in [2] is the use of discrete methods to find the timed reachable states of the system. Discrete state space exploration methods are proven to produce a number of states that is exponential in the size of the delays and the discretization constant. Implicit methods and string compression are introduced into the Markovian analysis as a means of dealing with the monolithic state space. It is important to note that work presented in this thesis does not use discrete methods of timed state space exploration, but rather uses methods rooted in geometric timing regions and partial orders. These methods produce significantly smaller highly periodic state spaces which are exploited to avoid many size and convergence issues in the Markovian analysis that are commonly encountered with discrete methods.

## 1.3 Contributions

The significant contributions of this thesis include the following: the augmentation of Petri nets to include various bounded distributions and timing information, the definition and derivation of transition probabilities in reachability graphs, the definition and derivation of trigger probabilities in timed circuits, and the stochastic cycle period as a performance metric for circuits with bounded time delays and choice.

In deriving the stochastic cycle period, this thesis presents an exact definition of conditional transition probabilities in timed systems and suggests three methods of approximating its value. The first method is simulation based and best approaches exact transition probabilities. The second method is an exponential approximation of the transition probabilities that draws upon Markovian theory to model the system as a continuous time Markov process. The final method is a burst-mode heuristic which operates on a general class of timed circuits and is shown to be exact for timed circuits that support a basic burst-mode specification requirement. Each method is implemented in `ATACS` and evaluated in the case studies.

The method of analyzing and optimizing timed circuits presented in this thesis is implemented in the tool `ATACS` and several practical examples are analyzed and optimized by the tool, the results of which are presented in this thesis. This thesis shows the relative merits of this type of approach and exposes both its strength and weaknesses for future considerations.

## 1.4 Thesis Overview

The organization of this text is as follows: Chapter 2 presents a formal definition of the timed stochastic Petri net, which is the event model used to represent a timed circuit. Chapter 3 discusses transition probabilities in reachability graphs. It presents a definition for the exact value of the transition probabilities and defines three methods for approximating the exact values: a simulation based method, an exponential approximation, and a burst mode heuristic. Chapter 4 is an overview of methods to find long-term behavior of the timed specification. This includes a simulation based method and a Markovian approach. Chapter 5 combines the methods in Chapters 3 and 4 to create the stochastic cycle period. This is done by first defining trigger probabilities and then presenting a method of finding the stochastic cycle period. At this point several case studies are evaluated in Chapter 6 to try and expose the import of the stochastic cycle period as a performance metric. This chapter is designed to illustrate the strengths and weaknesses of this approach to performance analysis. Chapter 7 concludes the thesis by summarizing the works completed. A brief analysis of the results is presented, as well as suggestions for future work and extensions to this research.

# CHAPTER 2

# EVENT SYSTEM MODEL

All work presented in this thesis is rooted in an event-based paradigm representing an asynchronous specification. This model is systematically derived from hand shaking expansions composed in *VHDL* as shown in [26] and is represented in a form that lends itself to easy manipulation by existing algorithms found in `ATACS`. `ATACS`' internal depiction of the aggregate is built upon an event system first proposed by Winskel in [23], that was incrementally refashioned by [5, 12, 21] to arrive at its contemporary embodiment as a *timed event-rule* (ER) structure and is a representation able to deal with complex highly concurrent timed systems. Unique to the timed ER structure is the representation of timing requirements as bounded delays having a *lower* and *upper* bound which must be satisfied before an event can occur.

Closely related to the timed ER structure, but distinctly different and more widely used, is a Petri net. Since a Petri net is a model that is generally understood and accepted, this thesis describes the maturation of the stochastic cycle period using a Petri net system. This fundamental change of syntactic expression is done in an effort to better convey the core of this research, while preserving the semantic impetus of the underlying representation. For specific differences between the two representation, the reader is referred to Appendix A.

## 2.1 Timed Stochastic Petri Nets

Compulsory to the syntactic transformation of timed ER structure is an augmentation of the basic Petri net model to accept timing and stochastic information. This is done by amending the Petri net models described in [10, 11] to incorporate the semantics of the timed ER structure and allow for bounded stochastic distributions on places, creating the *Timed Stochastic Petri Net* or *TSPN*. Where possible, common Petri net notation will be used. Formally, a *TSPN* system is a modified one-safe Petri net represented with the 6-tuple $\langle P, T, F, M_o, \Delta, \Upsilon \rangle$ where

- *$P$ is the set of places;*
- *$T$ is the set of transitions, $T \cap P = \emptyset$;*
- *$F \subseteq (P \times T) \cup (T \times P)$ is the flow relation;*
- *$M_o \subseteq P$ is the initial marking;*
- *$\Delta : P \rightarrow pdf$ , maps each place to a bounded probability distribution function;*
- *$\Upsilon : P \times T \rightarrow \Re$, maps place transition pairs to real numbers used in resolving conflicts.*

The preset of a transition $t \in T$ is the set of all places in the flow relation that preceed $t$ (i.e. $\bullet t = \{p \in P \mid (p,t) \in F\}$) and the postset of $t$ is the set of all places in the flow relation that follow $t$ (i.e. $t\bullet = \{p \in P \mid (t,p) \in F\}$). Similarly, the preset of a place $p \in P$ is the set of all transitions in the flow relation that preceed $p$ (i.e. $\bullet p = \{t \in T \mid (t,p) \in F\}$) and the postset of $p$ is the set of all transitions in the flow relation that follow $p$ (i.e. $p\bullet = \{t \in T \mid (p,t) \in F\}$). A marking or a state $M$ in a *TSPN* is $M \subseteq P$, where each $p \in M$ is a marked place. With a marking $M$, the *untimed enabled* function is defined to return the set of transitions $T_e$ that have all places in their presets marked in $M$ or $T_e(M) = \{t \in T \mid \forall p \in \bullet t, p \in M\}$.

The $\Delta$ function maps a bounded delay distribution onto places. It is used for reachability analysis, simulation, and performance analysis. Implemented distributions include uniform, truncated normalized Gaussian, and singular (i.e $U(l,u)$, $N(l,u,\mu,\sigma)$, and $S(l)$). Formally, when a token arrives in a place $p$, a random value is sampled from $\Delta(p)$ to obtain a time when the token becomes available to transitions in $p\bullet$. As time advances tokens become available and are placed into the set $P_a$. In this model, an interleaving semantics is enforced and therefore, if it is ever the case where two or more places should become available at the same time, then they are randomly sequenced to be made available one at a time. When a sufficient number of tokens are in $P_a$ to satisfy the $\bullet t$ of some transition, then the transition $t$ instantly fires.

If it is ever the case that the number of available transitions to fire is greater that one (i.e. $|T_e(P_a)| > 1$), then the *TSPN* has conflicting transitions available and arbitration is required before the system can move forward. A *conflict* arises from a choice-point in the Petri net where timing considerations are not able to arbitrate the transition selection and an appeal must made to $\Upsilon$ for knowledge on how to proceed. The responsibility of $\Upsilon$ is to provide a choice distribution for transitions which are enabled by a common place in the *TSPN*. Given a marking $M$, a set of available places $P_a \subseteq M$, and a place $p \in P_a$, where $p$ is the last place to be made available through time and $p$ causes multiple transitions $T_e(P_a)$ to become available, then the probability of any transition $t \in T_e(P_a)$ is

$$Pr\{t \mid P_a, p\} = \begin{cases} 0 & \text{if } t \notin T_e(P_a), \\ \frac{\Upsilon(p,t)}{\sum_{t' \in T_e(P_a)} \Upsilon(p,t')} & \text{otherwise.} \end{cases}$$

which is the conditional probability of $t$ given that $p$ is the last place made available in $P_a$. From this, conflict is resolved by correctly sampling the conditional distribution to select a transition to fire. Note that $\Upsilon$ has no range restrictions due to the normalization process in the probability calculation. This type of choice resolution maintains a semblance of user specified conflict behavior when timing fails to arbitrate the choice. In addition, this model can address known types of choice places which arise in Petri net representations.

Figure 2.1 shows possible choice structures that can be conceived in a *TSPN* model. Unique choice is a non-conflicting choice, meaning that the composition or structure of the net restricts either the left or the ride side to receive a token, but never both. Free, extended free, and controlled choice represent conflicting choice, meaning that two or more transitions can become enabled with a single place becoming available. For the case of free choice, when place 1 becomes available, the probability of $a+$ and $b+$ is

$$\begin{aligned} Pr\{a+ \mid \{1\}, 1\} &= \frac{\Upsilon(1, a+)}{\Upsilon(1, a+) + \Upsilon(1, b+)} \\ &= \frac{0.7}{0.7 + 0.3} \\ &= 0.7 \\ Pr\{b+ \mid \{1\}, 1\} &= 0.3 \end{aligned}$$

For extended free choice, the interleaving semantics restricts two places from becoming enabled at the same instance, so either the left or the right place becomes available last and the structure is reduced to free choice. Controlled choice is resolved with either timing or probability. With timing it is possible to have the situation where place 2 is in the available set $P_a$ and place 1 or place 3 is then added to $P_a$. In this case, $P_a$ is either $\{1, 2\}$ or $\{2, 3\}$ and no choice exists, so only one transition ($a+$ or $b+$) is enabled to fire.

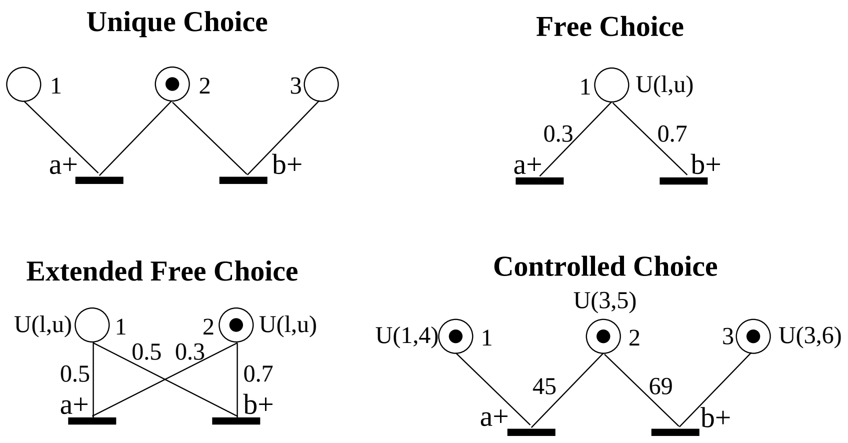

**Figure 2.1**. Supported choice structures in the TSPN model.

If it is ever the case that $P_a$ contains places $\{1, 3\}$ when place 2 becomes available, then the conditional probability of each enabled transition ($a$+ and $b$+ is computed as

$$\begin{aligned} Pr\{a+ \mid \{1,2,3\},2\} &= \frac{\Upsilon(2,a+)}{\Upsilon(2,a+)+\Upsilon(2,b+)} \\ &= \frac{45}{45+69} \\ &= 0.395 \\ Pr\{b+ \mid \{1,2,3\},2\} &= 0.605. \end{aligned}$$

The conflict is resolved by sampling the conditional distribution to correctly choose one of the clashing transitions to fire.

The function $C_p(M, M')$ is defined to identify marking pairs that are connected via a transition enabled by a choice place and returns that place if it exists, otherwise it returns the empty set. This function is formally defined as follows:

$$C_p(M,M') = \{p \in M \mid M \xrightarrow{t} M' \in \Gamma \wedge p \in \bullet t \wedge \exists t' \in T \;.\; p \in \bullet t' \wedge t' \in T_e(M)\}.$$

By the semantic definition of the *TSPN* with regards to allowable choice constructs, $|C_p(M, M')|$ for all $(M, M')$ pairs must be equal to one or zero, meaning that the function will return a single place or the empty set. In the case of extended free choice, the system restricts the two free choice places from every containing a token at the same time, therefore such a marking does not exist in the $RG$.

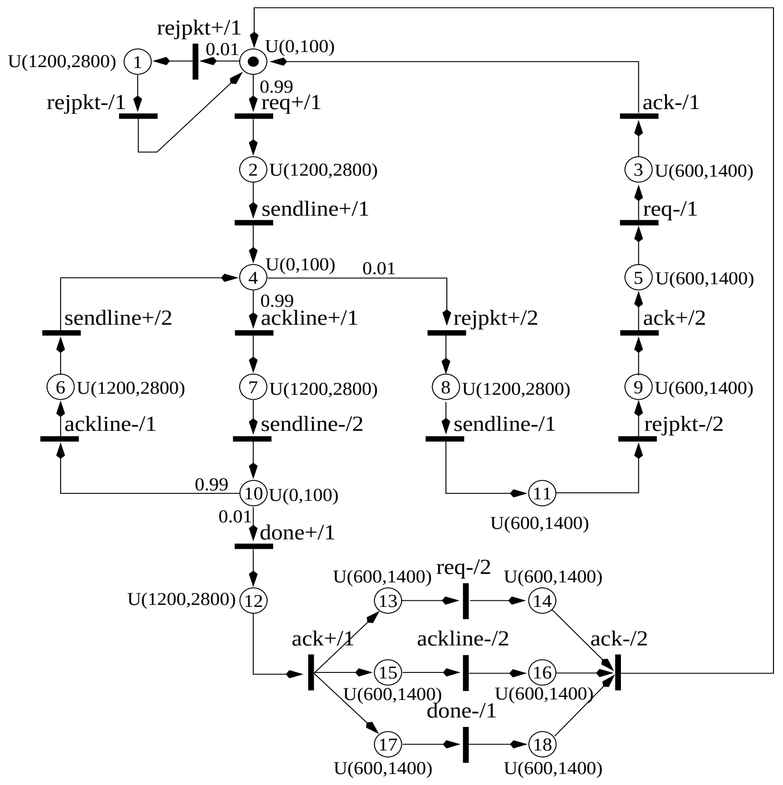

**Figure 2.2**. The TSPN for the sbuf-send-pkt2 circuit from the post office chip.

Figure 2.2 shows the *TSPN* for the sbuf-send-pkt2 circuit from the post office chip [6]. The sbuf-send-pkt2 circuit is used as a working example throughout the development of the stochastic cycle period. The circuit contains three free choice places: 0 (initially marked), 4, and 10. The choice distributions on each place are annotated with a 0.99 and 0.01 split denoting that most of the time the circuit is processing packets and that the completion of the transmission or the rejection of a packet is rare when compared to the receiving of packets. For simplicity and clarity of presentation, all delays are described with uniform distributions.

