# Probabilistic Timing Verification and Timing Analysis for Synthesis of Digital Interface Controllers

by
Marco Antonio Escalante
B.Sc., Universidad Iberoamericana, 1987
M.A.Sc., University of Victoria, 1991

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of
Electrical and Computer Engineering

We accept this thesis as conforming
to the required standard

Supervisor: Dr. N. J. Dimopoulos

## ABSTRACT

In this dissertation we present two techniques on the topic of digital interface design: a probabilistic timing verification and a timing analysis for synthesis, both rooted in a formal specification. Interface design arises when two digital components (e.g., a processor and a memory device) are to be interconnected to build up a system. We have extended a Petri net specification to describe the temporal behavior of the interface protocols of digital components. The specification describes circuit delays as random variables thus making it suitable to model process variations and timing correlation. Interface probabilistic timing verification checks that a subsystem, composed of components to be interconnected and the associated interface logic, satisfies the timing constraints specified by the components' specifications. Our verification technique not only yields tighter results than previous techniques that do not take timing correlation into consideration but also, if the timing constraint is not satisfied, determines the probability that a constraint will be violated. The second technique, timing analysis for synthesis, finds tight bounds on the delays of the interface logic, which are unknown prior to synthesis, such that all the timing constraints given in the component specifications are satisfied.

Examiners:

Dr. Nikitas J. Dimopoulos, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Kin F. Li, Departmental Member
(Department of Electrical and Computer Engineering)

Dr. Fayez El-Guibaly, Departmental Member
(Department of Electrical and Computer Engineering)

Dr. D. Michael Miller, Outside Member
(Department of Computer Science)

Dr. Robert D. McLeod, External Examiner
(Department of Electrical and Computer Engineering,
University of Manitoba)

# Table of Contents

# List of Figures

# List of Tables

# Acknowledgments

The author of this dissertation would like to thank to all the people that contributed, either academically or otherwise, to the successful completion of this thesis, with special consideration to the following people: My thesis advisor, Dr. Nikitas Dimopoulos, for his guidance in asking interesting questions, for the freedom he gave me to pursue my own directions, and for his thoughtful suggestions that always improved my ideas. The examiners for their versed comments on this dissertation: Dr. Kin Li, who is one of the founder members of DAME, the UVic's project that motivated the techniques developed in this dissertation; Dr. Fayez El-Guibaly, who shared his enthusiasm about affine sets and linear algebra with me in a memorable trip to Banff; Dr. Michael Miller, who gave crucial encouragement to this research since the early phases and brought about my first publication in the Canadian Conference on VLSI, the begininning of a fruitful participation in other conferences; and Dr. Robert McLeod, who kindly accepted to be the external examiner and whose suggestions have improved significantly the accuracy of the contents of this dissertation. A very special mention goes to Dr. Luciano Lavagno, who patiently read my manuscripts and always provided me with his friendly feedback and with invaluable pointers that opened up new avenues. Mr. Allan Silburt, who gave me the opportunity to work with his group at the Bell-Northern Labs (currently Nortel), a wonderful experience that was enriched by an exchange of ideas with Prof. Eduard Cerny and Dr. Karim Khordoc. Dr. Komei Fukuda, who graciously made his work and code on polytopes available to me. Dr. Mantis Cheng, who introduced me to the fascinating world of process algebras. The ECE team of secretaries, Maureen Denning, Lynne Barrett, and Vicky Smith, for making my days at UVic so enjoyable. And finally my parents whose immense love initiated this wonderful experience, and my wife Dongni Li whose indefatigable support brought it to a successful end.

*Amo al tzentzontle, pájaro de cuatrocientas voces,*
*amo al color de jade, y al enervante perfume de las flores,*
*pero amo más a mi hermano el hombre.*

Nezahualcóyotl

# Notation

| | |
|---|---|
| $\Psi = \langle \Sigma, C_N \rangle$ | interface specification |
| $\Omega = \langle T_\Omega, P_\Omega, M_{\Omega 0}, \Gamma_\Omega, ct_\Omega, Y, \lambda_\Omega \rangle$ | AOC di-graph |
| $\Omega_U = \langle T_{U\Omega}, P_{U\Omega}, \Gamma_{U\Omega}, ct_{U\Omega}, Y, \lambda_{U\Omega} \rangle$ | unfolding of AOC di-graph $\Omega$ |
| $\Sigma = \langle N, Y, \lambda \rangle$ | timed STG |
| $C_N = \{c_{ij}\}$ | set of constraint rules |
| $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon \rangle$ | constraint rule associated with net $N$ |
| $N = \langle P, T, F, M_0, \Gamma \rangle$ | probabilistic timed Petri net |
| $P$ | set of places |
| $T$ | set of transitions |
| $F \subseteq (P \times T) \cup (T \times P)$ | flow relation |
| $M : P \rightarrow \aleph$ | marking function |
| $M_0$ | initial marking |
| $\Gamma: P \rightarrow \tau$ | time labeling function |
| $\aleph$ | set of non-negative integers |
| $\lambda: T \rightarrow A(Y) \cup \{\varepsilon\}$ | signal transition labeling function |
| $Y$ | set of ports |
| $\bullet t = \{p \in P: (p, t) \in F\}$ | preset of transition $t$ |
| $t\bullet = \{p \in P: (t, p) \in F\}$ | postset of transition $t$ |
| pdf | probability density function |
| $f_{\tau 1 \ldots \tau M}(\tau_1, \ldots, \tau_M) =$ | joint pdf of random variables $\tau_1 \ldots \tau_M$ |
| $\tau$ | time (random) variable |
| $\tau_{t_i}$ | firing time of transition $t_i$ |

| | |
|---|---|
| $t_{i(k)}$ | $k$-th occurrence of transition $t_i$ |
| $\pi_{ij} = \langle t_i, t_j, p_{ij}, \tau_{ij}\rangle$ | timing parameter |
| $\phi_{ij} = \langle t_i, t_j, \rho_{ij}\rangle$ | correlation rule |
| $S = \{s_{ij}\}$ | semantic specification |
| $s_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon\rangle$ | set of semantic rules |
| $\Psi_c = \langle \Sigma_c, C_{Nc}\rangle$ | complete graph |

# Chapter 1

# Introduction

## 1.1 Outline

This dissertation presents results on the topics of timing verification and timing analysis for the synthesis of digital interface circuits. This introductory chapter aims to show the driving ideas that motivated our work and the main contributions in a rather informal fashion. Other chapters will deal with the task of explaining in more detail the framework and techniques that support the results outlined in this chapter.

We have developed a formal framework that can be the basis for the a better modeling of the timing aspects that play an important role during the synthesis of high-performance hardware systems. This will lead, we believe, to the creation of computer-aided design (CAD) tools that will relieve hardware designers from time-consuming, error-prone tasks, thus allowing them to focus on more creative steps of the design process. This is important in view of the fact that there is a clear trend towards very large and complex hardware designs, either general-purpose or application-specific chips, while there is constant pressure to reduce the time to market. A viable solution is to increase the design abstraction, and our effort is in that direction.

In the following section we present some basic ideas behind the design and synthesis of hardware interface logic.

## 1.2 Hardware interface synthesis

Increasingly more complex hardware systems are designed every day that must outperform (in speed, power consumption, etc.) the previous generation of hardware devices. This force creates new challenges to the hardware design flow. An attractive design option is to construct systems using already developed and tested modules. Such modules can be as simple as macrocells, or as complex as microprocessors. An important problem of this approach is system integration, that is interconnecting the off-the-shelf components to achieve the desired functionality. Integration of the modules may require designing interfacing logic which allows the modules to transfer information.

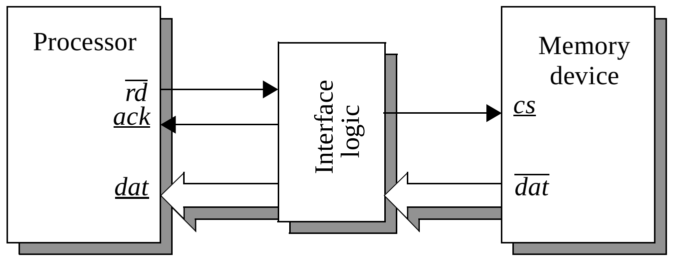


Figure 1.2.1 Data transfer read interface example.

Let us present a simple example to give a glimpse of the interface design problem. We present some terms rather informally but later in Chapters 2 and 3 we shall discuss them more thoroughly. Figure 1.2.1 shows a system composed of two components: a processor and a memory device. A typical *operation* between the two components is called data transfer, according to which the processor can *read*, or *write*, data from, or to, the memory device. To accomplish this, each component has external lines called *ports* which carry *signals,* which without loss of generality we assume to be electrical in nature. Ports can accept signals (called input ports), or emit signals (called output ports), or both (called bidirectional port). A binary signal can have two values, usually called high and low respectively. (Of course an electrical signal actually has a continuous value, either voltage or current, but a binary signal is a convenient abstraction in a digital system).

Figure 1.2.1 shows the ports involved in the read operation. On the processor side, the $\overline{rd}$ and $\underline{ack}$ ports are used to produce a sequence of events, or *signal transitions*, to tell the memory device when the processor expects to have the data ready (we use overlining/ underlining of the names of ports to identify them as output/input ports). For example by setting $\overline{rd}$ to high, the processor indicates that it wants to request a piece of data from the memory, and it will keep $\overline{rd}$ high until it detects a high in $\underline{ack}$; when the memory detects a high in $\underline{cs}$, it places a piece of data in $\overline{dat}$. A sequence of signal transitions that a component uses to exercise an interface operation is called a *protocol*.

Both components are available in different flavors from different manufacturers. Thus it is likely that the components use different protocols to exchange information. In our simple example the memory device does not have a port to tell the processor that a piece of data is available for reading. For example the interface circuit shown in Figure 1.2.1 must generate a signal to be fed in into $\underline{ack}$; this is called protocol conversion.

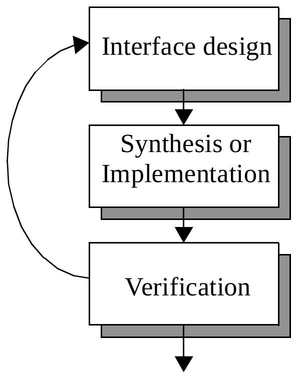


Figure 1.2.2 Interface synthesis task.