## 2.2 Reachablility

The *timed reachability set* of markings from $M_o$ is the set of markings that are reachable subject to the bounded timing constraints on places as defined by $\Delta$ and is found using methods described in [3, 4, 12] which are implemented in ATACS. The *reachability graph* is a labeled directed graph described by the tuple $RG = \langle \Phi, \Gamma \rangle$, where

- $\Phi$ *is the set of timed reachable markings, and*
- $\Gamma \subseteq \Phi \times \Phi \times T$ *is the edge relationship denoted by* $(M, M', t)$ *or* $M \xrightarrow{t} M'$.

A *RG* derived from a *TSPN* can be manipulated in many different ways. The *excited markings* of $t$ in a *TSPN* is the set of all markings that have tokens sufficient for $t$ to be untimed enabled to transition or $EM(t) = \{M \in \Phi \mid t \in T_e(M)\}$. The predecessor of a marking $M$, $Pred(M)$, is the set of all markings which lead to M; and successors of $M$, $Succ(M)$, is the set of all markings which can be reached from $M$ by firing a single enabled transition in $M$. Formally, $Pred(M') = \{M \in \Phi \mid M \xrightarrow{t} M' \in \Gamma\}$ and $Succ(M) = \{M' \in \Phi \mid M \xrightarrow{t} M' \in \Gamma\}$.

The *RG* generated by **ATACS** for the sbuf-send-pkt2 is shown in Figure 2.3. Because of the simplicity of the circuit, its *RG* closely resembles its *TSPN* specification. The only difference is seen in the concurrent burst of signals that result from the transition *ack+/1*, where the *RG* shows all possible interleaving of the three concurrent signals *req-/2*, *ackline-/2*, and *done-*. In this example,

$$\begin{aligned} EM(\mathit{done}\text{-}) &= \{\{13,15,17\},\{13,16,17\},\{14,16,17\}\} \\ Succ(13,15,17) &= \{\{14,15,17\},\{13,16,17\},\{13,15,18\}\} \\ Pred(13,15,17) &= \{\{12\}\}\,. \end{aligned}$$

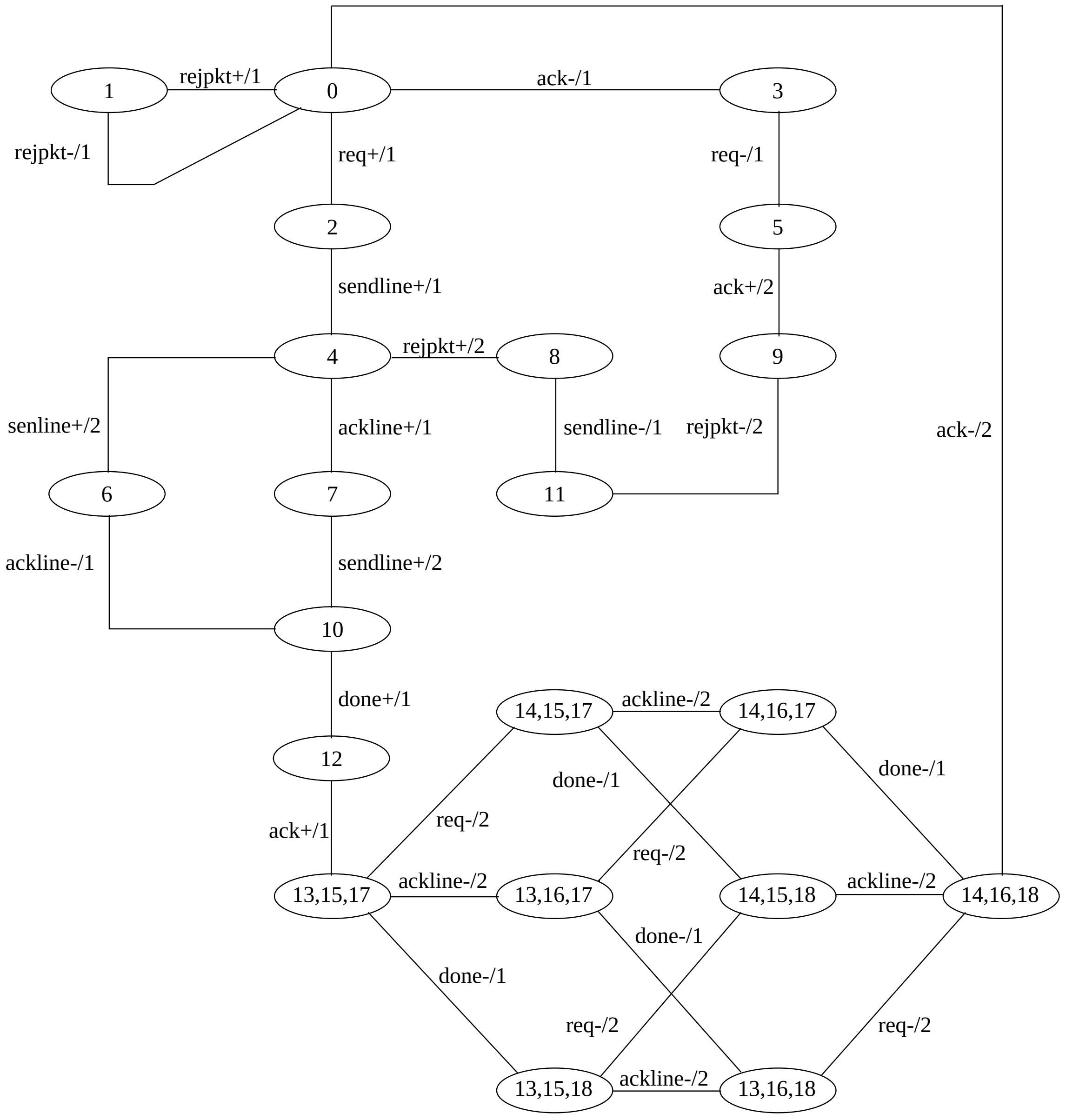

**Figure 2.3**. The RG for the sbuf-send-pkt2 circuit from the post office chip.

# CHAPTER 3

# TRANSITION PROBABILITIES

In a highly concurrent timed system, each marking of the $RG$ can have multiple transitions leaving it. Transition probabilities denote the relative importance or frequency of each of these transitions. The probabilities are used to identify highly probable markings, as well as the relative importance of each transition in the timed circuit specification.

In a $RG$, the probability of a transition from a given marking is *not dependent* solely on the given marking. This stems from the natural dependence of transitions on the places in their preset and the amount of time those places have had tokens. In an implemented free running timed circuit, when the circuit arrives at a marking $M$, each place $p \in M$ has a real valued timer associated with it showing the amount of time which must expire before its token is available to a transition and a transition cannot be enabled to fire until a sufficient number of tokens in $\bullet t$ have become available. The value that each of these timers must achieve is determined by the function $\Delta(p)$ at the time when $p$ is added to a marking following the firing of a transition. From this point, an arbitrary number of paths exist which arrive at markings that have a sufficient number of tokens available to fire $t$. In a concurrent system, a marking $M$ can be reached by a myriad of traces with each trace having a unique set of available tokens and timer values. This implies that reaching a state where previous trace history can be ignored is highly unlikely, making the transition probabilities from the marking $M$ dependent on the order of markings the circuit followed to get to $M$ and the unique timer values associated with each $p \in M$. This type of *memory* or trace dependence in the system makes timed circuits computationally challenging to analyze. If the trace dependence can be mitigated, then transition probabilities can be efficiently derived.

To remove trace dependence of transition probabilities in $RG$s, the *Markov memoryless* property is enforced. The Markov memoryless property or *single-step* transition probability states that given the edge $M_n \xrightarrow{t} M_{n+1} \in \Gamma$, then the probability of $M_{n+1}$ is solely dependent on the current marking $M_n$ and not any markings that preceded $M_n$. Formally, if $\mathbf{M}$ is a random variable for a marking, then

$$Pr\{M_{n+1} \mid M_0, M_1, \ldots, M_n\} = Pr\{M_{n+1} \mid M_n\},$$

where the $n$ subscript denotes a transition firing moving the circuit into a new marking [10, 20]. In this way, time is not used to evidence progress in the system, rather progress is monitored by the firing of transitions.

For the memoryless property to hold in a timed circuit, $Pr\{M_{n+1} \mid M_n\}$ must be defined to return the average value of the probability of moving to $M_{n+1}$ by considering all possible paths that arrive at $M_n$. In this way, the single-step transition probabilities represent an expected value elicited from all possible trace scenarios. From this, timed circuits are modeled as being memoryless by considering variable length traces and variable numbers of timer values while deriving the single-step transition probabilities in the $RG$. Depending on the amount of memory used, the transition probabilities move closer to, or farther away from, their exact average values. Formally, the exact average values for the single-step transition probabilities are defined as follows: let $\sigma \in \S_n$ be a trace in the set of all traces of length $n$ that are possible in the $RG$. Let $count(\sigma, \gamma)$ be a function that returns the number of times the edge $\gamma$ occurs in $\sigma$. Then, the average value of the single-step transition probability for a given trace $\sigma$ is

$$Pr\{M_{n+1} \mid M_n, \sigma\} = \frac{count(\sigma, M_n \xrightarrow{t} M_{n+1})}{\sum_{\forall M_n \xrightarrow{t'} M' \in \Gamma} count(\sigma, M_n \xrightarrow{t'} M')}.$$

which is the sum of all edges in the trace moving from $M_n$ to $M_{n+1}$ divided by all edges in the trace starting at $M_n$. Now let the function $Pr\{\sigma\}$ return the probability of a trace $\sigma \in \S_n$, then the exact average value of the single-step transition probabilities can be defined as the expected value of trace dependent single-step transition probabilities as the trace length approaches infinity or

$$Pr\{M_{n+1} \mid M_n\} = \lim_{n \to \infty} \sum_{\forall \sigma \in \S_n} Pr\{M_{n+1} \mid M_n, \sigma = \sigma\} \cdot \mathbf{Pr}\{\sigma\}.$$

This paper presents three methods of estimating transition probabilities in a $RG$ and each reflects a different level of memory used in the system. The *stochastic simulation* method has memory that is proportionate to the amount of time that the simulation is run and the number of times that the simulation is run. Transition probabilities from simulation most closely match their actual average values. An *exponential approximation* is at the other extreme where no memory is used in calculating the transition probabilities. In the middle is the *burst-mode heuristic*, which has a limited amount of memory that it uses when finding the transition probabilities. Each of these methods have merits which are described in later sections.

Transitions probabilities are referenced by the function $\Psi : \Phi \times \Phi \rightarrow \Re$, known as the transition probability function, and $\Psi(M, M')$ is the probability of moving from $M$ to $M'$ on the next transition $t$ given that the system is currently in $M$.

## 3.1 Simulation of Transition Probabilities

Stochastic simulation to find the transition probability function uses a maximum amount of memory to compute average values of the single-step transition probabilities by using a very long simulation trace. When using a simulation based approach to compute system metrics, regenerative stochastic simulation is typically preferred to avoid serial correlation. Regenerative stochastic simulation is often referred to as the *Monte-Carlo* method and is the process of running *several* random experiments until the average value of the experiments converges [18].

The Monte-Carlo method is not easily implementable for a $RG$. This is due to the continuous nature of timed circuits. Requisite to the Monte-Carlo method is the random experiment. The random experiment must have a definitive beginning and ending state. This implies that the system must reach a state where past history is irrelevant to future behavior. From this point, another random experiment can be started that is independent of the experiment that preceded it. For a $RG$, the system would need to start from some initial marking $M_o$ and simulate transition firings until it returns back to $M_o$ with identical timer values for all $p \in M_o$. Unfortunately, the probability of ever reaching a recurrent marking $M_o$ with identical timer values for all $p \in M_o$ is extremely small for most systems. Therefore, the Monte-Carlo method is not easily implemented for a general class of timed circuits without first identifying a recurrent marking that when reached by the system will always have identical timer values for all its places. Such markings do exist in timed burst-mode specifications with input to output causality.

A timed burst-mode circuit has several markings that serve as synchronization points where all timer values are set to zero (i.e. a new burst becomes enabled). With a timed burst-mode circuit, it is possible to begin and end a random experiment on a marking where a new burst in enabled, but this is a very narrow class of circuits that is to restrictive to specify highly concurrent systems. Therefore, to implement the Monte-Carlo method for a broad range of design styles it is necessary to first identify a burst marking where all places in the marking have timer values of zero. The frequency of these markings in general circuits and how they are identified in the *RG* are open research questions for future analysis. Therefore, stochastic simulation is used over the Monte-Carlo method in this application.

The stochastic simulation of a *TSPN* with a given *RG* is the process of firing transitions and flowing tokens from the initial marking $M = M_o$ as illustrated by the algorithm Simulate in Figure 3.1. In the algorithm Simulate, $Q_p$ is a set of $(p, \delta)$ pairs that acts as an ordered queue to store places that are not yet available to transitions. If when selecting a place to make available from the delay set $Q_p$ multiple places are found to have the same $\delta$ value, then the system randomly picks a place to insert into the available set $P_a$ using a uniform distribution based on the cardinality of the set of places with equal delay times. This random break of a tie is necessary to deal with places that have singular distributions on the same point. In this way, each place is equally likely to become available first. The simulation process proceeds until the edge counters divided by their appropriate marking counters converge to within a user specified tolerance. By convergence it is meant that the difference between results at time $n$ and the results at time $n + k$ where $k$ is some fixed amount of time or number of transitions falls within a user specified tolerance of percent change.

As an example of the simulation process, Figure 3.2 shows a portion of the *TSPN* and *RG* for the sbuf-send-pkt2 circuit introduced in Section 2.2. The simulation has been running for some time. $Q_p$ and $P_a$ are presently empty and the current marking $M$ is $\{10\}$. On line 2 of the Simulate algorithm, 10 is the only place in $M$ so line 3 assigns $\delta = 35$, which is a random value sampled from $\Delta(10) = U(0, 100)$. Line 4 sets $Q_p = \{(10, 35)\}$. Line 5 sets $(p', \delta') = (10, 35)$ as the next token to become available, $Q_p$ is made empty by line 6, $P_a = \{10\}$ by line 7, and time is advanced on line 8. The enabled set $T_e(\{10\})$ is $\{$*done+/1,ackline-/1*$\}$ moving the algorithm past the conditional clauses of lines 9 and 10 to line 11. The $Pr\{$*done+/1* $\mid \{10\}, 10\} = 0.01$ and $Pr\{$*ackline-/1* $\mid \{10\}, 10\} = 0.99$. For this example, the random sample from the distribution returns $t$ to be *done+/1.* The edge counter for $(\{10\}, \{12\},$ *done+/1*$)$ and the marking counter for $\{10\}$ is incremented in the *RG* and $M$ is set to $\{12\}$ on lines 14 and 15. The preset of *done+/1* is $\{10\}$

**Algorithm 3.1.1** Simulate( TSPN, RG ) {

```
1:   $M = M_o$ ; $Q_p = \emptyset$ ; $P_a = \emptyset$ ;
     do {
2:       foreach $p \in M$ . $p \notin Q_p$ {
3:           $\delta$ = a random delay value from the distribution $\Delta(p)$;
4:           $Q_p = Q_p \cup (p, \delta)$;
         }
5:       select $(p, \delta) \in Q_p$ . $\forall (p', \delta') \in Q_p$ . $\delta \leq \delta'$
6:       $Q_p = Q_p - (p, \delta)$;
7:       $P_a = P_a \cup p$;
8:       advance time by subtracting $\delta$ from each entry $(p', \delta') \in Q_p$ ;
9:       if $|T_e(P_a)| > 0$ then {
10:          if $|T_e(P_a)| > 1$ then {
11:             $\forall t \in T_e(P_a)$ use $Pr_c\{t \mid P_a, p\}$ to construct a probability distribution and
12:              select $t \in T_e(P_a)$ according to the distribution;
             }
13:          else $t$ is set to the element in $T_e(P_a)$ ;
14:          record( $M \xrightarrow{t} M'$ );
15:          $M = M'$ ;
         }
16:       $P_a = P_a - P_a \cap \bullet t$ ;
17:  }while(not converged);
}
```

**Figure 3.1**. Procedure to simulate a *TSPN* and RG.

and $P_a$ is once again set to empty on line 16. At this point, the system has not yet converged and the algorithm returns control to line 2. The algorithm processes place 12 and fires *ack+/1*, after which $M = \{13, 15, 17\}$. The loop on line 2 executes on each place in $M$ so when it completes $Q_p = \{(13, 650), (15, 1230), (17, 1333)\}$. Place 13 is selected to be made available and time is advanced by 650 leaving $Q_p = \{(15, 580), (17, 683)\}$. Transition *req-/2* fires and place 14 is added to $M$ and a new entry in $Q_p$ is added making $Q_p = \{(15, 580), (17, 683), (14, 630)\}$. Place 15 is made available firing *ackline-/2* and $Q_p$ is updated as $\{(17, 103), (14, 50), (16, 1345)\}$. Place 14 is the next place to become available and it does not cause a transition to fire, so the simulation continues by making 17 available, which does fire a transition and adds a new entry to $Q_p$ making $Q_p = \{(17, 1242), (18, 756)\}$. In this way the Simulate algorithm makes available places 18 and 17 to transitions, at which point *ack-/2* fires moving the system back to the initial marking ready to start the entire process again. An interesting note about this example is that it contains properties necessary for the Monte-Carlo method, which means that there

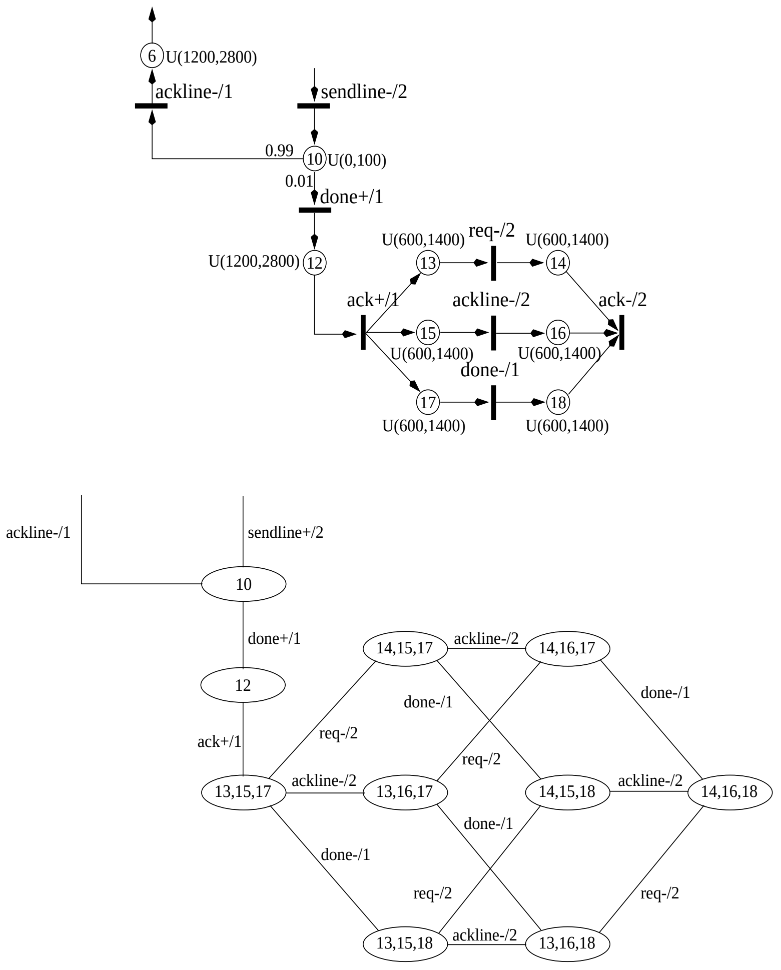

**Figure 3.2**. A portion of the *TSPN* and *RG* for the sbuf-send-pkt2 circuit.

exists a regenerative marking in the *RG* where past history is irrelevant. Such a marking is the initial marking. Each time the system reaches the initial marking a new simulation could be started which is independent of the previous trace history. Unfortunately, such markings do not always exist in a general class of circuits.

## 3.2 Exponential Approximation

The exponential approximation method does not consider any previous history in the *RG* when calculating transition probabilities and in essence, models the *TSPN* as a *Continuous Time Markov Chain* (CTMC) [20]. A CTMC is represented with an infinitesimal generator or transition rate function $Q$, where $Q(M, M')$ is the rate of occurrence of $M$ going to $M'$ and the rate is inversely related to the expected value of a negative exponential random variable. By virtue of the negative exponential distribution, this method models the system as being truly *memoryless.*

In the simulation formulation of Section 3.1, the system considers the time at which places become available to transitions. When a sufficient number of places are available to a transition to enable it to fire, it fires immediately. The ultimate goal of the simulation is to record the firing of transitions and movements of the system from one marking to another in the $RG$. The movement of places from a marking into the available set is secondary information only used to determine the actual firing of transitions. Similarly, in a CTMC model, information on the probability of the system moving from one marking to another is desired. To gain this type of insight, the transition rate function must consider the rate of occurrence of each transition in a marking, rather than the rate at which places become available in a marking. While the rate of places becoming available in a marking indirectly determines the rate of transitions from the marking, the two CTMC models yield very different information about the longterm and transitional behavior of the system. If the CTMC considered the rates of places becoming available, it could not report movement between markings in the $RG$, which is the necessary information for the transition probabilities. Why is this the case$\Gamma$ In order to model places becoming available in a CTMC, a state in the CTMC would need to be a marking paired with an available set. This allows the CTMC to change states as places in a marking move into the available set. The problem now becomes a question of mapping transition probabilities between states in the CTMC to transitions probabilities between markings in the $RG$. Methods of accomplishing the mapping without introducing new behaviors in the system are as yet, elusive and not obvious. Therefore, it is necessary to consider the rate of transitions from markings in the the CTMC model.

For a CTMC modeling the rate of transitions from markings, it is requisite to assume that all places in each marking necessary to enable a transition to fire from that marking become available together at the same time, causing the transition to instantly happen. This rate is the rate of occurrence of a transition from that marking and is used to build the transition rate function for the CTMC. Ignoring conflict, the rate of a transition can be defined as the inverse of the expected time at which a transition's entire preset becomes available ($E[\Delta(\bullet t)]$). From this it is implied that the exponential approximation assumes that $\forall (p, p') \in \bullet t . \Delta(p) = \Delta(p') = \Delta(t)$. Meaning that the expected time for $p$

to become available is equal to the expected time for $p'$ to become available for all $(p, p')$ in the preset of $t$. The result of this assumption is that the expected time to become available for any $p$ in $\bullet t$ is the expected time at which the transition $t$ will occur. Using this assumption, the function $\Delta(t)$ is defined to return $\Delta(p)$ where $p$ is any place in $\bullet t$ and the expected value of $\Delta(t)$ $(E[\Delta(t)])$ returns the expected time at which the transition $t$ will occur. This assumption limits the ability to associate unique distributions with causal transitions while deriving transition probabilities. With this, the transition rate function $Q$ for a CTMC model of a *TSPN* is defined as

$$Q(M, M') = \begin{cases} \frac{1}{E[\Delta(t)]} & \text{if } M \xrightarrow{t} M' \in \Gamma, \\ -\sum_{M'' \neq M} Q(M, M'') & \text{if } M = M', \\ 0 & \text{otherwise.} \end{cases}$$

To incorporate conflict into the *CTMC* model, it is necessary to correctly distribute rate amongst conflicting transitions. A choice place as presented in Section 2.1 is a location in the *TSPN* where multiple transitions are enabled by a common place. At such a point, the exponential approximation again assumes that all places necessary to fire any of the conflicting transitions become available at the same time. Therefore, all conflicting transitions fire at the same rate. However, this is incorrect since only one of the enabled conflicting transitions can be allowed to fire and each should fire at a different rate. Therefore, it is necessary to adjust the rate for each transition to reflect the user supplied firing distribution described by the $\Upsilon$ function. For each transition $t$ enabled in conflict by a common place $p$, the rate of $t$ is found by multiplying the rate of $p$ by the normalized value in $\Upsilon(p, t)$. This essentially creates from the rate of the place the rates of the transitions enabled by the place, which is consistent with the CTMC model. The function $Q$ can now be defined to correctly deal with choice as follows:

$$Q(M, M') = \begin{cases} \frac{1}{E[\Delta(t)]} & \text{if } M \xrightarrow{t} M' \in \Gamma \wedge \\ & C_p(M, M') = \emptyset, \\ \left(\frac{1}{E[\Delta(C_p(M,M'))]}\right) \cdot \frac{\Upsilon(C_p(M,M'),t)}{\sum_{t' \in C_p(M,M')\bullet} \Upsilon(C_p(M,M'),t')} & \text{if } M \xrightarrow{t} M' \in \Gamma \wedge \\ & C_p(M, M') \neq \emptyset, \\ -\sum_{M'' \neq M} Q(M, M'') & \text{if } M = M', \\ 0 & \text{otherwise.} \end{cases}$$

The function $C_p(M, M')$ is defined in Section 2.1 to identify marking pairs that are connected via a transition enabled by a choice place and returns that place if it exists, otherwise it returns the empty set.

As an example of the $Q$ function, consider the partial *TSPN* and *RG* for the sbuf-send-pkt2 circuit in Figure 3.2. For the marking $M = \{10\}$ containing a free choice, the function $C_p(M, \{12\})$ and $C_p(M, \{6\})$ both return the place $\{10\}$, so

$$\begin{aligned} Q(M, \{12\}) &= \frac{1}{E[\Delta(10)]} \cdot \Upsilon(10, \textit{done+/1}) \\ &= \frac{1}{50} \cdot 0.01 \\ Q(M, \{6\}) &= \frac{1}{E[\Delta(10)]} \cdot \Upsilon(10, \textit{ackline-/1}) \\ &= \frac{1}{50} \cdot 0.99 \end{aligned}$$

Similarly, for the marking $M = \{13, 15, 17\}$, $M$ does not contain a choice place so $Q$ values do not use case two, but use cases one and three as follows: $Q(M, \{13, 15, 18\}) = Q(M, \{13, 16, 17\}) = Q(M, \{13, 15, 18\}) = \frac{1}{1000}$. $Q(M, M) = -\frac{3}{1000}$.

Since $Q$ is defined to return rates of transitions in a system, transition probabilities in a marking $M$ are related to the sojourn time or the amount of time that the system spends in $M$. Thus, if the system is considered after some discrete amount of time, there is a non-zero probability of the system remaining in its current marking and not changing state, since there is a real valued rate of transition to itself. In order to remove the self-loop transition rate from the system, the sojourn time in each marking is extended to be infinitely long. It his way, the system is forced to fire a real transition leaving the marking. Time has now been abstracted away from the model and only transitions are considered as progress markers, creating the *Embedded Markov Chain* (EMC) in $Q$. The EMC of $Q$ is a discrete Markov process that represents the probability of moving from a given state to another state on the next transition, which is precisely the single-step transition probabilities of the system. The function $\Psi$ is now defined as the EMC of $Q$ by the following

$$\Psi(M, M') = \begin{cases} \frac{Q(M,M')}{-Q(M,M)} & \text{if } M \neq M', \\ 0 & \text{otherwise.} \end{cases}$$

As an example of the $\Psi$ function, for the partial marking graph shown in Figure 3.2, the values of $\Psi$ for the marking $M = \{13, 15, 17\}$ can be found as: $\Psi(M, \{13, 15, 17\}) = \Psi(M, \{13, 16, 17\}) = \Psi(M, \{13, 15, 18\}) = \frac{1000}{3000}$, and $\Psi(M, M) = 0$.

## 3.3 Burst-mode Heuristic

The burst-mode heuristic definition of $\Psi$ uses a limited amount of memory or history in calculating transition probabilities. This stems from the observation that burst-mode circuits contain synchronization points between input and output bursts where all timers on places are set. As new bursts become enabled, it is only necessary to consider firing sequences from the enabling points of the bursts to calculate transition probabilities. For a pure burst-mode specification [6, 17] (one that does not include *directed don't cares*), the transition probabilities are exact, since traces before bursts do not contain information pertinent to the firings in the current burst. Exactness is assured with the assumption of output to input causality in the burst-mode specification and that all outputs within a burst enable the same input burst. With this assumption, as a specification digresses from the burst-mode ideology, the transition probabilities may quickly diverge from their exact values. This is a direct result of the burst mode heuristic assuming that timers on places enabling transitions in a burst are all set at the same time, because all transitions became enabled together as part of the burst. As an example, a portion of the sbuf-send-pkt2 circuit is specified as two communicating burst-mode state machines as shown in Figure 3.3 where the input *ack+/1* enables the outputs *req+/2*, *ackline/2*, and *done+*; and together they enable the input burst of *ack-/2*. In this example, there is no notion of one timer being set at a different instance than another timer within the same burst. The behavior of the heuristic in a non-burst-mode circuit is to try and identify bursts in the $RG$. However, in a non-burst-mode circuit, new transitions may become enabled inside of what the heuristic considers an uncompleted burst. At this point, the heuristic believes it has found a new burst, so all transitions in the new burst have the timers in their presets reinitialized (i.e. a completely new burst has become enabled at that point).