Figure 1.2.2 shows the typical steps in the interface synthesis task. This design task occurs during the integration phase of system design once the modules that comprise the system have been chosen, interface circuits must be designed to achieve inter-module communication. The result of the interface synthesis task is a complete implementation of the system. The system implementation is then checked to meet design constraints, either by the use of extensive simulation or by the application of formal verification techniques. If no violations are found, the process successfully terminates, otherwise some steps are repeated.

An important contribution of this dissertation is to offer an alternative strategy to the above iterative process. We suggest that before interface synthesis, a timing analysis for synthesis (TAFS) be performed on the interface design which determines tight bounds on the interface delays. After such analysis it is possible to decide on the feasibility of the design (if the design is implementable) and if that is the case, time-driven synthesis techniques can be used to complete the implementation. The main problem is that the delays of the interface circuitry are not known.

In the following section we shall discuss where the techniques developed in this dissertation fit in this picture.

## 1.3 Main contributions of this dissertation

The general direction of our work is to address the timing aspects particular to the interface synthesis task. In particular, we propose a formal framework suitable for the specification of systems composed of components and interface circuits, and two techniques to analyze and verify timing properties of such systems.

As mentioned at the beginning of the chapter, timing plays an important role during interface synthesis, and thus timing verification techniques, which can prove that the system timing behavior is correct, promise to be effective tools to facilitate the design process. As a matter of fact, interface timing verification research has attracted considerable attention recently [16, 17, 20, 48, 49, 67, 72, 114].

It is our tenet that in order to verify a hardware interface between two modules, one does not need to know all the details of the implementation of the modules. What is needed is the specification of each module's interface behavior. This specification is usually given in textual form describing the sequence of events that define the protocol, accompanied with timing diagrams that show explicitly the temporal relationships between the protocol events. One of our goals is to establish a formal specification adequate for describing interface behaviors of hardware modules. In the literature, various approaches have been proposed for describing hardware: modal logics [15, 35, 88, 94], process algebras [92, 63, 86, 72], and nets [115, 29, 79, 124, 114, 93, 66]. Our proposed representation is an extension and generalization of signal transition graphs [115, 29], which belongs to the net approach.

Once the formal specification framework was set, we developed two techniques aimed at supporting the interface synthesis task. Both techniques are rooted in formal verification which, in contrast to simulation, tries to determine that a system satisfies certain timing properties (i.e., timing constraints) under all circumstances.

The first technique, interface timing verification, is able to verify that a subsystem, comprising two components to be interconnected and the associated interface logic, satisfies the timing constraints specified by the components' interfaces. In this dissertation we present a novel probabilistic model which not only yields tighter results than previous models that do not take timing correlation into consideration but also provides more information to the designer by returning qualitative and quantitative information about the probability that a constraint will be violated rather than just a fail/pass result as is the case with traditional interval-based timing verification techniques.

The second technique, timing analysis for synthesis, is a powerful tool during synthesis because it treats the interface as a module to be designed, whose timing parameters are unknown, and finds the delay boundaries that the interface timing parameters must satisfy to comply with the timing constraints given in the components' specifications. If the solution space is empty, the interface design is infeasible. Otherwise, bounds can be known about the interface delays that can be used advantageously during synthesis. The difference of this preliminary analysis from formal verification is that actual temporal information about the interface is not completely known in advance of synthesis.

## 1.4 Dissertation outline

In this chapter we have introduced informally the motivation and goals of this dissertation. We address the timing aspects of the interface synthesis task that must be carried out during the construction of modular systems. A fundamental problem in interface synthesis is to verify that an interface implementation satisfies the timing constraints imposed by the components that the interface interconnects. High-performance systems and sub-micron technologies are pushing the timing of system modules and silicon to the limit. It is of paramount importance for CAD tools to support verification techniques that help hardware designers in coping with shorter times to market new products.

In Chapters 2 and 3, we develop a suitable formal representation framework that makes explicit the various timing relationships that are present in the module protocols. In Chapter 4, we formulate the timing verification problem as a constraint satisfaction problem that determines if a set of timing constraints are satisfied and, if that is not the case, it produces a probability distribution that a constraint will be violated, which can be used to assess the reliability of the system. Finally in Chapter 5, we present a technique called timing analysis for synthesis which allows designers to assess the feasibility of an interface design prior to synthesis.

# Chapter 2

# Representation of Interface Specifications

## 2.1 Introduction

In this dissertation we aim to study temporal properties of interface logic. As we mentioned earlier, hardware systems can be constructed using readily available building blocks, which we call system components, such as processors, memories and I/O devices. Interface logic has the important function of providing the necessary paths to facilitate the transfer of information between components. As we shall discuss in Chapter 3, a component expects certain events, whose partial ordering is defined by a protocol, for proper operation.

In this chapter, we present a formal model that we use to represent component protocols and component interconnection. Two of the main features of our formalism are: that it represents distinctly the two different timing information present in timing diagrams, propagation delays, and timing constraints; and that it can handle correlation information that is present in timing diagrams.

## 2.2 Petri net model

### 2.2.1 Petri nets

Petri nets are widely used to model concurrent systems because they have simple and intuitive semantics.

A Petri net [107] is a tuple $N = \langle P, T, F \rangle$, where $P$ is a non-empty set of places, $T$ is a non-empty set of transitions, and $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation. The marking of a Petri net is a function $M : P \rightarrow \aleph$ that assigns to each place of the net a (non-negative) number of *tokens* ($\aleph$ is the set of non-negative integers). A marked Petri net is a tuple $N = \langle P, T, F, M_0 \rangle$, where $M_0$ is the initial marking. The state of a Petri net can be described by its marking.

A Petri net is usually represented as a directed bi-partite graph with transition nodes (bars) and place nodes (circles) and links from transitions to places and from places to transitions as defined by the flow relation (refer to Figure 2.2.1a).

For any transition $t \in T$, the set of all its incoming places is denoted as $\bullet t = \{p \in P: (p, t) \in F\}$. Likewise, the set of all its outgoing places is written as $t\bullet = \{p \in P: (t, p) \in F\}$. Analogous definitions exist for the set of incoming transitions and outgoing transitions of a place $p \in P$, denoted $\bullet p$ and $p\bullet$ respectively. The number of tokens assigned to a place $p$ by a marking $M$ is written as $M(p)$.

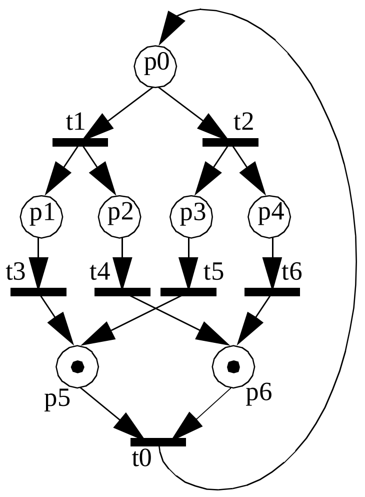
The *firing rule* determines the dynamics of a Petri net, *i.e.*, how the tokens are propagated through the net. A transition $t \in T$ is enabled at a marking $M$ *iff* $M(\bullet t) \geq 1$. Every enabled transition may *fire*. The effect of the firing of a transition is as follows: After a transition fires, a new marking $M'$ is obtained from $M$ as follows: $M' = M - \bullet t + t\bullet$.

The firing of an enabled transition $t$ in marking $M$ is written $M \xrightarrow{t} M'$ where $M'$ is the new marking after firing $t$. The pair $(t, M')$ is called an immediate $t$-derivative of $M$. In general $M'$ is an $(t \dots \mathrm{v})$-derivative (or just derivative) of $M$ if $M \xrightarrow{t} \dots \xrightarrow{v} M'$. The double sequence $ES = \{(M_{(0)}, \dots, M_{(j)}), (t_{(1)}, \dots, t_{(j)})\}$ is called an execution sequence if for all $i = 1, \dots, j$, $M_{(i\text{-}1)} \xrightarrow{t_{(i)}} M_{(i)}$. The set of all execution sequences starting from $M_0$ is denoted by $S'(M_0)$. Note that the sequence of transitions and the first marking uniquely determine the sequence of markings. A marking $M'$ is said to be reachable from $M$ if and only if there exists an execution sequence $ES$ in which, for some $i<j$, $M = M_{(i)}$ and $M' = M_{(j)}$.

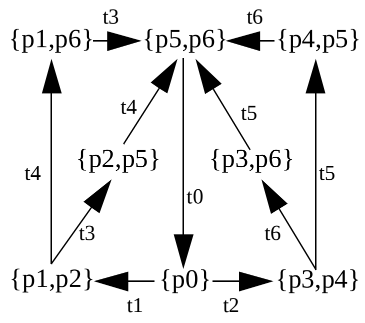
A labelled transition system is the triple $\langle S, T, \{\xrightarrow{t}, t \in T\}\rangle$, where $S$ is a set of states, $T$ is a set of transition labels, and $\xrightarrow{t} \subseteq S \times S$ is a transition relation for each $t \in T$. We define the meaning of a Petri net in terms of the labelled transition system $\langle SM_o, T, \{\xrightarrow{t}, t \in T\}\rangle$ where $SM_o$ is the set of reachable markings from $M_o$.

A derivation tree of the initial marking $M_o$ is a tree which collects all the derivatives of $M_o$. The nodes of the tree are reachable markings from $M_o$. An edge of the tree joining $M$ and $M'$ is labelled with the firing action $t$ if $M \xrightarrow{t} M'$. Derivation trees are usually infinite. A reachability graph is drawn from a derivation tree by collapsing identical markings, which have the same immediate derivatives, into a single node. Figure 2.2.1b shows the reachability graph of the Petri net of Figure 2.2.1a.

A Petri net marking is live if for each $M \in SM_o$ and for each transition $t$ there exists a marking $M' \in SM$ that enables $t$. A marked Petri net is live if its initial marking is live. A marked Petri net is $k$-bounded (or simply bounded) if there exists an integer $k$ such that for each place $p$, for each reachable marking $M$, $M(p) \leq k$. A marked Petri net is safe if it is 1-bounded.



(a)



(b)

Figure 2.2.1   (a) Petri net, and (b) its reachability graph.

A transition $t_1$ disables another transition $t_2$ at a marking $M \in SM_o$ if both $t_1$ and $t_2$ are enabled at $M$ and $t_2$ is not enabled in any $M' \in SM$. A marked Petri net is persistent if no transition can ever be disabled at any reachable marking.

Two transitions $t_1$ and $t_2$ in a marked Petri net are concurrent if there exists a reachable marking $M \in SM_o$ where both $t_1$ and $t_2$ are enabled and neither $t_1$ disables $t_2$ nor viceversa. Two transitions $t_1$ and $t_2$ of a marked Petri net are in direct conflict if there exists a reachable marking $M \in SM_o$ where both $t_1$ and $t_2$ are enabled and either $t_1$ disables $t_2$ or viceversa (or both).

A Petri net is a marked graph if for every place $p \in P$, $|\bullet p| = 1$and $|p\bullet| = 1$. A marked graph is persistent for every initial marking $M_o$. Furthermore every strongly connected marked graph has at least one live and safe initial marking [96].

A Petri net is a state machine if for every transition $t \in T$, $|\bullet t| = 1$and $|t\bullet| = 1$. Every strongly connected state machine has at least one live and safe initial marking. The Petri net subclass of state machines is isomorphic to classical Finite State Machines if we label each transition of the state machine with an input/output state pair and we interpret each place as an internal state.

A choice place is a place for which $|p\bullet| > 1$. A choice place is said to be unique choice if at most one of the successor transitions $|p\bullet|$ ever becomes enabled. A Petri net is free-choice if for any two transitions $t_1$ and $t_2$ that share a predecessor place, both $t_1$ and $t_2$ have only one predecessor. A Petri net is extended free-choice if any two transitions that share one or more predecessor places have exactly the same set of predecessor places.

Classic Petri nets as discussed in this section do not have an explicit mechanism to account for time. Time is of paramount importance in our application. In the following section we survey some extensions of Petri nets that model time explicitly.

### 2.2.2 Time extensions of Petri nets

From Section 2.2.1 it is clear that classic Petri nets cannot model particular time values, which is of paramount importance for performance evaluation and timing verification. There exist in the literature different flavors of time extensions to Petri nets that overcome that problem. In the following, we survey time extensions of Petri nets that have been proposed in the literature that we consider relevant to our work.

Ramchandani [111] associates an execution time $r$ whose domain is the real numbers, with each transition of the Petri net. Ramchandani's time-extended Petri nets are called Timed Petri nets. A transition is enabled according to the classic Petri net's firing rule. When a transition initiates its execution, it immediately consumes tokens in the set $\bullet t$ of its input places. The transition takes $r$ units of time to complete its execution before sending tokens to its output places $t\bullet$. Thus Ramchandani Timed Petri nets are deterministic.

Merlin [90, 91] increased the expressiveness of Ramchandani's Timed Petri nets in two ways. Firstly he assigned a compact non-negative non-empty interval [$d$, $D$] to each transition of the Petri net. A transition can fire only if it has been enabled for $d$ time units, and it must fire if it has been enabled for $D$ time units. Secondly Merlin modified the firing rule as follows: the tokens in the input places of an enabled transition $t$ that fires are removed from $\bullet t$ when $t$ fires. Merlin's time-extended Petri nets are called Time Petri nets. In Merlin's Time Petri nets, two or more transitions can be enabled by a common set of tokens such that when one transition fires, it disables the firing of the others. Recall that in Ramchandani's Timed Petri nets, the tokens in the input places of an enabled transition are committed when the transition starts execution.

A timed execution of a time-extended Petri net from the initial marking $M_0$ is an execution sequence $ES$ of $S'(M_0)$ augmented with a non-decreasing sequence of real non-negative values representing the instants of firing of each transition such that consecutive transitions $\{t_i, t_{i+1}\}$ correspond to ordered firing times (or epochs) $\tau_i \leq \tau_{i+1}$. The interval $[\tau_i, \tau_{i+1})$ between consecutive epochs represents the period in which the net remains in marking $M_i$, where $\tau_0 = 0$.

Berthomieu and Diaz [11] used an enumerative analysis technique related to the reachability analysis method for classic Petri nets to analyze the timed behavior of Timed Petri nets in which the infinite number of firing times possible from a certain marking $M$ are finitely represented by state classes. A state class is a pair ($M$,$D$) where $M$ is a marking and $D$ is a domain which is described as a system of inequalities. We have also developed a timing analysis for synthesis technique that uses the concept of system of inequalities although for a different class of time-extended Petri nets as will be discussed in Chapter 5.

In Generalized Stochastic Petri nets (GSPN) [1] a random variable with a known probability density function is associated to each transition of the net. Because of the memoryless property of the negative exponential density function $f(x) = \alpha\, e^{-\alpha x}$, most of the research on GSPN has assumed exponential random variables. It has been shown that a

GSPN with exponential random variables can be transformed to a discrete Markov chain [1]. However because potentially a transition can take arbitrarily long time to fire, it is difficult to place upper bounds on a timed execution, and thus the performance analysis using GSPN has focused on producing probabilistic averages.

To overcome that limitation, Juanole and Atamna [71] have proposed the stochastic timed Petri net (STPN) model in which the probability density functions of the random variables associated with the transitions of the net are of the form $f_i(x_i) = f_{ci}(x_i) + f_{di}(x_i)$, where $f_{ci}(x_i)$ is the continuous component, and $f_{di}(x_i)$ is the discrete component of $f_i(x_i)$. In [71] the authors only considered uniform probability density functions for the continuous component.

In the aforementioned time-extended Petri net models, time was associated with the transitions. Alternatively time can be associated with the places. We have chosen this alternative due to the intuitive interpretation in the realm of digital hardware that a marking of the net has a direct correspondence to the state of the system, and the firing of a transition indicates a change of state which is idealized to be instantaneous. Thus to us it seems more natural to associate time with places. Sifakis [117] first defined Timed Petri nets in which fixed time values were associated with the places.

Van der Aalst [126] introduced an extension to Sifakis Timed Petri nets in which intervals are associated with the places of the net. The firing rule is analogous to the one presented in Section 2.2.3. Our model is a natural extension of van der Aalst's in the sense that in it random variables are associated with the places of the net rather than just intervals.

Although Ramchandani also used the term Timed Petri nets to refer to his time extensions, in the sequel we shall differentiate between the Petri net models that assign time to transitions from the Petri net models that assign time to places by using the term Time for the former and Timed for the latter; and time-extended Petri nets shall refer in general to Petri nets with timing extensions. In the following section we present the time-extended Petri net model that we have developed in this dissertation.

### 2.2.3 Probabilistic timed Petri net model

The classic Petri net does not include an explicit representation of time. As discussed in the previous section, Petri nets have been extended to model time, by assigning arbitrary time values, time intervals, or random variables to transitions, or places, of the net. Other time extensions of Petri nets were discussed in Section 2.2.2. In this work we have developed a more general Petri net model in order to be able to handle correlation information which shall be further discussed in Sections 2.2.3 and 2.5.4, that we have called probabilistic timed Petri nets.

**Definition 2.2.1.-** A probabilistic timed Petri net is a quintuple $N = \langle P, T, F, M_0, \Gamma\rangle$ where $P$ is a non-empty set of places, $T$ is a non-empty set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, $M$: $P \to \aleph$ is the marking function and $M_0$ is the initial marking ($\aleph$ is the set of the non-negative integers), and $\Gamma$: $P \to \tau$ is the time labeling function that assigns to each place $p_i \in P$ a random variable (r.v.) $\tau(p_i)$ [105].

The preset (postset) of a transition $t$ is the set of incoming places to (outgoing places from) $t$ and is denoted $\bullet t$ ($t\bullet$). Similarly the preset (postset) of a place $p$ is the set of incoming transitions to (outgoing transitions from) $p$ and is denoted $\bullet p$ ($p\bullet$).

The random variables $\tau_i$'s are used to represent circuit delays as defined by the following firing rule:

**Firing rule**

1. A transition $t \in T$ is enabled when every place $p \in \bullet t$ contains a visible token.

2. An enabled transition must fire immediately (unless the firing of another enabled transition disables the transition instantaneously). When it fires, an enabled transition consumes a visible token in each place $p \in \bullet t$ and sends a token to each place $p \in t\bullet$.

3. A place $p_i$ upon receiving a token at time $\tau$ makes it visible to transitions $t \in p\bullet$ at time $\tau + \tau_i$, where $\tau_i$ is the random variable associated with place $p_i$. A place holds a visible token until it is consumed by the firing of an enable transition.

To illustrate the firing rule, consider the partial Petri net shown in Figure 2.2.2. Three transitions *a*, *b*, and *c* are connected to transition d through places labeled $\tau_1$, $\tau_2$ and $\tau_3$ respectively. Let us assume that the transitions *a*, *b*, and *c* fire at times $\tau_a$, $\tau_b$ and $\tau_c$ respectively. Then a token is placed in the firing transition's output place at the firing time. To represent a circuit delay, the place holds the token invisible to its output transition for certain time controlled by a random variable associated with the place. Let us assume that the three random variables $\tau_a$, $\tau_b$ and $\tau_c$ are independent and that their probability density functions are as shown in Figure 2.2.2. According to the firing rule transition *d* will fire as soon as there is a visible token in each of its input places.

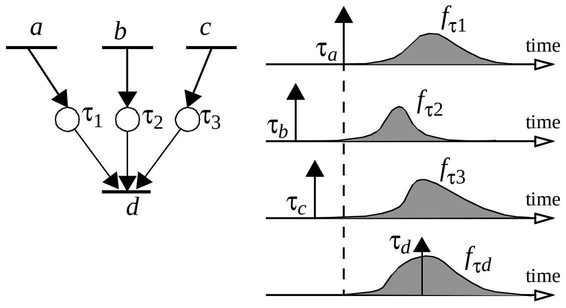


Figure 2.2.2 Probability density function of the firing time of a transition.

The firing of transition *d*, denoted by $\tau_d$, is a probabilistic event. Our approach to the analysis of probabilistic timed Petri nets is to find the probability density function of the firing (or occurrence) times of the transitions of a net. Chapter 3 describes how this is accomplished.