Before the burst-mode heuristic is defined, it is necessary to present a few support definitions and functions. A *path* is defined as a sequence of markings connected by transitions (i.e. $(M_1, M_2, \ldots, M_n)$ such that $\forall i < n \ . \ M_i \xrightarrow{t} M_{i+1} \in \Gamma$). A *burst-path* is defined as a path such that the transitions enabled in each marking is a superset of the enabled transitions in the next marking in the sequence (i.e. $(M_1, M_2, \ldots, M_n)$ is a path such that $\forall i < n \ . \ T_e(M_i) \supset T_e(M_{i+1})$). A *maximal burst-path* is a burst-path such that there doesn't exist a marking with an edge to the first marking in the sequence whose enabled transitions are a superset of the enabled transitions found in the first marking in the sequence. (i.e. $(M_1, M_2, \ldots, M_n)$ is a burst-path such that $\neg\exists M' \ . \ M' \xrightarrow{t} M_1 \in \Gamma \land T_e(M') \supset T_e(M_1)$). The superset burst function, $Sup(M)$, is a function that returns the marking $M'$ which contains the maximal superset of enabled transitions in $M$ defined as

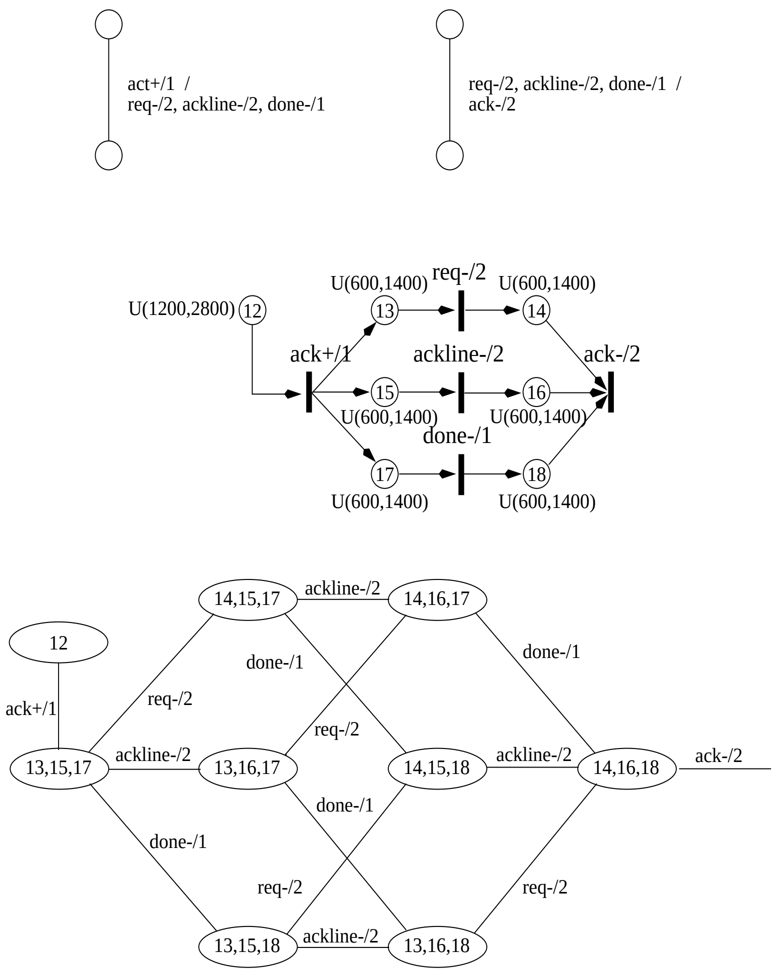

**Figure 3.3**. A partial burst mode specification of the sbuf-send-pkt2 circuit with its *TSPN* and *RG*.

$$Sup(M) = M_1 \;.\; (M_1, M_2, \ldots, M_n) \textit{ is a maximal burst-path containing } M.$$

If it is ever the case in a non-burst-mode specification that it is possible to have multiple maximal burst-paths for $M$, the function arbitrarily selects a single $M'$ to return as $Sup(M)$. In the implementation, the selection is based on the order in which the states are examined during the calculation and is completely arbitrary, but once the selection is made it is consistent in returning the same value from that moment forward. Also for a non-burst-mode specification, if a marking $M'$ is returned from $Sup(M)$ and $M'$ is a marking where transitions have become enabled inside of an incomplete burst, then all timers are reset in $M'$ in the calculation of $\Psi$ values from $M$. The effect of this is that places that have been waiting for some time to become available are assumed to have just barely been added to the marking $M'$ as part of a new burst starting at that point, thus moving them from a higher probability of becoming available to a lower probability. The fired function, $T_f(M, M')$, returns the set of transitions which must have fired for the system to move from the marking $M$ to $M'$ and is defined as

$$T_f(M', M) = \begin{cases} T_e(M') - T_e(M) & \text{if } T_e(M) \subset T_e(M'), \\ \emptyset & \text{otherwise.} \end{cases}$$

Note that $T_f$ returns the empty set when $Sup(M)$ is not strictly a superset of $M$. And finally, the probability function, $Pr(t \mid M)$, returns the probability of a transition $t$ given a marking $M$ defined as

$$Pr(t \mid M) = \int_0^\infty \Delta(t) \left( \prod_{t' \in T_e(M), t' \neq t} (1 - F_{t'}(t)) \right) \left( \prod_{t'' \in T_f(Sup(M), M)} F_{t''}(t) \right) dt,$$

where $F_{t'}(t) = \int_0^t \Delta(t')dt'$, $F_{t''}(t) = \int_0^t \Delta(t'')dt''$, and each integral is calculated numerically using Simpson's method [19]. Intuitively, the function $Pr(t \mid M)$ calculates the probability of $t$ firing given that all other transitions enabled in $M$ do not fire and given that all transitions in the fired set for $M$ do fire. As with the exponential approximation in Section 3.2, it is assumed that all $p \in \bullet t$ have equal $\Delta(p)$ functions and therefore $\Delta(t) = \Delta(p)$ where $p$ is any place in $\bullet t$. In this integral, it is important to note that each internal integral must be evaluated at each step of the external integral making the probability calculation somewhat costly and that the integral must be carefully partitioned to avoid integrating across discontinuities caused by the bounded nature of the *pdf*s. The burst-mode function definition of $\Psi$, ignoring choice, is now constructed as follows:

$$\Psi(M, M') = \begin{cases} Pr\,(t \mid M) & \text{if } M \xrightarrow{t} M' \in \Gamma, \\ 0 & \text{otherwise.} \end{cases}$$

As an example of the burst-mode heuristic, ignoring choice, for the *TSPN* and *RG* in Figure 3.3, the calculation of the probability for $M = \{14, 15, 17\}$ and $M' = \{14, 15, 18\}$ proceeds as follows: since $M \xrightarrow{b\uparrow} M' \in \Gamma$, the first case holds for $\Psi$. For the integrals, $T_e(M) = \{$*done-/1*,*ackline-/2*$\}$, $Sup(M') = \{13, 15, 17\}$, and $T_f(Sup(M'), M) = \{$*req-/2*$\}$. Let $t =$ *done-/1*, $t' =$ *ackline-/2*, and $t'' =$ *req-/2*, then

$$\begin{aligned} \Psi(M, M') &= Pr(t = \textit{done-/1} \mid M = \{14, 15, 18\}) \\ &= \int_{600}^{1400} \Delta(t) \left(1 - \int_{600}^{t} \Delta(t' = \textit{ackline-/2})dt'\right) \cdot \end{aligned}$$

$$\begin{aligned}
&\left(\int_{600}^{t} \Delta(t'' = \textit{req-/2})dt''\right) dt \\
= &\int_{600}^{1400} \Delta(p_{17}) \left(1 - \int_{1}^{t} \Delta(p_{15})dt'\right) \left(\int_{2}^{t} \Delta(p_{13})dt''\right) dt \\
= &\ 0.5,
\end{aligned}$$

where $\Delta$ maps the places $P_{17}$, $P_{15}$, and $P_{13}$ to the uniform distribution $U(600, 1400)$ as shown in Figure 3.3.

Choice is introduced into the burst-mode heuristic by altering how conflicting transitions are dealt with in a manner similar to that presented in Section 3.2. In essence, a perspective shift must take place to first consider the probability of a single place becoming available causing a single transition to fire, rather then considering each enabled transition separately. Why is this necessaryΓ Because in reality, only a single transition is allowed to fire when multiple transitions are enabled in conflict. The probability of each enabled conflicting transition is first dependent on the probability of the place enabling the transitions becoming available. Therefore, if the system first calculates the probability of the place enabling the conflicting transitions becoming enabled, it then can correctly assign probabilities to each of the conflicting transitions using the $\Upsilon$ function as shown in Section 3.2. This approach preserves the semantics of the *TSPN* model by first considering the probability of the place becoming available and then weighting that probability with the normalized user supplied firing distribution in $\Upsilon$. For a place $p$ enabling conflicting transitions, the probability of $p$ given a marking $M$ where $p \in M$ is

$$Pr(p \mid M) = \int_{0}^{\infty} \Delta(p) \left(\prod_{t' \in (T_e(M) - p\bullet)} (1 - F_{t'}(t))\right) \left(\prod_{t'' \in T_f(Sup(M), M)} F_{t''}(t)\right) dt.$$

The complete definition of $\Psi$ for the burst-mode heuristic handling choice can now be defined as follows:

$$\Psi(M, M') = \begin{cases} Pr\left(t \mid M\right) & \text{if } M \xrightarrow{t} M' \in \Gamma \wedge \\ & C_p(M, M') = \emptyset, \\ Pr\left(C_p(M, M') \mid M\right) \cdot \frac{\Upsilon(C_p(M,M'),t)}{\sum_{t' \in C_p(M,M')\bullet} \Upsilon(C_p(M,M'),t')} & \text{if } M \xrightarrow{t} M' \in \Gamma \wedge \\ & C_p(M, M') \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

where the function $C_p(M, M')$ is identifies marking pairs that are connected via a transition enabled by a choice place and is formally described in Section 2.1.

# CHAPTER 4

# STEADY-STATE DISTRIBUTIONS

Transition probabilities provide information at a micro-level when considering the circuit. By this, it is meant that transition probabilities show the stochastic behavior of the circuit in a fixed area of the circuit. Given that the system is in the marking $M$, the transition probabilities give the probability of the system moving to the marking $M'$. This is information that is local to $M$ and does not shed light as to the relative importance of $M$ in the system.

For a clearer stochastic profile of the system, a macro-level understanding of the circuit behavior must be obtained. This is achieved through the *steady-state* distribution, which represents the long-term behavior of the system and indicates the markings that are important or highly probable. The steady-state distribution answers the question, "What is the relative import of a marking $M$ when compared to another marking $M'$ in the $RG\Gamma$" This macro-level information is paramount in deriving the stochastic cycle period and can be discovered by two different and distinct methods: simulation and Markovian analysis.

For a $RG$, its steady-state distribution is referred to as $\Pi$ and is the relation $\Pi \subseteq \Phi \times \Re$, where $\Pi(M)$ is the probability of $M$ relative to all the other markings in the system.

## 4.1 Stochastic Simulation

The method of stochastic simulation to find $\Pi$ is very similar to the method of stochastic simulation to define the transition function $\Psi$ in Section 3.1, only different values are tracked as the simulation proceeds. As transitions fire into new markings, the system increments the counter associated with the new marking, and increments a global counter which records the total number of transitions that have occurred in the simulation. Like the method described in Section 3.1, the simulation terminates when the steady-state distribution $\Pi$, found by dividing the marking counters by the total number of transitions in the system, converges to within a user specified tolerance.

## 4.2 Markovian Analysis

Since the transition function $\Psi$, in essence, is a model of the $RG$ as a *Discrete Time Markov Chain* (DTMC), where $\Psi$ is the transition matrix of the $RG$, it is possible to solve for steady-state distribution directly using Markov techniques. However, before $\Psi$ can be analyzed, it is necessary to insure that is is irreducible, otherwise $\Psi$ cannot be analyzed with Markovian methods. Note that for $\Psi$ representing the transition relations in a $RG$, $\Psi$ is ergodic meaning that the steady-state solution does change through time [1]. This ensures that a steady-state solution exists if the DTMC is irreducible.

By definition, a DTMC is irreducible if and only if it contains a single strongly connected component [20]. If $\Psi$ contains several strongly connected components, the relation is reducible and cannot be analyzed. Therefore, the $RG$ is divided into its strongly connected components $S_c$ before analysis proceeds using Tarjan's algorithm as presented in [19]. After the $RG$ has been partitioned, each connected set of markings $s_c$ is unioned into $S_c$. Each $s_c \in S_c$ is then analyzed and those which are not closed (meaning $s_c$ contains an edge $M \xrightarrow{t} M'$ such that $M \in s_c \wedge M' \notin s_c$) are removed from $S_c$ as transient components, because when considering the long-term behavior of the system, the markings contained in the non-closed sets have zero probability in $\Pi$. Therefore, each connected component $s_c$ remaining in $S_c$ can now be treated as a separate DTMC.

A strongly connected component $s_c$ with positive recurrent markings is said to be a *p-cyclic* DTMC. The periodicity of a p-cyclic DTMC can be exploited to converge to a steady-state solution $p$ times faster than normal iterative methods, where $p$ is the period of $s_c$. Therefore, $s_c$ is divided into its periodic classes and analyzed to find $\Pi$ using the reduced power method as presented in [20]. An important property of p-cyclic systems is that for all $M \xrightarrow{t} M' \in s_c$, the amounts of time taken for the $i^{\text{th}}$ and the $(i+1)^{\text{th}}$ cycle of $t$ are equal [5, 20], meaning that all transitions in the strongly connected component have the same cycle period. This property is utilized in Chapter 5 to solve for the stochastic cycle period of $s_c$. From this point on, it is assumed that after the transient analysis, only a single strongly connected component exists in $S_c$, which in practice, is usually the case. If it is ever the case that more than one strongly connected component exists with positive recurrent markings, then each component is analyzed and reported separately.