The probabilistic timed Petri net that we have introduced is a generalization of previous **Timed** Petri net models. In our model arbitrary probability density functions are associated with the places of the net. Furthermore, our model admits random variables that are not independent, a fact that plays an important role in the modeling of time correlation that appears in interface specifications of off-the-shelf hardware components.

Due to causality, it is required that the probability that any random variables $\tau_i$ take a negative value be zero. (For strict causality, the probability that the random variables $\tau_i = 0$ should be zero too.) The set of random variables $\tau_i$, $i = 1..M$, associated with the places of the net are fully described by the joint probability density function (in short pdf) $f_{\tau 1 \ldots \tau M}(\tau_1, \ldots \tau_M)$.

In some cases some of the random variables are independent, so that $f$ may have a compact form. For example, if all $\tau_i$ are independent then

$$f_{\tau 1 \ldots \tau M}(\tau_1, \ldots, \tau_M) = f_{\tau 1}(\tau_1) \ldots f_{\tau M}(\tau_M) \qquad \text{(Eq. 2.2.1)}$$

Of course in order to be able to model time correlation, one has to use the most general form in which not all random variables are independent.

The probabilistic aspect of our model has practical applications in describing interface specifications of components. An interface specification describes the behavior of not one but an ensemble of components. Thus a probabilistic approach to modeling seems very adequate to take into account variations in component behavior. Those variations are due to different instances of the same class of components affected by factors such as fabrication process, and different operational conditions such as temperature variations. We will exploit that in the reliability analysis of systems, that is we will be able to quantify not only if a system meets the (timing) constraints but also if it fails to meet some constraints, by how much. How to compute the probability that a constraint can be violated can be described by a probabilistic measure. This is the topic of Chapter 4.

### 2.2.4 Examples of probabilistic timed Petri nets

In this subsection we introduce two simple examples to give a flavor of probabilistic timed Petri nets. In particular in the second example we show the fact that time-extended Petri nets have a different behavior from classic Petri nets. More examples will be shown in this and following chapters. The firing rule will be discussed in more depth in Section 2.4.1.

The first example shown in Figure 2.2.3 consists of one place and two transitions. The only random variable associated with the net is described by the probability density function $f_x(x)$ (also shown in the figure). The initial marking is shown in the figure, thus at time $\tau_0 = 0$ there is a token in the place of the net. The token in the place is not visible to transitions $t_1$ or $t_2$ until a time $\tau_1 = x$, where the value of random variable $x$ follows the known pdf $f_x(x)$. Because the place is a free-choice place (refer to Section 2.2.1), either transition $t_1$ or $t_2$ will fire (but not both). Once a transition fires, it places a token in the place which will be made visible at $\tau_2 = \tau_1 + x$.

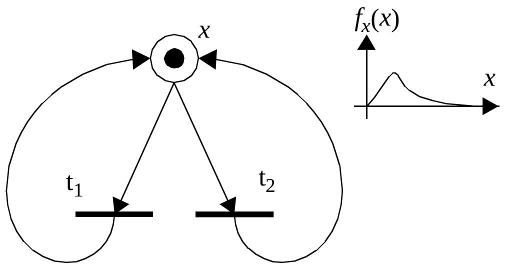


Figure 2.2.3 Petri net with a free choice place labeled with random variable $x$.

Note that there is a non-deterministic choice in the model for the firing of $t_1$ or $t_2$. We can use non-deterministic choice to abstract out some phenomena that are not relevant to our verification procedure. For instance if a hardware component is capable of performing either a read or a write cycle, this can be modeled using a free choice place because when attempting to verifying that both cycles meet the timing constraints (as will be discussed later) it is not important to know the ratio of read vs. write cycles, but just that both cycles can occur. From a performance point of view, assuming that a write cycle takes, say, longer than a read cycle, it might be important to determine the profile of read and write cycles to be able to quantify the performance of a system. In that case, one could also assign a scheduling variable to a free-choice place that computes (*e.g.* deterministically or probabilistically) which transition (of the several enabled in the current marking) should fire in an execution of the net. In the sequel we consider that the choice of firing transition is made non-deterministically.

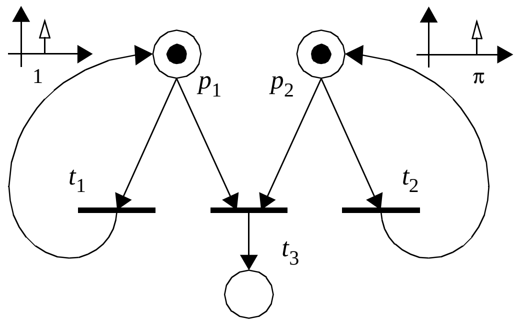


Figure 2.2.4 A probabilistic timed petri net that does not present deadlock.

The second example shown in Figure 2.2.4 consists of three places and three transitions. If transition $t_3$ fires, the system deadlocks. Random variables associated with places $p_1$ and $p_2$ are independent and their corresponding pdf's are Dirac's delta functions (if the pdf is the Dirac's delta function $f_\tau(\tau) = \delta(\tau\text{-}\tau_o)$, the token is made visible with probability 1 at time $\tau_o$). At time $\tau = 0$ both tokens are put in places $p_1$ and $p_2$ respectively. The token in $p_1$ will be made visible at $\tau = 1$, and $t_1$ will fire immediately. Similarly the token in $p_2$ will be made visible at $\tau = \pi$, and $t_2$ will fire immediately. It is clear that unlike the untimed (classic) version of the Petri net, the probabilistic timed Petri net in Figure 2.2.4 will never deadlock.

Of course, if more realistic pdf's are used to model the delays of places $p_1$ and $p_2$, such that the pdf's are non-zero for a (possibly infinite) interval, then deadlock will arise in the Petri net of Figure 2.2.4. However in our probabilistic timed Petri net, unlike classic Petri nets, one can quantify the probability of deadlock.

## 2.3 Signal transition graphs

Signal transition graphs, or STG's, are a widely used representation of asynchronous digital circuits [29, 115, 79, 124]. STG's are Petri nets whose transitions are interpreted as signal transitions of a circuit. In this section we extend STG's in the obvious way to use the probabilistic timed Petri net proposed in Section 2.2.3. Before doing so, we briefly overview previous related work on timed STG's.

### 2.3.1 Previous work on timed signal transition graphs

The work by Brzozowski *et al.* [17] aimed at providing a mathematical foundation to the interface timing verification problem. Their result holds for a restricted case of timing behavior, namely if every signal transition is caused by another single transition. McMillan *et al.* [82] presented a more general formulation of the timing verification problem and proved that it is NP-complete and developed algorithms for sub-cases of the problem. Independently Burks *et al.* [20] followed a mathematical programming approach to solve a class of problems which includes the interface timing verification problem and suggested a branch-and-bound algorithm to solve the problem which is worst-case exponential in time. The above research did not use an underlying Petri net model, however it uses mathematical programming techniques that are the foundation of the techniques we shall present in this dissertation.

STG's were first used for the specification and synthesis of asynchronous digital circuits in [29, 115]. No time annotation was used in the underlying Petri net model. Vanbekbergen [124], Rockicki [114], and Escalante and Dimopoulos [46] proposed similar timing extensions to STG's to represent timing in asynchronous digital circuits. Vanbekbergen [124] proposed a Petri-net based model, called timed STG's, that he used to represent asynchronous circuits with time bounds. Independently Rokicki [114] proposed another Petri-net based model, called orbital nets, to model a class of digital logic. Independently Escalante and Dimopoulos [46] used a Petri-net based model similar to Vanbekbergen's timed STG's, to specify component interface protocols and associated interface logic. An important feature of all three models is that they make a clear distinction between circuit delays and timing constraints in the specification of component behavior.

Myers and Meng [97, 98] used a conservative estimate of gate delays to remove redundant edges in an STG; with their technique they could synthesize much simpler circuits thus showing the advantage of taking timing into account. Hulgaard and Burns [67] have developed algebraic techniques to find bounds on the maximum time separation between two given signal transitions of a timed STG. Their results are exact for Petri nets without choice, but they also explored approximations for free choice Petri nets.

In the research mentioned so far in this section, timing is represented using intervals. In [48, 49] we proposed a more general STG model with an underlying probabilistic timed Petri net. Thus we needed to develop novel time verification techniques that shall be presented in Chapter 4. Moreover, as explained in the Introduction, the other main goal of this dissertation is to determine tight bounds on interface logic prior to synthesis, a technique called timing analysis for synthesis that shall be discussed in Chapter 5. Before tackling those tasks we need to complete the presentation of our timed STG model.

## 2.3.2 Components, ports, signals and signal states

A *component* communicates with its environment through *ports*. A port has a direction associated with it. The direction of a port can be input or output. An input port accepts information from the environment, while an output port sends information to the environment. Several ports can be grouped together into a combined port. Bi-directional ports can be modeled as two ports, one of type input and one of type output. A *combined* port is an *n*-bit port, where *n* is the number of single ports that comprise the combined port. A *single* port is also called a 1-bit port. Another common term used to describe a port is *line*. For example the 32 data lines of a memory component constitute a 32-bit port.

*Signals* are the means to convey information. The relationship between a port and a signal is that a port is an entity that can be physically located usually on the boundary of a circuit, and a signal associated with such a port describes the value of the port as a function of time. Most current implementations of electronic digital circuits use electrical signals, although optical and other physical media can be used as well. We use a continuous model of the time domain (also called dense time) although discrete models have also been studied in the literature. In general discrete time models are computationally simpler but suffer the problem of resolution accuracy (*i.e.*, what is the right granularity to properly describe the nuances of time, *cf.* [114]). In this dissertation we consider digital signals. The range of values that a digital signal can take is discrete and is called the set of *states* of the signal. The states of a digital signal in a single port, in the simplest case (called binary case), are logic ‘0’ and logic ‘1’. Tri-stated signals can be *floating*, or in a high-impedance state ‘Z’, too. We supplement these basic states with the following states:

**Valid**: This state is particularly useful to describe the state of a combined port whose individuals signals are binary. A valid state of a combined port occurs when such a port has a value within a range of allowed values. The particular value of the signal at the combined port is not important nor is the fact that the port carries a value that can be used by another part of the system. For example, when the value of a group of data lines of a certain component is valid, it can be read by another component. A valid state for a group of signals is an effective way of describing a large number of states compactly. For example, a valid state for a 32-bit binary data port (*i.e.*, whose individual ports can take only the values ‘0’ and ‘1’) of a memory component may represent $2^{32}$ states. This can be advantageously exploited to reduce the number of cases to consider for representation, analysis or verification purposes whenever the actual value on the port is not relevant.

**Invalid**: This state is complementary to the valid state of a combined port. The relevant piece of information is not the particular value at the combined port but the fact that the value should not be used by another component. For example, when the address lines of a component are changing, their state is invalid and should not be used for decoding purposes.

**Driven**: A tri-stated signal is driven if it is not in a high-impedance state. Thus a driven binary signal is either ‘0’ or ‘1’. A driven signal can be valid or invalid. For example a don’t care state ‘X’ of a binary signal can be modeled using a driven state.

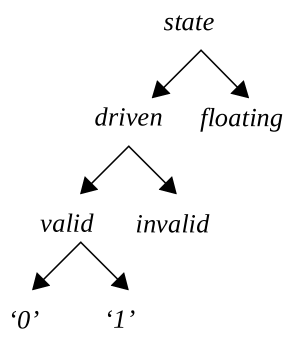


Figure 2.3.1 Signal states.

**Floating**: A tri-stated signal is floating if it is in a high-impedance state.

We define the *includes* binary relation $I$ on the set of signal states as shown by the directed graph in Figure 2.3.1 such that there is a directed edge from state $s_1$ to state $s_2$ if $s_1\ I\ s_2$. The include relation is important when trying to determine if two ports can be connected (refer to Definition 2.3.4). Before we discuss this, we need to give some basic definitions on the description of signals which are adapted from a similar treatment described in [17].

**Definition 2.3.1.-** A (possibly infinite) *timed state sequence* of port $p$ is the sequence $ls_p = \{s_0, \tau_0, s_1, \ldots, \tau_{n-1}, s_n\}$, where $s_i$ are signal states and $\tau_i$ are times, such that $s_i \neq s_{i+1}$ and $\tau_j \leq \tau_{j+1}$ for $i = 0, \ldots, n-1$ and $j = 0, \ldots, n-2$. The sub-sequence $\{\tau_0, \ldots, \tau_{n-1}\}$ is called the time sub-sequence of $ls_p$.

**Definition 2.3.2.-** A *signal transition* is a pair $\langle s_1, s_2\rangle$ of states where $s_1 \neq s_2$.

**Definition 2.3.3.-** If $ls_p = \{s_0, \tau_0, s_1, \ldots, \tau_{n-1}, s_n\}$ is the state sequence of port $p$, the corresponding *timed signal transition sequence* of port $p$ is given by sequence $lt_p = \{\langle\tau_0, s_0, s_1\rangle, \langle\tau_1, s_1, s_2\rangle, \ldots, \langle\tau_{n-1}, s_{n-1}, s_n\rangle\}$.

A timed state sequence is an enumerative description of the signal associated with a port $p$ (*i.e.*, the values that port $p$ takes as a function of time). The time subsequence $\{\tau_0, \ldots, \tau_{n-1}\}$ indicates the instants when the port change state. The port is in state $s_0$ during $-\infty < \tau < \tau_0$, in state $s_n$ during $\tau_{n-1} \leq \tau < \infty$, and in general in state $s_i$ during $\tau_{i-1} \leq \tau < \tau_i$ for $i = 1, \ldots, n-1$.

A signal transition describes a change in port $p$ from state $s_1$ to state $s_2$. Although the values of the time sub-sequence are not strictly increasing, *i.e.*, any number of signal transitions are allowed to occur at any instant $\tau$, we only consider in this work state sequences (or timed signal transition sequences) for which there is a finite number of signal transitions that occur at any given time $\tau$.

**Definition 2.3.4.-** Let us assume that two ports $p_1$ and $p_2$, having input and output direction respectively. If for any given time $\tau$ the values of the ports $p_1$ and $p_2$ are $s_1$ and $s_2$ respectively, and $s_1\ I^*\ s_2$, where $I^*$ is the reflexive and transitive closure of $I$, then ports $p_1$ and $p_2$ are said to be *compatible*.

The definition of compatibility of two connected ports, one of them being an input port and the other being an output port, restricts the state of the output port to those included by the state of the input port, *i.e.* those states at or below the input state node in the state graph of Figure 2.3.1. Two compatible ports can be connected via a wire. In that case the value of the input port follows the value of the output port.

The alphabet $A(p)$ of a port $p$ is the set of signal transitions $\{\langle s_i, s_j \rangle\}$ of the timed signal transition sequence $lt_p$. Notice that $A(p)$ is finite. The alphabet of a set of ports $P$ is given by $A(P) = \bigcup_{p \in P} A(p)$ .

We use the following notational conventions: a port whose direction is always an input is denoted with its name underlined. A port whose direction is always an output is denoted with its name overlined.

We deal now with some implementation issues. The logic levels of a signal are implemented as physical values of a circuit. Without loss of generality let us consider the implementation of logic levels using voltage levels. For a port that uses positive logic, a low voltage corresponds to logic ‘0’ and a high voltage corresponds to a logic ‘1’. For a port that uses negative logic, a low voltage corresponds to logic ‘1’ and a high voltage corresponds to a logic ‘0’. To distinguish the logic implementation of a port, we append a ‘*’ as a suffix to the name of a port $p$ that uses negative logic (*e.g.*, $p*$). Because logic values rather than voltage values are more meaningful in the description of signal transitions, we use the terms *asserted* (*negated*) to denote a signal transition from ‘0’ to ‘1’ (from ‘1’ to ‘0’), independently of the logic implementation.

| **transition** | **symbol** |
|---|---|
| $\langle$*negated*, *asserted*$\rangle$ | $p+$ |
| $\langle$*asserted*, *negated*$\rangle$ | $p-$ |
| $\langle$*invalid*, *valid*$\rangle$ | $p+_v$ |
| $\langle$*valid*, *invalid*$\rangle$ | $p-_v$ |
| $\langle$‘Z’, *driven*$\rangle$ | $p\uparrow$ |
| $\langle$*driven*, ‘Z’$\rangle$ | $p\downarrow$ |

Table 2.3.1. Notable transitions on port $p$.

For some notable transitions we use the special symbols given in Table 2.3.1. Please the reader be aware of our usage of negated with the opposite meaning of asserted (*cf.* [119]), although in other areas negated has the connotation of logic inversion.

## 2.3.3 Timed signal transition graphs

In the previous section we proposed a signal state lattice to describe the value of a port. The lattice allows us to define compatibility of port connection in a straightforward way. In this section we introduce signal transition graphs (STG’s) which are Petri nets whose transitions are associated with signal transitions.

**Definition 2.3.5.-** An (extended) timed STG is a tuple $\Sigma = \langle N, Y, \lambda\rangle$ where $N$ is a probabilistic timed Petri net, $Y$ is as set of ports, and $\lambda$: $T \rightarrow A(Y) \cup \{\varepsilon\}$ is a signal transition labeling function which assigns transitions $t \in T$ of the Petri net to signal transitions $a \in A(Y)$ or the silent signal transition $\varepsilon$, where $A(Y)$ is the alphabet of $Y$.

In the sequel we use the terms transition and signal transition interchangeably whenever there is no possibility of confusion.

Figure 2.3.2 shows a probabilistic timed Petri net (left) and a corresponding timed signal transition graph (right). The Petri net consists of the set of places $P = \{p_0, p_1, p_2, p_3\}$, the set of transitions $T = \{t_0, t_1, t_2, t_3\}$, the flow relation

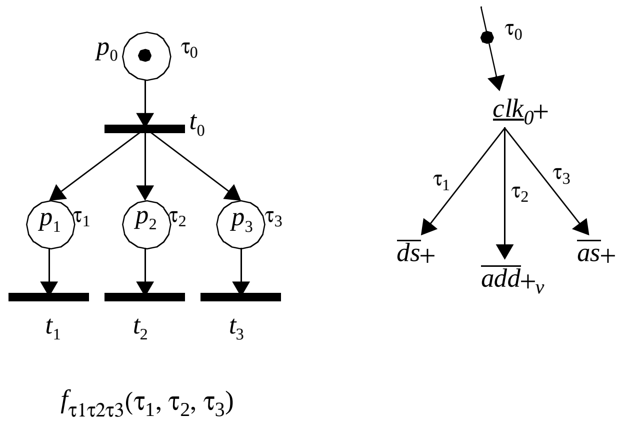


Figure 2.3.2 Signal transition graph.

$F = \{(p_0,t_0), (t_0,p_1), (t_0,p_2), (t_0,p_3), (p_1,t_1), (p_2,t_2), (p_3,t_3)\}$, the initial marking $M_0 = \{(p_0,1), (p_1,0), (p_2,0), (p_3,0)\}$, and the time labeling function $\Gamma = \{(p_0,\tau_0), (p_1,\tau_1), (p_2,\tau_2), (p_3,\tau_3)\}$. The joint probability function $f_{\tau_1\tau_2\tau_3}(\tau_0, \tau_1, \tau_2, \tau_3)$ fully characterizes the set of random variables $X = \{\tau_0, \tau_1, \tau_2, \tau_3\}$. To draw the STG we use the usual convention according to which a place with a single input transition and a single output transition is shown as an edge labeled with the random variable associated with the place. The set of ports is $Y = \{\underline{clk}_0, \overline{add}, \overline{as}, \overline{ds}\}$, and the signal transition labeling function is $\lambda = \{(t_0,\underline{clk}_0+), (t_1,\overline{ds}+), (t_2,\overline{add}+_v), (t_3,\overline{as}+)\}$.