### 4.2.1 Power Analysis

Markov chain analysis to find a steady-state distribution for the probability of being in marking $M_i$ after any transition $t$ is done by taking the one-step transition probabilities $\psi_{ij}^{(1)}$ in $\Psi^{(1)}$ (represented in matrix form) and an initial marking distribution $\Pi^{(0)}$ (represented in vector form) and calculating the n-step transition probability of being in marking $M_i$ at transition step $n$ as $n \to \infty$, denoted by $\Pi$. There are several methods for calculating steady-state probabilities. Direct methods solve the equation $\Pi = \Pi\Psi$ subject to the constraint $\sum_i \pi_i = 1$, which can be done with Gaussian elimination, LU decomposition, or other methods. But these algorithms suffer possible rounding error accumulation and large memory resource requirements.

In general, $\Pi$ can be found using iterative methods. These methods are based on the following idea: If $\Psi^{(1)} = \Psi^1$ is the single-step transition probability matrix, then $\Psi^{(2)} = \Psi^2$ is the two-step transition probability matrix, and $\Psi^{(n)} = \Psi^n$ is the n-step transition probability where $\Psi^{(n)}$ means that $\Psi$ is raised to the $n^{\text{th}}$ power. Therefore, if $\Psi^n$ is calculated as $n \to \infty$, it will converge to a point where $\Psi^n = \Psi^{n+1}$. From this, the steady-state distribution can be calculated as: $\Pi = \Pi^{(0)}\Psi^n$, where $n$ is the number of steps required for $\Psi$ to converge. This approach will effectively find $\Pi$ in $n$ matrix-matrix multiplications and a single vector-matrix multiplication. Unfortunately, as the squaring process continues, the $\Psi$ matrix begins to fill, until it reaches its steady-state where it is completely full with $n \times n$ entries. This basic method obliterates the sparseness of the $\Psi$ matrix, has slow convergence rates, and suffers from rounding error accumulation [20].

The power method is an extension to the basic iterative method, but does not change the values in the $\Psi$ matrix. The power method stems from the following observation

$$\Pi^{(n+1)} = \Pi^{(n)}\Psi.$$

To find a steady-state solution to the Markov chain it is sufficient to apply successive vector-matrix multiplications on the $\Psi$ matrix until the following relation holds

$$\Pi = \Pi\Psi,$$

at which time a convergence point is reached yielding the steady-state distribution. While this might take more iterations, it is substantially more memory efficient than the previous method because it does not incrementally fill the $\Psi$ matrix, thereby taking full advantage of its sparse representation. Moreover, the initial values of $\Pi$ can be seeded with an approximation of the steady-state distribution, giving faster convergence rates because the algorithm starts closer to the actual solution.

### 4.2.2 Reduced Power Analysis

As stated previously, $RG$s often have the property that upon leaving a given marking $M_i$, there is some multiple of $p$ single-step transitions that return back to marking $M_i$, through all paths in the graph. When this occurs, the graph is said to be *periodic of period* $p$ or $p$*-cyclic* [20]. The periodic nature of the Markov chain can be used to divide the system into periodic classes, which are subsets of the original matrix. The steady-state analysis can then be applied to the smaller subsets of the transition matrix making the convergence rate go $p$ times faster, where $p$ is the cycle period of the Markov chain.

To apply the reduced power method on a periodic Markov chain, the transition matrix $\Psi$ must be changed into the normal form for periodic systems

$$\mathbf{S} = \begin{pmatrix} 0 & S_1 & 0 & \dots & 0 & 0 \\ 0 & 0 & S_2 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & S_{p-1} \\ S_p & 0 & 0 & \dots & 0 & 0 \end{pmatrix},$$

where $S_1 \cdots S_p$ are the *periodic classes* of the matrix $P$. A periodic class $S_i$ is a maximal set of markings that can be reached in exactly $i$ transitions from an initial marking or set of markings. The periodicity $p$ of a Markov chain is determined by the number of distinct periodic classes which it contains. The periodic classes are found using an algorithm presented in [20] shown in Figure 4.1. The algorithm systematically steps through the transition matrix from an initial marking $M_o$. As it traverses the matrix, it calculates periodic pre-classes that represent the sets of markings which are exactly $k$ steps from $M_o$ and stores each period class in the set $W$. When the algorithm finds a marking which is a member of a previous pre-class, it computes the period $p$ for the matrix as the difference between the two pre-classes plus one and updates the flow relation $F$, which shows the movement between periodic classes.. If multiple values of $p$ are found for the matrix, the actual period for the matrix becomes the greatest common denominator of all the different $p$ values. Once the $p$ value for the matrix is known, it can be used to fold the pre-classes into periodic classes, which can be arranged in the normal form $S$. For the $RG$ of a timed C-element in Figure 4.2, the algorithm proceeds by first setting $W_1 = \{0\}$ and the other variables as shown in step 1. The first loop on line 3 sets $M = 0$. The step of line 4 is executed two times for markings 1 and 2. Since both markings 1 and 2 are not found in any other pre-classes, after the loop on line 4 completes, $W_2 = \{1, 2\}$ and $F = \{(W_1, W_2)\}$. The algorithm proceeds in this manner until in completes with

```
Algorithm 4.2.1 Periodic_Analysis( TSPN, RG ) {
 1:  $W_1 = M_o$ ; $W = W_1$ ; $F = \emptyset$ ; $k = 1$ ;
 2:  $W_{k+1} = \emptyset$ ;
 3:  foreach $M \in W_k$ {
 4:      foreach $M' \in \Phi$ . $M \overset{\rightarrow}{t} M' \in \Gamma$ {
 5:          if $\exists k' \leq k$ . $M' \in W_{k'}$ then
 6:              $F = F \cup \{(W_k, W_{k'})\}$ ;
 7:          else
 8:              $W_{k+1} = W_{k+1} \cup \{M'\}$ ;
         }
     }
 9:  if $W_{k+1} \neq \emptyset$ then {
10:      $W = W \cup W_{k+1}$ ;
11:      $F = F \cup \{(W_k, W_{k+1})\}$ ;
12:      $k = k + 1$ ;
13:      Goto 2 ;
     } 14:   else
15:      return$(W, F, k)$ ;
}
```

**Figure 4.1**. Procedure to calculate the pre-classes of an RG.

$W = \{\ S_1 = \{0\},\ S_2 = \{1,2\},\ S_3 = \{3\},\ S_4 = \{4,5\}\}$ ($S_i$ denotes the location in the permuted normal form of a periodic system) and $F = \{\ (W_1, W_2),\ (W_2, W_3),\ (W_3, W_4),\ (W_4, W_1)\}$. The period is derived by first building a set of potential periods which are found by looking a the members of F and considering possible periods from members whose difference in subscript pairs is greater than 1 or formally,

$$\mathit{Period} = \{|k' - k| \mid (W_k, W_{k'}) \in F \land k' - k > 1\}.$$

The actual period of the $RG$ for the C-element is now computed as the greatest common divisor of the *Period* set. As a note, the sbuf_send_pkt2 circuit from the post office chip has a period of 2 and is not shown here as an example due to its complexity.

With the Markov system in the normal form $S$, the reduced power method is applied by stepping through each of the periodic classes as follows:

$$x_p = x_p S_p S_1 S_2 \cdots S_{p-1} = x_p R_p,$$

where

$$x_p^{(0)} = \frac{1}{(\text{Number of classes in } S_p)\, p.}$$

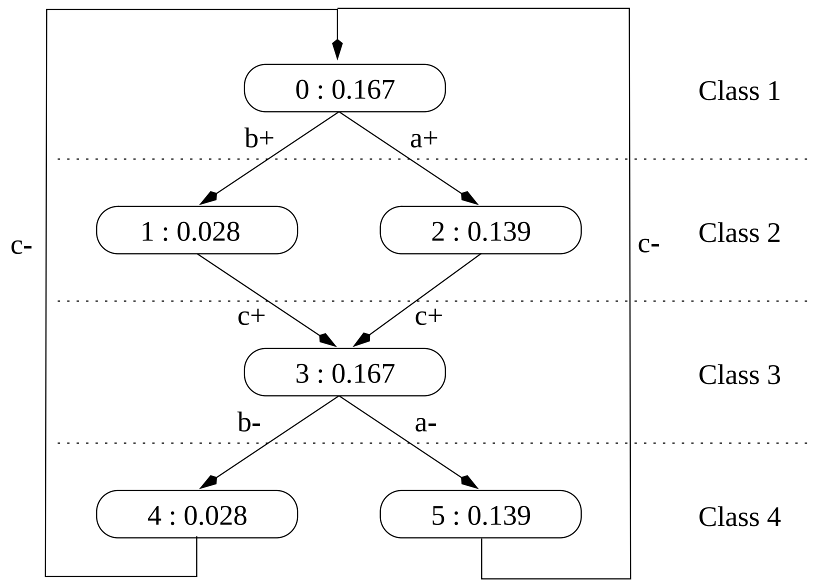

**Figure 4.2**. The probability annotated $RG$ for a timed C-element with periodic classes.

Once the steady-state distribution $x_p$ is known, the distributions for the other periodic classes are found from the following relation

$$\begin{aligned} x_1 &= x_p S_p \\ x_2 &= x_1 S_1 \\ \vdots & \quad \vdots \quad \vdots \\ x_{p-1} &= x_{p-2} S_{p-2} \\ x_p &= x_{p-1} S_{p-1}. \end{aligned}$$

The application of one step of the reduced power method is equivalent to $p$ steps of the power method. The number of computations performed on each iteration of the two methods is identical, but the reduced method converges $p$ times faster. Moreover, the power method is not able to handle periodic systems, thereby limiting its applications.

# CHAPTER 5

# STOCHASTIC CYCLE PERIOD

The *stochastic cycle period* is a metric that conveys performance information by weighting the contribution of each causal signal in the $RG$. The contribution of a trigger transition, in this application, is delay, but this could be set to other metrics such as power. A trigger or causal transition is the last transition to occur which causes another transition to fire (i.e. the late arriving signal on a gate) and is determined by the last $p \in \bullet t$ to be made available. Trigger signals must always be included in the timed circuit implementation. If other signals are required for hazard-freedom, those signals are referred to as context signals and by definition, can *never* be causal signals. Therefore, while their presence affects the delay of a gate, they never trigger the gate to transition. From this, the cycle period in a circuit is controlled by the set of trigger transitions that occur in the cycle.

The stochastic cycle period ($\rho$) is now defined as a weighted sum of cost metrics that represent the expense of a trigger occurring on the cycle according to some cost function,

$$\rho = \sum_{u,v \in T} w_{uv} \alpha_{uv},$$

where $w_{uv}$ is the relative weight of transition $u$ being trigger to $v$, and $\alpha_{uv}$ is the cost of $u$ being trigger for $v$. Since relative weights are shown on trigger transitions in the cycle, the circuit can be optimized to favor transitions which make significant offerings to the overall cycle period. Since each possible trigger in the cycle is considered and weighted, the metric provides a stochastic profile of the cycle period which reflects the notion of average cycle time.

The calculation of $\rho$, the stochastic cycle period, is a two stage affair. The first stage uses $\Psi$ and $\Pi$, the transition probabilities and steady-state distribution respectively, to derive the possible trigger transitions in the cycle of $s_c$ with the probability of those triggers actually occurring. The second stage uses the trigger set with their probabilities in a timing simulation of the *TSPN*. The timing simulation engenders $\rho$, which is a stochastic picture of triggers that regulate the cycle period of the *TSPN*.

## 5.1 Trigger Probabilities

The trigger probabilities show the probability of one transition causing another transition in the circuit implementation and the transitions by definition must appear in the synthesized circuit for the *TSPN*. If the probability of each trigger transition is known, triggers with high probabilities can be moved near the output of a gate. Furthermore, if a trigger is shown to have zero probability, it is possible to optimize the specification by tightening timing assumptions such that the trigger is no longer required in the implementation. The trigger probabilities, as well as the stochastic cycle period, are metrics created to expedite this type of aggressive circuit design

The trigger probability function relies upon the transition probability function $\Psi$ and the steady-state distribution $\Pi$ to calculate long-term transition probabilities. The long-term transition probabilities are then used in computing the actual trigger probabilities. Trigger probabilities are implemented by the function $\Theta : T \times T \to \Re$, where $\Theta(t, t')$ returns the probability of $t'$ being triggered by $t$ and is defined as

$$\Theta(t, t') = \frac{\sum_{M \xrightarrow{t} M' \in \Xi(t')} \Pi(M)\Psi(M, M')}{\sum_{M \xrightarrow{t''} M' \in \Xi(t')} \Pi(M)\Psi(M, M')},$$

where

$$\Xi(t') = \{M \xrightarrow{t} M' \in \Gamma \mid M \notin EM(t') \wedge M' \in EM(t')\}.$$

In other words, $\Theta(t, t')$ (i.e. the probability of $t$ causing $t'$) is the sum of the long-term probabilities of all edges moving into the excited markings for $t'$ on $t$, normalized by the sum of the long-term probabilities of all transitions moving into the excited markings for $t'$. The calculation is facilitated by the parameterized function $\Xi(t, t')$ which returns the set of marking pairs $(M, M')$ such that $M$ is not in the excited markings of $t'$ and $M'$ is in the excited markings for $t'$ and an edge $M \xrightarrow{t''} M'$ exists in the set of all edges $\Gamma \in RG$.

As an example of the function $\Theta$ consider the partial $RG$ shown in Figure 5.1. The calculation of $\Theta(rdy-, req+)$ proceeds as follows

$$\begin{aligned} \Theta(rdy\text{-}, req+) &= \frac{\Pi(M_5)\Psi(M_5, M_6)}{\Pi(M_5)\Psi(M_5, M_6) + \Pi(M_{13})\Psi(M_{13}, M_6) + \Pi(M_{15})\Psi(M_{15}, M_{14})} \\ &= \frac{0.06506}{0.06506 + 0.03483518 + 0.000105} \end{aligned}$$

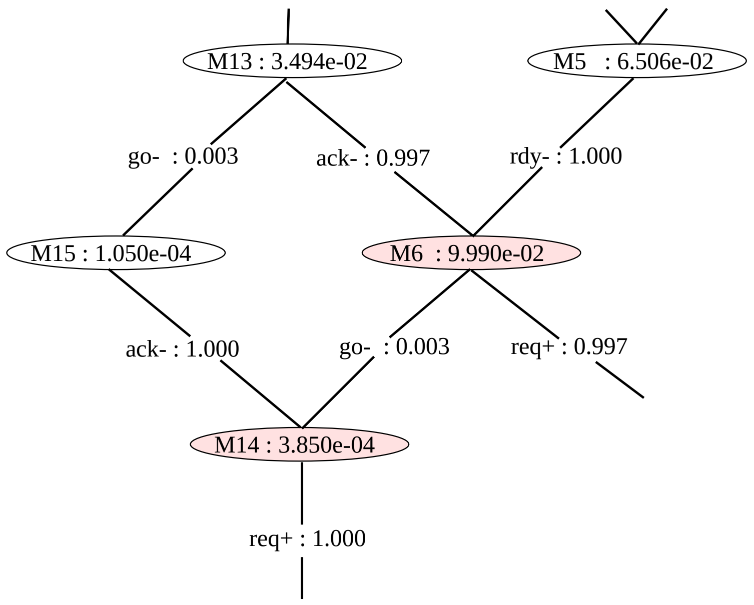

**Figure 5.1**. A partial $RG$ with its steady-state distributions and transition probabilities.

$$\approx \quad 0.651.$$

Similarly,

$$\begin{aligned}
\Theta(\textit{ack-},\textit{req+}) &= \frac{\Pi(M_{13})\Psi(M_{13},M_6)+\Pi(M_{15})\Psi(M_{15},M_{14})}{\Pi(M_5)\Psi(M_5,M_6)+\Pi(M_{13})\Psi(M_{13},M_6)+\Pi(M_{15})\Psi(M_{15},M_{14})} \\
&= \frac{0.03483518+0.000105}{0.06506+0.03483518+0.000105} \\
&\approx 0.349.
\end{aligned}$$

Although not shown by this example, it is interesting to note that although a transition is included in the trigger set of a gate implementation by synthesis, it is possible that the transition has a very small probability of ever occurring. This type of information can be used to further optimize timed circuits moving low probability triggers away from outputs or by alter the timing assumptions in the specification enough to exclude the signal from the trigger set. The easiest optimization is to move the low probability triggers away from output in the circuit implementation. A portion of the sbuf-send-pkt2 $RG$ is in Figure 5.2 showing the excitation region for the transition rejpkt+/1 and the markings moving into the excitation region with their steady-state and transition probabilities. The trigger probabilities for rejpkt+/1 are

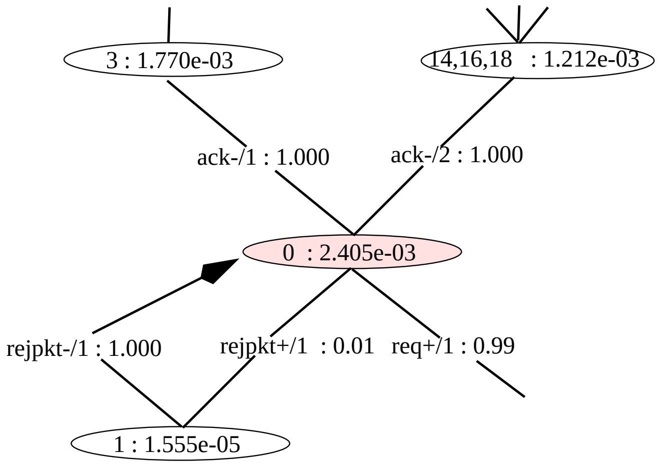

**Figure 5.2**. A portion of the sbuf-send-pkt2 $RG$ with its steady-state distributions and transition probabilities.

$$\begin{aligned}\Theta(\textit{ack-/1,rejpkt+/1}) &= 0.490\\ \Theta(\textit{ack-/2,rejpkt+/1}) &= 0.504\\ \Theta(\textit{rejpkt-/1,rejpkt+/1}) &= 0.006.\end{aligned}$$

An implementation of of the circuit for the rejpkt+/1 transition would be best optimized if the rejpkt-/1 transition is placed as close to the power rail as possible because rejpkt+/1 rarely, if ever, is triggered by the rejpkt-/1 transition. This transistor also becomes a good candidate for sizing if the rejpkt+/1 transition needs to be made faster. By increasing the size of the transistor switched by rejpkt-/1, the total resistance that must be driven to charge the rejpkt+/1 node is lowered. In this way, trigger signals can be better placed and sized in circuit implementations to yield better average case performance. The one missing link in this optimization methodology is a way to guide the designer to those signals in the circuit implementation that can have the most significant contribution to the overall performance of the circuit. To optimize every single signal in an implementation is wasteful, since resources are expended in optimizing signals that do not make significant contributions to circuit performance

## 5.2 Timing Simulation

The stochastic cycle period provides information to a designer showing which signals in a circuit actually control the overall circuit performance and is found through a timing simulation. The timing simulation concept is introduced to the asynchronous community in [5] as a method for finding the cycle period of a choice-free event-rule system. The idea is to record the time at which each transition occurs in the cycle and using the time of one instance of a transition in the cycle to the next instance of the same transition in the next cycle, it is possible to converge to an average cycle period if enough cycles are considered. Since the event-rule model is based on constant delays and does contain stochastic information, transitions that are triggered by multiple transitions use the *max* function to choose which transition would actually be the trigger. An example of a simple conflict-free timing simulation for a C-element shown in Figure 5.3 follows:

$$\begin{aligned}
\tau(c\downarrow, i) &= \rho i \\
\tau(a\uparrow, i) &= \tau(c\downarrow, i) + \alpha_{c\downarrow a\uparrow} \\
\tau(b\uparrow, i) &= \tau(c\downarrow, i) + \alpha_{c\downarrow b\uparrow} \\
\tau(c\uparrow, i) &= \mathit{max}\,(\tau(a\uparrow, i) + \alpha_{a\uparrow c\uparrow}, \tau(b\uparrow, i) + \alpha_{b\uparrow c\uparrow}) \\
\tau(a\downarrow, i) &= \tau(c\uparrow, i) + \alpha_{c\uparrow a\downarrow} \\
\tau(b\downarrow, i) &= \tau(c\uparrow, i) + \alpha_{c\uparrow b\downarrow} \\
\tau(c\downarrow, i) &= \mathit{max}\,(\tau(a\downarrow, i) + \alpha_{a\downarrow c\downarrow}, \tau(b\downarrow, i) + \alpha_{b\downarrow c\downarrow}) \\
&= \rho(i+1),
\end{aligned}$$

where $i$ represents the cycle period index and $\rho$ is the cycle period. Substituting in the times for each transition in the timing simulation and solving for $\rho$, the cycle period of the C-element becomes

$$\rho = max(\alpha_{c\downarrow a\uparrow} + \alpha_{a\uparrow c\uparrow}, \alpha_{c\downarrow b\uparrow} + \alpha_{b\uparrow c\uparrow}) + max(\alpha_{c\uparrow a\downarrow} + \alpha_{a\downarrow c\downarrow}, \alpha_{c\uparrow b\downarrow} + \alpha_{b\downarrow c\downarrow}).$$

Note that $\rho$ in this simulation converges in a single cycle to the upper bound on the worst-case performance of the specification and is composed of the trigger signals used in the implementation of the circuit. Unfortunately, not all circuits are this simple and many require several cycles before they converge.

The natural extension to the timing simulation is to replace the maximum values with stochastic information showing the contribution of each possible delay in the cycle period.

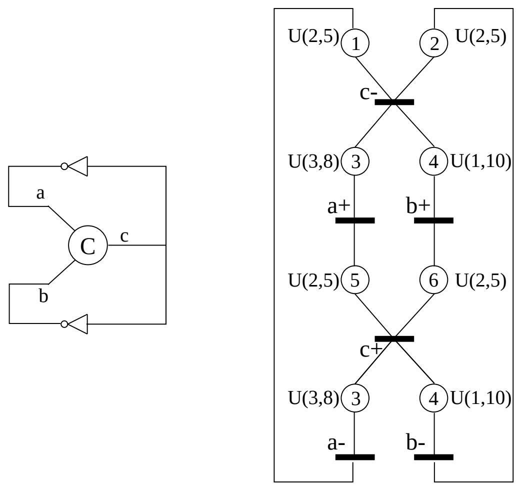

**Figure 5.3**. A simple two input C-element with its TSPN.

With this substitution, the cycle period of the C-element becomes

$$\rho = w_{a\uparrow c\uparrow}(\alpha_{c\downarrow a\uparrow} + \alpha_{a\uparrow c\uparrow}) + w_{b\uparrow c\uparrow}(\alpha_{c\downarrow b\uparrow} + \alpha_{b\uparrow c\uparrow}) + w_{a\downarrow c\downarrow}(\alpha_{c\uparrow a\downarrow} + \alpha_{a\downarrow c\downarrow}) + w_{b\downarrow c\downarrow}(\alpha_{c\uparrow b\downarrow} + \alpha_{b\downarrow c\downarrow}),$$

which is the stochastic cycle period of the circuit, a weighted sum of delays. Each weight in the stochastic cycle period denotes the contribution of the associated delay to the expected delay of a single cycle of the circuit.

Conflict is introduced into the stochastic cycle period calculation by altering what the performance metric means when conflict is present. As stated previously, the stochastic cycle period is a measure of the expected delay of a single cycle of any cyclic transition in the circuit. What is the cycle period of a circuit with conflicting transitionsΓ Since the cycle period of a particular signal is dependent on the arbitration of the conflict, its value may or may not converge to a steady state in a simulation ( i.e. the cycle period is not ergodic ). Looking at the sbuf-send-pkt2 circuit in Figure 5.4, the time between consecutive instances of done+/1 is dependent on the number of packets received as well as the number of times packets are rejected. The actual value of this delay cannot be enumerated without making the conflict resolution completely deterministic. Since preserving the stochastic model of the conflict is important, the delay dependence of the cycle period on the conflict resolution is addressed by altering the behavior of the choice points. This is done by making all choice points concurrent branches. How does this affect the cycle periodΓ The stochastic cycle period of the circuit is now the expected delay of the circuit as if all transitions are executed once, and those that are in conflict are executed concurrently. The stochastic cycle period can now be used to look at delays in cycles in the circuit created by conflict and weight their importance to the overall performance of the circuit by considering their frequency of occurrence using the conflict distributions provided in $\Upsilon$ in the *TSPN* model. This effectively preserves the stochastic model of the conflict representation. As an example, consider the following portion of a timing simulation for the sbuf_send_pkt2 circuit in Figure 5.4

$$\begin{aligned}
&\vdots \\
\tau(\mathit{sendline+/1}, i) &= w_{\mathit{req+/1}\ \mathit{sendline+/1}}(\alpha_{\mathit{req+/1}\ \mathit{sendline+/1}} + \tau(\mathit{req+/1}, i)) \\
\tau(\mathit{ackline+/1}, i) &= w_{\mathit{sendline+/1}\ \mathit{ackline+/1}} \cdot \\
&\quad (\alpha_{\mathit{sendline+/1},\ \mathit{ackline+/1}} + \tau(\mathit{sendline+/1}, i)) + \\
&\quad w_{\mathit{sendline+/2}\ \mathit{ackline+/1}} \cdot \\
&\quad (\alpha_{\mathit{sendline+/2},\ \mathit{ackline+/1}} + \tau(\mathit{sendline+/2}, i)) \\
\tau(\mathit{rejpkt+/2}, i) &= w_{\mathit{sendline+/1}\ \mathit{rejpkt+/2}} \cdot \\
&\quad (\alpha_{\mathit{sendline+/1},\ \mathit{rejpkt+/2}} + \tau(\mathit{sendline+/1}, i)) + \\
&\quad w_{\mathit{sendline+/2}\ \mathit{rejpkt+/2}} \cdot \\
&\quad (\alpha_{\mathit{sendline+/2},\ \mathit{rejpkt+/2}} + \tau(\mathit{sendline+/2}, i)) \\
&\vdots \\
\tau(\mathit{req+/1}) &= w_{\mathit{ack\text{-}/2}\ \mathit{req+/1}} \cdot \\
&\quad (\alpha_{\mathit{ack\text{-}/2},\ \mathit{req+/1}} + \tau(\mathit{ack\text{-}/2}, i)) + \\
&\quad w_{\mathit{ack\text{-}/1}\ \mathit{req+/1}} \cdot \\
&\quad (\alpha_{\mathit{ack\text{-}/1},\ \mathit{req+/1}} + \tau(\mathit{ack\text{-}/1}, i)) \\
&\vdots
\end{aligned}$$

In the partial simulation, the two concurrent transitions ackline+/1 and rejpkt+/2 are executed concurrently as if they had been concurrently enabled instead of enabled in conflict. Later in the simulation, when the time for req+/1 is computed, it is important to note that req+/1 is still triggered by either ack-/1 or ack-/2. Even though the two conflicting transitions are now concurrent, the location where the conflicting paths converge retains its original semantic structure. Therefore, req+/1 is allowed to fire when either ack+/1 or ack+/2 fire. The expected delay of req+/1 is correct due to the fact that

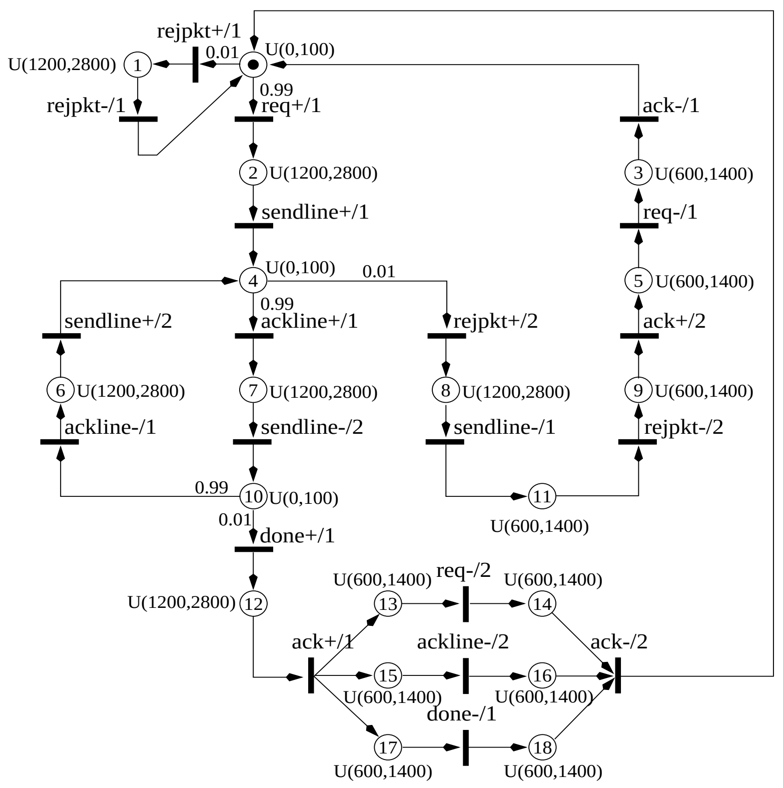

**Figure 5.4**. The TSPN for the sbuf-send-pkt2 circuit from the post office chip.

each possible delay value for req+/1 is weighted and the weights are calculated to reflect the probability of each possible delay when considering the original semantic meaning of the conflict where the two paths began.