### 2.3.4 Signal transition graphs and signal transition sequences

In Section 2.3.2, we introduced timed signal transition sequences of tuples $\langle\tau_{i-1}, s_{i-1}, s_i\rangle$, which describe a change in the value of a port at time $\tau_{i-1}$ from state $s_{i-1}$ to state $s_i$ to describe the signal activity at a port. In that subsection our main goal was to formally define a signal transition and we were not concerned about how to represent the behavior of ports using such a sequence. A (potentially infinite) signal transition sequence describes one possible observation of the activity at a port. An also potentially infinite set of signal transition sequences is necessary to describe all the possible behaviors even of simple ports. For example, if the places of the timed STG shown in Figure 2.3.3 are associated with independent random variables whose probability density function is the uniform probability density function defined in the interval [1, 1.01], one possible signal transition sequence is the infinite sequence $\{\langle 0, a+\rangle, \langle 1, b+\rangle, \langle 2, a-\rangle, \langle 3, b-\rangle, \ldots\}$ as it is the also infinite sequence $\{\langle 0, a+\rangle, \langle 1.001, b+\rangle, \langle 2.001, a-\rangle, \langle 3.001, b-\rangle, \ldots\}$, and so on.

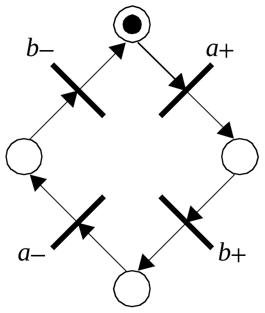


Figure 2.3.3 Simple STG.

Thus signal transition sequences are limited in their expressiveness in the sense that they describe only one observation. Typical component interface specifications comprise the behavior of an ensemble of components and thus they must allow for variations. Rather than listing a possibly infinite set of observations, signal transition graphs can compactly describe the behavior of an ensemble of variations. Signal transition sequences are useful as the formal underlying semantics of a single observation. For instance they are used in [11] to analyze the behavior of time Petri nets. Of course in [11], sequences are grouped into classes. A class potentially represents an infinite number of observations. On the other hand, signal transition sequences are more general than timed STG's, that is, there are sets of signal transition sequences which cannot be expressed by a timed STG. For example an infinite sequence in which events *a*, *b*, *c*, and *d* appear randomly such that a given event cannot be followed by itself.

From the previous discussion it is clear that a timed STG is a compact representation of multiple possible behaviors. In the sequel we shall use STG's to describe timed behaviors. However notice that a brute force approach to the analysis of timed STG's is not feasible, due to the infinite number of sequences that one has to consider. In the following section we discuss a subset of timed STG's for which we have developed techniques that analyze "classes" of behaviors rather than individual sequences.

## 2.4 Subclass of STG's

Analyzing the timing behavior of an arbitrary net topology is a difficult problem. In this dissertation we have developed timing analysis techniques for the sub-class of Petri nets which describe only AND and OR causality.

### 2.4.1 AND and OR causality

The interface timing verification has been formulated as the solution of a set of linear/min/max inequalities [6]. In this section we link this result to two types of causality discussed in the concurrent systems community: AND and OR causality.

We first present the classical definitions of AND and OR causality, without explicit time. A transition *t* is said to be AND-caused by a set *S* of transitions if *t* occurs after all the transitions in *S* in all signal transition sequences (refer to Definition 2.3.3). Similarly, a transition *t* is said to be OR-caused by a set *S* of transitions if *t* occurs after the first transition in *S* occurs.

When delays are taken into consideration, the definitions of AND and OR causality are modified as follows: the effect of a transition takes place at time $\tau_t + \tau_d$, where $\tau_t$ is the time when the transition occurred and $\tau_d$ is the delay of the transition. (Notice that in our case, the delay is given by a random variable.) A transition *t* is said to be AND-caused by a

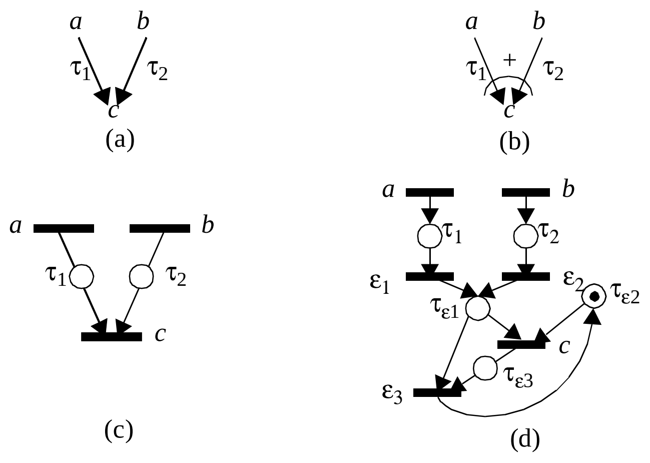


Figure 2.4.1 Causality classes: (a) AND causality; (b) OR causality; (c) and (d) Petri net constructs.

set *S* of transitions if *t* occurs after all the effects of transitions in *S* have taken place in all possible signal transition sequences. Similarly, a transition *t* is said to be OR-caused by a set *S* of transitions if *t* occurs after the earliest effect of a transition in *S* has taken place in all possible signal transition sequences. For example in Figure 2.4.1a, transition *c* occurs only after the effects of *a* **and** *b* have taken place. In Figure 2.4.1b, transition *c* occurs as soon as the earliest effect of *a* **or** *b* takes place.

Figures 2.4.1c and d depict the corresponding Petri net constructs [131]. Clearly, AND causality is a direct mapping of the firing behavior of our probabilistic timed Petri nets: transition *c* will fire only when there are visible tokens in all its input places; the tokens in places labeled with random variables $\tau_1$ and $\tau_2$ make the token visible to transition *c* sometime after transitions *a* and *b* respectively have occurred (the delay is controlled by the respective random variable).

The Petri net fragment that implements OR causality is more involved (refer to Figure 2.4.1). For the sake of the following presentation, we distinguish between the three silent transitions $\varepsilon$ by assigning a subscript to each of them. We also use the convention of calling a place labeled with random variable $\tau$, a $\tau$ place. The random variable $\tau_1$ (or $\tau_2$) represents a delay after transition *a* (or *b*). Places labeled with random variables $\tau_{\varepsilon i}$ are bound to make tokens visible immediately (*i.e.*, the probability density function of the random variable is the Dirac's delta function at the origin). Let us assume that transition *a* occurs first and that the value of $\tau_1$ is less than the value of $\tau_2$. Thus the silent transition $\varepsilon_1$ will fire sometime after $\tau_1$ and when it fires, it will send a token to place $\tau_{\varepsilon 1}$. The place $\tau_{\varepsilon 1}$ will make the token visible immediately and transition *c* will fire after delay $\tau_1$. When *c* fires, the token in places $\tau_{\varepsilon 1}$ and $\tau_{\varepsilon 2}$ are consumed and a token is sent to place $\tau_{\varepsilon 3}$. After transition *b* fires, another token is sent to place $\tau_{\varepsilon 2}$ after $\tau_2$, and then the silent transition $\varepsilon_3$ fires, putting a token back in place $\tau_{\varepsilon 2}$. Notice that for proper operation of the OR causality, there must be a token in place $\tau_{\varepsilon 2}$ at the beginning of an OR cycle. In Section 2.5.2 we shall use the concept of a constraint rule to guarantee that the OR sub-net is properly initialized.

For the sake of clarity from now on we adopt the more compact representation shown in Figures 2.4.1a and b. We call these two compact representations AND and OR arcs respectively. In the sequel we shall develop a timing verification procedure for the subclass of Petri nets which have only AND and OR causality arcs.

Let us consider the multiple AND join, that is a transition with multiple incoming places, shown in Figures 2.4.1a and c. Places are associated with random variables $\tau_i$ which, without loss of generality, are assumed to have independent probability density functions $f_{\tau i}(\tau_i)$. After the firing of a transition, say *a* at time $\tau_a$, a token is made visible to transition *d* at time $\tau_a + \tau_1$, with pdf $f_{\tau i}(\tau_1)$. According to the firing rule, *c* fires when all tokens in *a*, and *b* are visible, which happens at:

$$\tau_c = max(\tau_a + \tau_1, \tau_b + \tau_2) \qquad \text{(Eq. 2.4.1)}$$

Similarly for a multiple OR join (refer to Figures 2.4.1b and d), transition *c* will fire as soon as the first of *a* or *b* occurs, which happens at:

$$\tau_c = min\ (\tau_a + \tau_1,\ \tau_b + \tau_2) \qquad \text{(Eq. 2.4.2)}$$

Note that both AND and OR causality collapse to linear causality (*i.e.*, one transition is uniquely caused by another transition) if there is only one predecessor to the generated transition *d*.

### 2.4.2 The AOC class of timed signal transition graphs

We have developed timing verification and analysis techniques for a subclass of signal transition graphs. Although limited to this subclass of STG's, we have been able to describe interface specifications for a great variety of off-the-shelf components using this subclass that we have called the AOC class. AOC stands for AND and OR causality STG's.

The AOC class of nets comprises STG's with the following properties:

1. Only AND and/or OR causality are allowed (refer to Section 2.4.1).

2. All the places of the STG are safe (refer to Section 2.2.1), except the unique choice places of OR causality edges which are *k*-bounded, where *k* is the number of edges.

The AOC class is more general than the class of marked graphs which is properly included in AOC. Recall that OR causality introduces unique choice places, which are not allowed in marked graphs, and AOC STG's are not necessarily strongly connected. However neither free choice, nor arbitration choice, are allowed in AOC (refer to Section 2.2.1).

It is possible to represent AOC STG's using AND and OR arcs (Figures 2.4.1a and b), that we have called AOC directed graphs or di-graphs, although it is straightforward to obtain their Petri net representation of an AOC di-graph.

**Definition 2.4.1.-** An AOC di-graph is the labeled di-graph $\Omega = \langle T_\Omega, P_\Omega, M_{\Omega 0}, \Gamma_\Omega, ct_\Omega, Y, \lambda_\Omega \rangle$, where $T_\Omega$ is a non-empty set of nodes, $P_\Omega \subseteq T_\Omega \times T_\Omega$ is a set of edges, $M_{\Omega 0}$: $P_\Omega \rightarrow \aleph$ is the initial marking function that assigns to each edge a non-negative number of tokens ($\aleph$ is the set of the non-negative integers), $\Gamma_\Omega : P_\Omega \rightarrow \tau$ is the time labeling function that assigns to each edge $p_i \in P_\Omega$ a random variable (r.v.) $\tau(p_i)$, $ct_\Omega : T_\Omega \rightarrow \{\text{AND, OR}\}$ is the causality type function that assigns to each node the type AND or OR, $Y$ is a set of ports, and $\lambda_\Omega$: $T_\Omega \rightarrow A(Y) \cup \{\varepsilon\}$ is a signal transition labeling function which assigns to each node a signal transition $a \in A(Y)$ or the silent signal transition $\varepsilon$, where $A(Y)$ is the alphabet of $Y$.