The weights ($w_{uv}$) of each contributing member in the stochastic cycle period are found using the timing simulation and the function $\Theta$, which returns trigger probabilities. As the timing simulation proceeds, the time at which transitions occur is calculated by weighting the time of the enabling transition $t$ and delay of the enabled transition $t'$ when triggered by $t$, by the probability of $t$ being the trigger for $t'$. For the C-element in Figure 5.3, the time of the $i^{\text{th}}$ transition of $c\uparrow$ would now be calculated as

$$\tau(c\uparrow, i) = \Theta(a\uparrow, c\uparrow)(\tau(a\uparrow, i) + \alpha_{a\uparrow c\uparrow}) + \Theta(b\uparrow, c\uparrow)(\tau(b\uparrow, i) + \alpha_{b\uparrow c\uparrow}).$$

From this it is possible to derive $\rho$ and the weights $w_{uv}$ of each of the contributing members in the stochastic cycle period by selecting an arbitrary cyclic transition $x$ in the simulation and differencing the time of its $i^{\text{th}}$ firing by the time of its 1st firing and dividing by $i-1$ or

$$\rho = \lim_{i\to\infty} \frac{\tau(x,i)-\tau(x,1)}{i-1},$$

which is precisely the average cycle period for a conflict-free system where $i$ is the number of cycles. If conflict is present in the system, then it is the average cycle period of the system is all conflict points are considered as concurrent branches as state previously.

The timing simulation can now be applied to finding the stochastic cycle period by estimating the firing times of each transition in the cycle using the trigger probabilities and looking at the average weights generated by differencing two consecutive occurrences of an arbitrary cyclic transition. This process proceeds the weights converge to within a user specified tolerance of percent change.

To further aid the designer in quickly evaluating the performance of a design, another performance metric derived from the stochastic cycle period is provided by the system. The cycle metric is the expected value of the stochastic cycle period when the expected value of each of the contributing transitions is used for the alpha terms. Formally,

$$cm = \sum_{(u,v)\in T} w_{uv} E[\Delta(u \bullet \cap \bullet v)],$$

where $E[\Delta(u \bullet \cap \bullet v)]$ is the expected delay time from the place connecting $u$ and $v$. In a conflict-free system, the cycle-metric is the expected delay of a cycle as computed by the stochastic cycle period. When conflict is present, the cycle metric is the expected delay of the circuit if all choice points are converted to concurrent branches. To gauge the effectiveness of the stochastic cycle period and cycle metric in estimating the expected cycle time of a conflict-free system, a secondary simulation is used to estimate the limiting process for the exact average value of the cycle period. This simulation proceeds as described in Section 3.1 only the time of a single transition is recorded at each instance of its occurrence. In this way the limit is calculated by differencing the time of the $i^{\text{th}}$ and $(i+n)^{\text{th}}$ occurrence of the transition and dividing by $n-1$, where $n$ is the number of occurrences observed in a very long simulation. This simulation is not applicable to systems that have or can have conflicting transitions. Such a simulation has the potential of not converging and if it does converge, the reported delays will significantly differ from one simulation to another.

# CHAPTER 6

# CASE STUDIES

Three types of circuits are presented to show three applications of the stochastic cycle period $\rho$. First, handshaking protocols are examined and evaluated in passive active buffer control circuits. In this study, $\rho$ is shown to be able to correctly identify the better protocols. And second, two versions of a latch controller are examined to evaluate protocols and to choose optimal pin assignments. In this example the stochastic cycle period is not only used for finding good pin orderings but it is also used in manipulating timing assumptions in the specification to simplify the final circuit implementation. The last case study is an example of conflict and shows the effect of optimization on and off the highly probable path.

## 6.1 Passive Active Buffer

Burns in [5] evaluated every possible interleaving of a simple four phase handshaking protocol in a passive active buffer control unit using his cycle period. A fixed environment delay of 40 time units is used for all examples except those with isochronic forks (pab_a1,pab_pa, and pab_c8) which use an environment delay of 30 time units. All gates are assigned a delay of 10 time units. The timed versions of these circuits are given uniform delays with bounds that are $\pm 20\%$ of the fixed delays ($U(32,48)$,$U(24,36)$,$U(8,12)$). Table 6.1 shows the cycle metric computed from the stochastic cycle period compared to Burns' cycle period. The limit is the simulated actual value of the cycle period. The simulation column is the cycle metric for the stochastic cycle period found using stochastic simulation for $\Psi$ and $\Pi$ while the other columns represent the analytical methods of finding $\Psi$ and $\Pi$. To be able to correctly compare results, the delay from Burns' symbolic cycle period equations is computed using the single valued delay numbers and the resultant delay is then divided by the apparent number of unrollings used in finding his cycle period. In the table, pab_c4, pab_c3, and pab_b1 are shown to be better interleavings by all methods of deriving the stochastic cycle period and by Burns' cycle period. This shows the cycle metric from the stochastic cycle period to correctly track and correlate with Burns' results.

**Table 6.1**. Protocol analysis of passive active buffer controllers.

| Examples | limit | simulate | burst-mode | exponential | burns |
|---|---|---|---|---|---|
| pab_c4 | 105 | 99 | 99 | 99 | 100 |
| pab_c3 | 110 | 107 | 101 | 103 | 102 |
| pab_b1 | 109 | 109 | 102 | 104 | 102 |
| pab_pla | 120 | 120 | 107 | 116 | 105 |
| pab_pa | 120 | 120 | 120 | 120 | 120 |
| pab_c8 | 121 | 119 | 119 | 119 | 128 |
| pab_a1 | 122 | 120 | 120 | 120 | 128 |
| pab_b2 | 150 | 150 | 150 | 150 | 150 |
| pab_c5 | 153 | 150 | 150 | 150 | 160 |

## 6.2 Latch Controllers

In [7] several versions of a latch controller are presented. Timed versions of these circuits model all delays with uniform distributions which have time windows that are ±20% of the SPICE'ed delay values shown in [7] and the input delays are set so that ATACS produces the circuit shown in [7]. The stochastic cycle period analysis of the simple and enhanced latch controllers shows the enhanced controller to be 1.41 times faster than the simple controller, which correlates to the 1.47 times speedup shown in the SPICE results. As an interesting aside, the stochastic cycle period is the average delay of the circuit in a steady-state behavior, as opposed to a SPICE result, which is the cycle period as determined by the time at which inputs are specified to occur. The weights of the stochastic cycle period are shown on the *STG* in Figure 6.1. The weights are derived using the exponential approximation for transition probabilities and show the critical components of the cycle. The pin orderings for the circuit implementation in Figure 6.1 can be determined by examining the weights on the *STG*. According to the weights, the delay for the transition A+ is largely controlled by the trigger D+ and both D+ and A+ make significant contributions to the cycle period and therefore, D+ should be near the output of the gate implementing A+ to optimize its performance. For the falling transition A-, D- controls the delay and thus D- should be near the output of the gate. The transition E+ is triggered by Rin+ and A-, but Rin+ is not directly on the

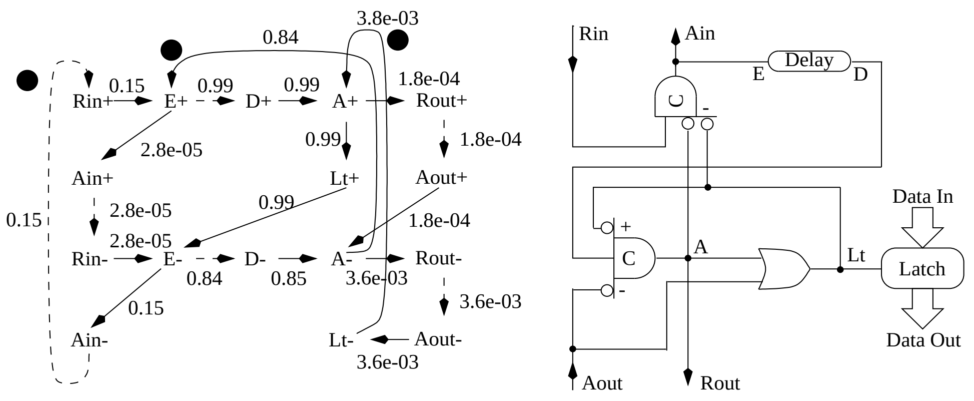

**Figure 6.1**. The STG and circuit implementation of an enhanced latch controller courtesy of [7].

critical path, so A- is moved near the output of the gate implementing E+. Accordingly, E- is triggered by Rin- and Lt+, but Rin- has negligible weight and is therefore not a contributor to the cycle period, so Lt+ should be moved near the output of the gate for E-. Finally, for the gate implementing Lt, the weights show Aout- to rarely contribute to the length of the cycle period, so A+ should be placed near the output of Lt to speedup the Lt+ transition.

The weights from the stochastic cycle period can also be used to identify trigger signals that do not contribute to the cycle period. In this example, transitions Rin- and Aout+ make negligible offerings to the delay of the cycle. If the bounded delays for Aout+ and Rin- are tightened, then it might be possible to remove them from the gates implementing A- and E- respectively. To verify this hypothesis, the timing assumptions for Aout+ and Rin- are modified to reflect a new upper bound that is lower than the previous upper bound by about 0.5%. The new circuit generated by `ATACS` no longer includes the two signals in the implementation of A- and E-. This shows how the stochastic cycle period and the trigger probabilities can be used to order pins and possibly restrict out triggers in timed circuits

To show the speed of the analytical methods on a larger example, three enhanced latch controllers are chained together in a pipeline. In this configuration, `ATACS` finds 453 states and using analytical methods is able to calculate the transition probabilities, find the steady-state distribution, calculate the trigger probabilities, and run the timing simulation in under 20 seconds, while the simulation of the transition probabilities and the steady-state distribution takes 38 minutes. The weights in the stochastic cycle period generated by the analytical methods correctly identify the critical path and mark the correct signals as contributing more delay to the cycle period than other signals. However, due to the limited amount of trace information used by the analytical methods in calculating the average transition probabilities, the ratio of the weights may significantly differ when compared to the ratio of weights generated with the simulated probabilities. Finally all methods estimated the cycle period delay to within 6% of the delay calculated by the simulated limit.

## 6.3 Conflict

As an example of the effect of conflict on the stochastic cycle period calculation, the choice points in the *sbuf-send-pkt2* circuit from the post office chip are annotated with a 99%/1% split as shown in Figure 6.2. The stochastic cycle metric for the circuit is reported as 4146.7 time units. If the ackline-/2 transition on the low probability path of the circuit is optimized to have a delay distribution equal to U(300,1000), then the value of cycle metric remains unchanged. This is expected since the contribution of ackline-/2 to the cycle period is insignificant. If, on the other hand, the sendline+/2 transition on the highly probable path is optimized to have a delay distribution equal to U(800,2000), then the resulting cycle period is 3553.47, which is a significant speedup in the cycle period of the circuit. This demonstrates the ability of the cycle period to respond to optimizations on the most probable paths in the circuit and correctly deal with user specified choice distributions. The only aspect not captured by the metric is the number of times inner loops a conflict locations are executed. This type of information cannot by captured by the stochastic cycle period.

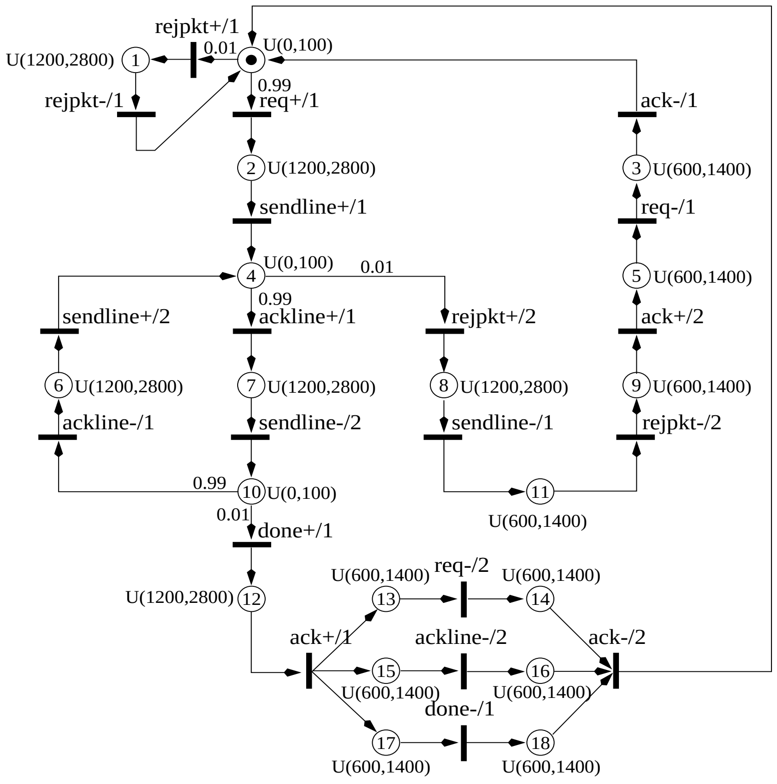

**Figure 6.2**. The TSPN for the sbuf-send-pkt2 circuit from the post office chip.

# CHAPTER 7

# CONCLUSIONS AND FUTURE WORK

This thesis presents the stochastic cycle period as a metric to analyze performance in timed asynchronous circuit. The method uses simulation and analytical techniques combined with stochastic information to define the components in the critical path of a specification along with their contribution to the overall delay of the circuit. The stochastic cycle period is derived using transition probabilities in the *TSPN* representation of the timed circuit, the steady-state distribution of the *RG* for the circuit, the trigger probabilities found using the long-term probabilities, and finally a timing simulation of the circuit. Several case studies have been described where the stochastic cycle period is used to identify better handshaking protocols and to improve final circuit implementations through pin reordering and gate selection. Future work includes improving the simulation method for finding transition probabilities and steady-state distributions by using regenerative methods [1]. While simulation is not the solution to trace dependence in the timed circuit, it does provide the most accurate estimates of the exact average values of the transition probabilities and deserves further study. Other work includes improving the burst-mode heuristic to consider a limited set of traces in non-burst-mode circuits. In this way, some marking history can be recouped in calculating transition probabilities. Finally, future work includes the automation of transistor sizing and adjustment of other low level parameters using the stochastic cycle period.

# APPENDIX

# CONFLICT SEMANTICS

All of the algorithms presented in this thesis are actually implemented to work on *timed event-rule (ER) structures*, [13] which can be automatically generated from some higher level language such as CSP[15] or VHDL [27]. Timed ER structures and Petri nets can represent an equivalent set of specifications, but ER structures have a somewhat more concise representation [23].

A timed ER structure S can be represented with the tuple $\langle A, E, R, \# \rangle$ where:

1. $A$ *is the set of atomic actions*;
2. $E \subseteq A \times (N = \{0, 1, 2...\})$ *is the set of events*;
3. $R \subseteq E \times E \times N \times (N \cup \{\infty\})$ *is the set of rules*;
4. $\# \subseteq E \times E$ *is the conflict relation.*

The set $A$ contains the atomic actions possible in the system. The occurrence of an action is an *event* and is denoted $(a, i)$ where $a$ is the action and $i$ is an *occurrence index* for the action. The rule set $R$ represents a causal dependence between events. Each rule, of the form $\langle e, f, l, u \rangle$ is composed of an *enabling event* $e$, an *enabled event* $f$, and a *bounded timing constraint* $\langle l, u \rangle$. A rule states that the enabled event cannot occur until the enabling event has occurred. Ignoring conflict for the moment, if $n$ rules enable the same event, then that event cannot occur until *all* $n$ enabling events have occurred. A rule is *enabled* if its enabling event has occurred. The timing constraint places a lower and upper bound on the timing of a rule. A rule is *satisfied* if the amount of time which has passed since the enabling event has exceeded the lower bound of the rule. A rule is said to be *expired* if the amount of time which has passed since the enabling event has exceeded the upper bound of the rule. Again ignoring conflict, an event cannot occur until *all* rules enabling it are satisfied. An event must always occur before *every* rule enabling it has expired. Since an event may be enabled by multiple rules, it is possible

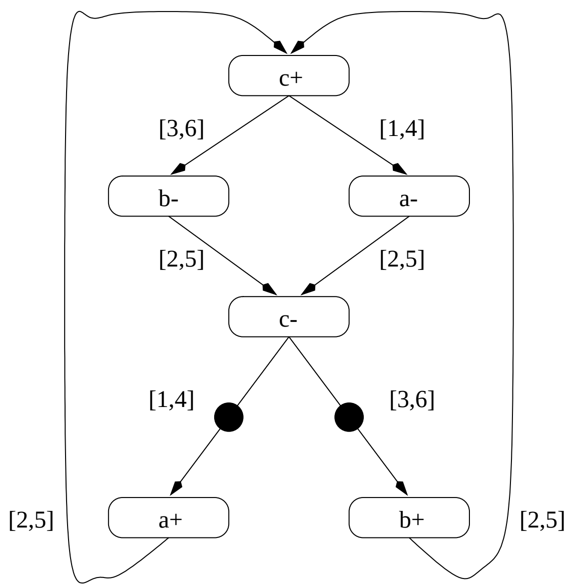

**Figure A.1**. The timed ER structure for a simple c-element.

that the difference in time between the enabled event and some enabling events exceed the upper bound of their timing constraints, but not for all enabling events.

The conflict relation is added to model *disjunctive* behavior and choice. When two events $e$ and $e'$ are in conflict, (denoted $e\#e'$), this specifies that either $e$ can occur or $e'$ can occur, but not both. Taking the conflict relation into account, if two rules have the same enabled event and conflicting enabling events, then only one of the two mutually exclusive enabling events needs to occur to cause the enabled event. This models a form of disjunctive causality. *Inherently disjunctive* behavior, or true OR causality, cannot currently be modeled, but we are working on techniques to address this. Choice is modeled when two rules have the same enabling event and conflicting enabled events. In this case, only one of the enabled events can occur.

Figure A.1 shows an example of a timed ER structure for a simple two input c-element. Events are the vertices and the rules that relate them are the edges. A '#' below a vertex indicates that the events enabled by this vertex are in conflict. Tokens can be used to indicate that a rule's enabling event has fired. Each rule has a delay range associated with it as shown in the figure.

In most instances, the semantics of the timed ER structure closely resemble those of the *TSPN*. There is, however, a specific construct in the two representations which does not preserve semantic equality amongst the two paradigms. This notable construct is the free and extended free choice structures in the *TSPN*. Free and extended free choice, as presented in Section 2.1, cannot be exactly represented in the timed ER structure. An example of this nuance is shown in Figure A.2, which shows the standard conflict model used in the timed ER structure. In this examples, the '#' symbol denotes that events $a+$ and $b+$ are in conflict with each other. The system should only allow one of the two events to fire, not both. As shown, the timed ER structure places timing constraints on edges between nodes. The edges are referred to as rules and the nodes as events. Rules retire in the same manner as places becoming available to transitions in the *TSPN*. When enough rules have retired sufficient to enable an event, then the event instantly fires. Since each rule has a timing constraint, a true conflict construct where two events are enabled to occur at the same instance is not easily created in the timed ER structure ideology. All choice resolution is resolved through timing constraints, meaning that in a system simulation, the system chooses a time for each rule to retire and the first event to have a sufficient number of rules retired to enable it to fire, fires. Since the timed ER structure uses an interleaving semantics equivalent to the *TSPN*, the probability of two events becoming enabled to fire at the exact same instance is assumed to be zero, unless a singular distribution is used. If singular distributions are used on the rules enabling the conflicting events, then the delay for either of the two events from enabling to firing becomes fixed, because both rules must be singular on the same point to force a conflict, thus omitting the stochastic model of the firing delay times.

In contrast, conflict in the *TSPN* world can only exist at a free or extended free choice construct in the net, as shown in Figure A.2. Conflict at a choice construct in a *TSPN* is resolved through a user supplied distribution as discussed in Section 2.1. In a simulation of a *TSPN* when the choice place receives a token, the system obtains a time for that token to become available to transitions from its respective distribution. When that token becomes available, it enables two transition to fire at the same instance. Since only one token is available, only one transition can fire. The system chooses which transition to fire according to the user supplied distribution. This model is distinctly different than that of the timed ER. The timed ER structure samples a time for each rule to resolve conflict where the *TSPN* selects a time for the common place and uses a choice distribution to resolve conflict. This semantic difference forces the introduction of a specialized construct

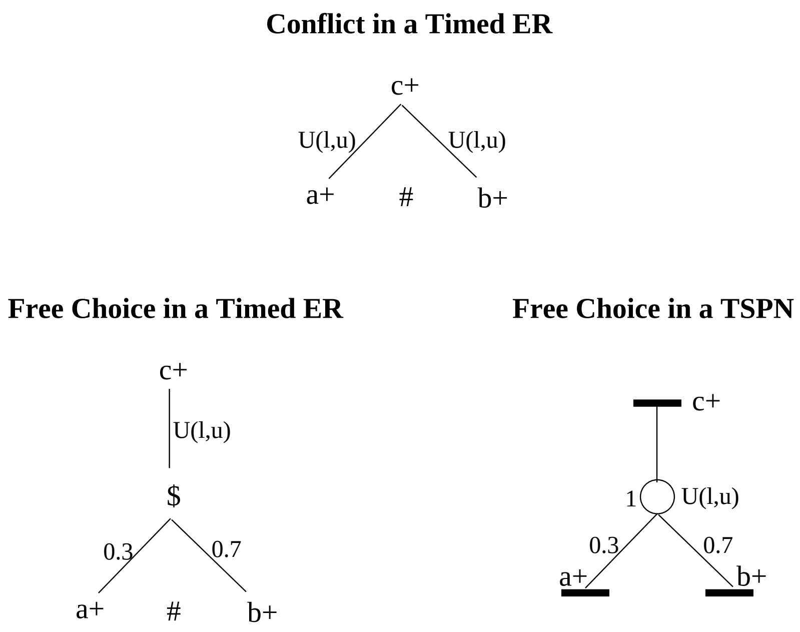

**Figure A.2**. Free choice structure in a *TSPN* and timed ER structure.

in the timed ER structure to preserve semantic equivalence to the *TSPN*.

Figure A.2 shows a free choice construct in a timed ER structure that is semantically equivalent to the free choice construct in a *TSPN*. When the event *c+* fires, the *$* event is scheduled to fire at a time determined by its respective distribution. When the *$* event fires, it enables two events that, by their respective distributions (singularities at point 0), must fire instantly. This forces a conflict in the system between the events *a+* and *b+*, which cannot be resolved through timing consideration. Therefore a user supplied distribution is used to choose which event to fire. In this example, the *$* event is a system event that is used only to mark that enough time has transpired to simultaneously enable events *a+* and *b+* and is analogous to a place becoming available to transitions in the *TSPN* model. In this way choice semantics are preserved in the *TSPN* and timed ER structure representations and the stochastic delay model is retained in the timed ER structure conflict representation.

# REFERENCES

[1] Aiguo Xie, S. K., and Beerel, P. A. Bounding average time separations of events in stochastic timed petri nets with choice. forthcoming paper.

[2] Beerel, P. A., Yun, K. Y., and Chou, W. C. Optimizing average-case delay in technology mapping of burst-mode circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems* (Mar. 1996), IEEE Computer Society Press.

[3] Belluomini, W., and Myers, C. Verification of timed systems using posets. In *International Conference on Computer Aided Verification* (1998), Springer-Verlag.

[4] Belluomini, W., and Myers, C. J. Efficient timing analysis algorithms for timed state space exploration. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems* (Apr. 1997), IEEE Computer Society Press, pp. 88–100.

[5] Burns, S. M. *Performance Analysis and Optimization of Asynchronous Circuits.* PhD thesis, California Institute of Technology, 1991.

[6] Davis, A., Coates, B., and Stevens, K. The Post Office experience: Designing a large asynchronous chip. In *Proc. Hawaii International Conf. System Sciences* (Jan. 1993), vol. I, IEEE Computer Society Press, pp. 409–418.

[7] Furber, S. B., and Liu, J. Dynamic logic in four-phase micropipelines. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems* (Mar. 1996), IEEE Computer Society Press.

[8] Hauck, S. Asynchronous design methodologies: An overview. Tech. Rep. TR 93-05-07, Department of Computer Science and Engineering, University of Washington, Seattle, 1993.

[9] Krieger, C. D., and Myers, C. J. Performance driven complete state coding of timed circuits. forthcoming paper.

[10] M. Ajmone Marsan, G. Balbo G. Conte, S. D., and Francheschinis, G. *Modelling With Generalized Stochastic Petri Nets.* John Wiley & Sons Ltd., Baffins Lane, Chichestester, West Sussex PO19 1UD, England, 1996.

[11] Merlin, P., and Faber, D. J. Recoverability of communication protocols. *IEEE Transactions on Communications 24*, 9 (1976).

[12] Myers, C. J. *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits.* PhD thesis, Dept. of Elec. Eng., Stanford University, Oct. 1995.

[13] Myers, C. J. *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits.* PhD thesis, Stanford University, 1995.

[14] Myers, C. J., and Meng, T. H.-Y. Synthesis of timed asynchronous circuits. *IEEE Transactions on VLSI Systems 1*, 2 (June 1993), 106–119.

[15] Myers, C. J., and Meng, T. H.-Y. Synthesis of timed asynchronous circuits. *IEEE Transactions on VLSI Systems 1*, 2 (June 1993), 106–119.

[16] Myers, C. J., Rokicki, T. G., and Meng, T. H.-Y. Automatic synthesis of gate-level timed circuits with choice. In *Advanced Research in VLSI* (1995), IEEE Computer Society Press, pp. 42–58.

[17] Nowick, S. M., and Dill, D. L. Synthesis of asynchronous state machines using a local clock. In *Proc. International Conf. Computer Design (ICCD)* (Oct. 1991), IEEE Computer Society Press, pp. 192–197.

[18] Rubinstein, R. Y. *Simulation and the Monte Carlo Method.* John Wiley & Sons, Baffins Lane, Chichestester, West Sussex PO19 1UD, England, 1981.

[19] Sedgewick, R. *Algorithms in C++.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1992.

[20] Stewart, W. J. *Introduction to the Numerical Solution of Markov Chains.* Priceton University Press, 41 William Street, Princeton, NJ, 08540, 1994.

[21] Subrahmanyam, P. What's in a timing disciplineΓ considerations in the specification and synthesis of systems with interacting asynchronous and synchronous components. In *Hardware Specification, Verification and Synthesis: Mathematical Aspects* (1990), Springer-Verlag.

[22] Thacker, R. A. Implicit methods for timed circuit synthesis. Master's thesis, University of Utah, 1998.

[23] Winskel, G. An introduction to event structures. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. Noordwijkerhout, Norway* (June 1988).

[24] Xie, A., and Beerel, P. A. Symbolic techniques for performance analysis of timed systems based on average time separation of events. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems* (Apr. 1997), IEEE Computer Society Press, pp. 64–75.

[25] Xie, A., and Beerel, P. A. Accelerating Markovian analysis of asynchronous systems using string-based state compression. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems* (1998), pp. 247–260.

[26] Zheng, H. Specification and compilation of timed systems. Master's thesis, University of Utah, 1998.

[27] Zheng, H., and Myers, C. J. Specification and compilation of mixed-timed

systems using vhdl. forthcoming paper.