Notice the analogies between an AOC di-graph $\Omega = \langle T_\Omega, P_\Omega, M_{\Omega 0}, \Gamma_\Omega, ct_\Omega, Y, \lambda_\Omega \rangle$ and an STG $\Sigma = \langle N, Y, \lambda \rangle$ and its associated probabilistic timed Petri net $N = \langle P, T, F, M_0, \Gamma \rangle$. The sets of nodes and edges, $T_\Omega$ and $P_\Omega$, of the AOC di-graph describe the connectivity of the graph; $P$, $T$ and $F$ in the STG achieve the same purpose. The marking, time labeling, and signal transition labeling functions have similar form. The only function characteristic of AOC di-graphs is the causality type function $ct_\Omega$, which is used to specify the firing semantics as discussed in Section 2.4.1.

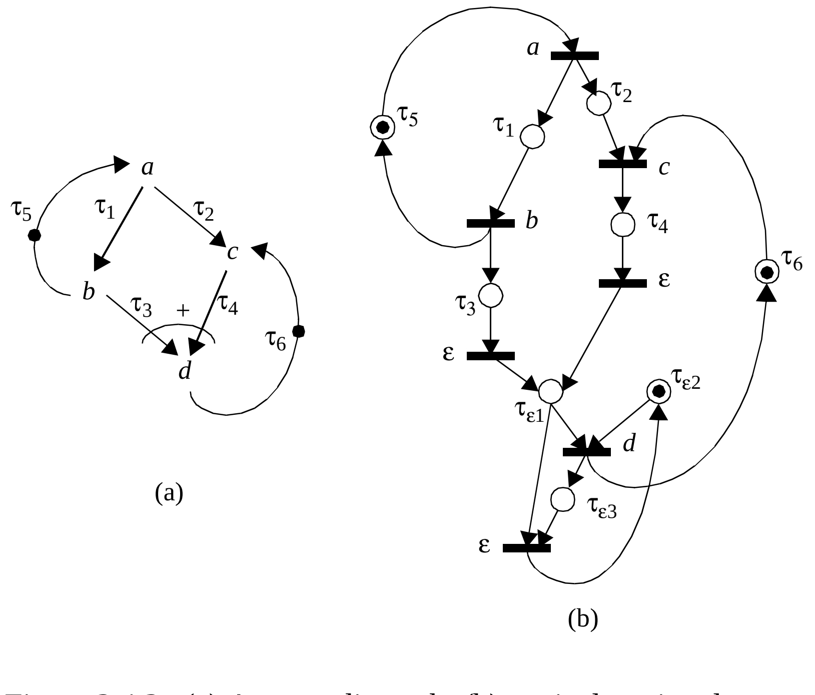


Figure 2.4.2 (a) An AOC di-graph; (b) equivalent timed STG.

An example of AOC di-graph is shown in Figure 2.4.2a. Signal transition *c* is AND-caused by transitions *a* and *d*, while transition *d* is OR-caused by *b* and *c*. The equivalent timed STG is shown in Figure 2.4.2b. Although we shall use the graphic representation of AOC di-graphs, which is more compact, in our figures, we shall refer to the AOC digraph $\Omega = \langle T_\Omega, P_\Omega, M_{\Omega 0}, \Gamma_\Omega, ct_\Omega, Y, \lambda_\Omega \rangle$ or its equivalent STG $\Sigma = \langle N, Y, \lambda \rangle$ with the associated net $N = \langle P, T, F, M_0, \Gamma \rangle$ indistinctly.

## 2.5 Interface specifications

In the previous sections we have presented timed STG's which are suitable to describe the (internal) behavior of components. However, specifications of components include information about the environment. For instance, a component may specify that an input signal should not be changed in certain interval during which the input is being sampled. If those environment specifications are not followed, the component may not behave as expected.

In this section we introduce a new type of timing relationship, different from the places of the probabilistic timed Petri net that are suitable to model (circuit) delays. We called them constraint rules. In the literature, the term "constraint" has been used to designate both circuit delays and timing constraints. To avoid confusion, we differentiate them explicitly and when we want to refer to both we called them timing relationships. It shall be clear from our formal description of constraint rules, that circuit delays and timing constraints have different semantics. Vanbekbergen [124] and Rokicki [114] used two types of places with different firing semantics to describe delays and timing constraints. We argue in the following section that places are not the best model for timing constraints.

### 2.5.1 Constraint rules

Event-rule (ER) schemata were introduced in [22] to analyze the performance of asynchronous circuits. It was modified in [98] to represent timing constraints in a circuit specification. An ER schema consists of a set of atomic actions, called events, a set of causal dependencies between them, called rules, and an initial marking, which is a subset of the set of rules. A rule is of the form $\langle e, f, \varepsilon, \tau\rangle$ where $e$ and $f$ are two events, $\varepsilon$ is defined to be 1 if the rule belongs to the initial marking and 0 otherwise, and $\tau = [l, u]$ is a compact interval with lower bound $l$ and upper bound $u$. In [98] timing constraints and delays are treated in the same manner. In this dissertation we make the distinction between timing constraints and circuit delays. We adapt the ER system as follows:

**Definition 2.5.1.-** A constraint rule associated with the timed probabilistic Petri net $N = \langle P, T, F, M_0, \Gamma\rangle$ is a tuple $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon\rangle$ where $t_i, t_j \in T$ is a pair of transitions of the net, $\Delta_{ij} \in \Im$ is a non-empty compact real interval ($\Im$ is the set of compact real intervals [112]), and $\varepsilon$ is an integer in $\{0, 1\}$.

Let us denote the time of firing of the $i$-th occurrence of transition $a$ as $\tau_{a(i)}$. The interval $\Delta_{ij}$ of a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon\rangle$ defines a time window with respect to the $k$-th occurrence of transition $t_i$ ($t_{i(k)}$), given by $\tau_{ti(k)} + \Delta_{ij}$, such that the ($k$+$\varepsilon$)-th occurrence of $t_j$ ($t_{j(k+\varepsilon)}$) must occur within this window for any occurrence index $k$. Notice that the bounds of $\Delta_{ij}$ are not required to be non-negative.

**Definition 2.5.2.-** The set of constraint rules $C_N = \{c_{ij}\}$ associated with the timed probabilistic Petri net $N = \langle P, T, F, M_0, \Gamma\rangle$ denotes a set of timing constraints given on transitions of the net $N$.

As illustrated in Figure 2.5.1 for constraint rule $c_{ij} = \langle a, b, \Delta_1, 0\rangle$, once the $k$-th occurrence of transition $a_{(k)}$ fires at time $\tau_{a(k)}$, then the ($k$+$\varepsilon$)-th occurrence of transition

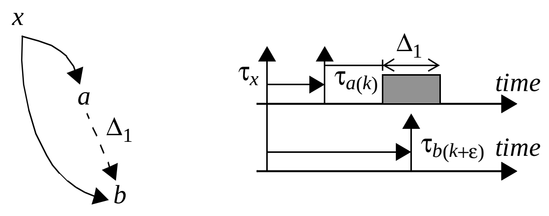


Figure 2.5.1 Constraint rule for transitions *a* and *b*.

$b_{(k)}$ must occur during the interval $\tau_{a(k)} + \Delta_1$, where interval addition [112] is used, otherwise the constraint rule is *violated*. It is said that transition *b* is constrained by the firing of transition *a*, or equivalently that transition *a* is constraining transition *b*.

Although constraint rules are defined for transitions of a probabilistic timed Petri net, the extension of constraint rules to be defined for signal transitions of a signal transition graph can be done in the obvious way by using the signal transition labeling function $\lambda$ of the STG (refer to Section 2.3.3). Graphically in an AOC STG, a constraint rule can be depicted as a directed edge from the constraining transition to the constrained transition, labeled with the constraint interval, as shown in Figure 2.5.1. We use the convention that constraint edges are drawn using dotted lines. Because $\varepsilon$ can take only the values 0 or 1, in the former case the directed arc connects two transitions belonging to the same cycle; in the latter case, the directed arc “folds back” connecting a transition belonging to a current cycle to a transition which belongs to the next cycle (called $\varepsilon$ constraints in [46]).

There is no causality implied in a constraint rule because the lower bound of the constraint interval can be negative, in which case the constrained transition can occur before the constraining transition. As a matter of fact if the upper bound of the constraint interval is also negative, the constrained transition must occur before the constraining transition. We shall give an example in Section 2.5.5. It is this fact that makes difficult to represent timing constraints using places.

The trivial constraint interval is $\Delta = (-\infty, +\infty)$, which signifies that the constrained transition is not constrained at all by the firing of the constraining transition. If the constraint interval is non-negative (*i.e.*, its lower bound is non-negative), the constraint rule is called a *sequencing* constraint.

The basis of our timing verification and timing analysis for synthesis procedures is the satisfaction of the constraint rules. In Chapter 3 we develop some concepts that shall prove useful to check that the constraint rules are not violated.

## 2.5.2 OR causality revisited

In Section 2.4.1 we mentioned that for proper operation of the OR causality sub-net (refer to Figure 2.5.2) it is required that there be a token in place $\tau_{\varepsilon 2}$ before a token arrives at place $\tau_{\varepsilon 1}$ due to the firing of the first of $a_{(k)}$ or $b_{(k)}$, for any $k > 0$. In this section we show that sequential constraint rules (refer to Section 2.5.1) can be used to detect this potential problem.

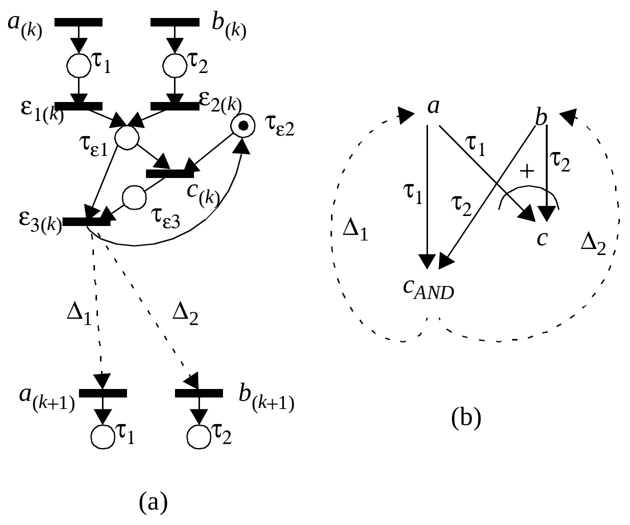


Figure 2.5.2 OR causality constraints.

Consider the sub-net shown in Figure 2.5.2a where transition $c$ is OR-caused by transitions $a$ and $b$. We shall show that by adding the sequential constraint rules $c_1 = \langle \varepsilon_3, a, \Delta_1, 1 \rangle$ and $c_2 = \langle \varepsilon_3, b, \Delta_2, 1 \rangle$, it is guaranteed that, in a correct behavior, the $k$-th occurrence of transitions $a$ and $b$ must precede the ($k$+1)-th occurrence of transition $c$.

**Theorem 2.5.1.-** Let us assume that the initial marking assigns a token to place $\tau_{\varepsilon 2}$ and no tokens to places $\tau_{\varepsilon 1}$ and $\tau_{\varepsilon 3}$. Then if constraint rules $c_1 = \langle \varepsilon_3, a, \Delta_1, 1 \rangle$ and $c_2 = \langle \varepsilon_3, b, \Delta_2, 1 \rangle$ are satisfied, where $\Delta_1$ and $\Delta_2$ are non-negative non-empty compact intervals, there is a token in place $\tau_{\varepsilon 2}$ before a token arrives at place $\tau_{\varepsilon 1}$ due to the firing of $a_{(k)}$ or $b_{(k)}$ for any index $k > 0$.

**Proof.-** Clearly for $k = 1$ the OR sub-net is properly initialized. We have to check that this is also the case for $k > 1$. If the sequential constraint rules $c_1$ and $c_2$ are satisfied, then, using the fact that $\Delta_1$ and $\Delta_2$ are non-negative non-empty compact intervals, the $k$-th occurrence of transition $\varepsilon_3$ must precede the ($k$+1)-th occurrence of both transitions $a$ and $b$, for any $k > 1$. When transition $\varepsilon_3$ fires, it sends a token immediately to place $\tau_{\varepsilon 2}$. But place $\tau_{\varepsilon 2}$ makes the token visible immediately. Therefore a token is visible at place $\tau_{\varepsilon 2}$ prior to or at the same time as the first of the ($k$+1)-th occurrences of transitions $a$ or $b$. □

In our short-hand graphical notation, we describe constraint rules $c_1$ and $c_2$ as constraint edges from a virtual transition $c_{AND}$ which is AND-caused by transitions $a$ and $b$. Notice that in this case the same random variable is associated to two edges, the original OR edge and the virtual AND edge. Thus the firings of the virtual $c_{AND}$ and $\varepsilon_3$ are indistinguishable.

### 2.5.3 Interface specifications

The aim of this chapter is to develop a formal representation capable of describing accurately the temporal behavior of off-the-shelf components of hardware systems, in particular components of microprocessor-based systems. Because a system is comprised of the interconnection of several components, there are two facets in component specifications: the internal operation of the component and the set of constraints that the environment must satisfy for proper operation. In the previous sections we have developed two formal structures: timed signal transition graphs suitable to model delays, and constraint rules suitable to describe timing constraints. In this section we put both structures together to form a new structure that we call interface specification.

**Definition 2.5.3.-** Given a timed signal transition graph $\Sigma = \langle N, Y, \lambda \rangle$ and a set of constraint rules $C_N = \{c_{ij}\}$associated with net $N$, an interface specification is the pair $\Psi = \langle \Sigma, C_N \rangle$.

In preparation for the examples of component interface specifications, in the following section we present a simple transformation

## 2.5.4 Projections

Our goal is to use interface specifications to formally specify the behavior of hardware components. Manufacturers usually provide this information in the form of timing diagrams and associated timing parameters. In this section we address the problem of interpreting the timing information from the timing diagrams to construct an equivalent interface specification.

As mentioned in the previous section, an interface specification consists of a timed signal transition graph and a set of constraint rules. The signal transition graph is composed of places, transitions, the connectivity of the net, the initial state, a set of random variables associated with the places, and a set of signal transitions associated with the transitions. A constraint rule defines an occurrence window for a transition with respect to a reference transition. Thus in our model, operational timing is captured by the set of random variables, and environmental timing is specified by the $\Delta_{ij}$ windows of constraint rules.

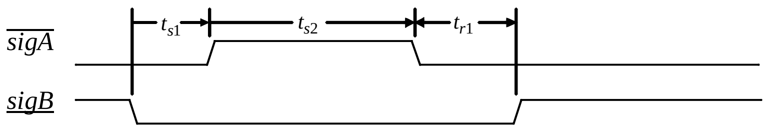


Figure 2.5.3 A simple timing diagram.

In component data sheets, timing parameters may be classified as delays and timing constraints (a common nomenclature is to call them switching characteristics and timing requirements respectively), although sometimes there is no indication of the type of parameter and ingenuity from the part of the designer is required to interpret the parameters correctly. Consider for example the simple timing diagram shown in Figure 2.5.3. The timing diagram describes three timing relationships between two signals, $\overline{sigA}$ and $\underline{sigB}$ (according to our notation, $\overline{sigA}$ is an output signal, and $\underline{sigB}$ is an input signal). One can interpret timing relationship $t_{r1}$ as a timing constraint, and the other two, $t_{s1}$ and $t_{s2}$, as delays. It is easy to see why $t_{r1}$ is a timing constraint, if one considers that it goes from an output signal transition to an input signal transition, and therefore it could not be an internal propagation delay, but a restriction on the environment. Analogously, $t_{s1}$ should be a delay because it relates an input signal transition to an output signal transition.

The sequence of signal transitions implied by the above timing relationships is described by the graph shown in Figure 2.5.4, where solid lines represent the two delays, and the dotted line describes the timing constraint.

Timing parameters are usually given as intervals $[\tau_{min}, \tau_{max}]$ in the data sheets. For the timing diagram of Figure 2.5.3, the timing parameters are: $t_{s1} = [5, 10]$, $t_{s2} = [0, 10]$,

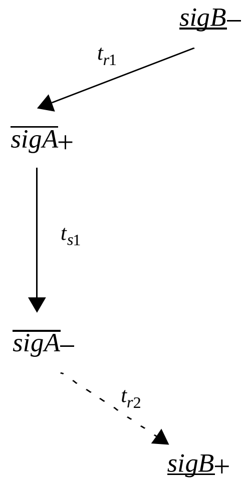


Figure 2.5.4 Interface specification corresponding to the timing diagram shown in Figure 2.5.3.

and $t_{r1} = [5, \infty)$. The problem we want to address is how to convert the timing parameters into the parameters of the corresponding interface specification. For a constraint timing parameter, which is to be transformed into a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon\rangle$, the mapping of the timing parameter to the window $\Delta_{ij}$ is direct. For a delay timing parameter, the mapping is not as straightforward because in our model a delay is modeled using a random variable, and the set of random variables of the probabilistic timed Petri net is characterized by a joint probability density function (pdf). To solve this problem, let us first take a look at possible representations of delay information.

Delays can be modeled in increasing degree of accuracy by: a fixed parameter [111, 117]; a value belonging to an interval [124, 114, 67]; or a random variable with a certain probability density function [49]. The single value case is the easiest to analyze but not very realistic when considering an ensemble of components subject to different conditions (*e.g.*, temperature variations). Cerny and Khordoc [72] have previously identified that timing specifications of components describe more complex timing behavior than AND and OR causality. They have developed timing verification techniques under a different framework based on process algebra which includes what they have called conjunctive constraints to model timing correlation.

Probabilistic models have proved expressive in related domains of gate-level power estimation [99, 100], and gate-level timing analysis [70]. In this dissertation we use both intervals and random variables to describe delays. Furthermore, we have made a connection between the interval representation and the probabilistic representation by means of the concept of projections which is discussed in the rest of this section.

Our probabilistic interface timing verification manipulates the joint probability density function of the random variables to obtain a detailed picture of the temporal behavior of the circuit. Our timing analysis for synthesis is to be used prior to synthesis, when some timing parameters are still missing, thus an interval analysis is more appropriate. In this section we show the relation between the interval representation and the statistical representation of a delay.

For the sake of simplicity, let us first consider the case in which all the random variables are independent. Then the joint probability density function is given by:

$$f_{\tau 1 \ldots \tau M}(\tau_1, \ldots, \tau_M) = f_{\tau 1}(\tau_1) \ldots f_{\tau M}(\tau_M) \qquad \text{(Eq. 2.5.1)}$$

where the set of random variables is $\{\tau_i | i=1, \ldots, M\}$. Thus each random variable $\tau_i$ is characterized by its pdf $f_{\tau i}(\tau_i)$. Chip manufacturers usually have process data that can be used to construct the pdf's that characterize the delays of a component [23]. In the data sheets, manufacturers only provide the boundaries of the values of the pdf's. Thus a delay parameter $[\tau_{min}, \tau_{max}]$ defines an infinite set of pdf's which are non-zero in the interval $[\tau_{min}, \tau_{max}]$ and are zero otherwise (refer to Figure 2.5.5).

The case in which some or all the random variables are not independent is more interesting. In that case, one needs to know the expression for the joint pdf $f_{\tau 1 \ldots \tau M}(\tau_1, \ldots, \tau_M)$.

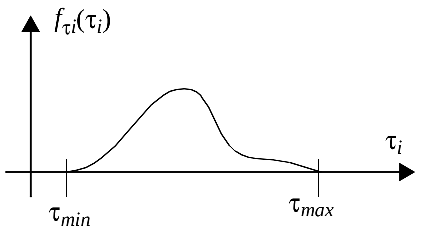


Figure 2.5.5   Probability density function of an independent delay.

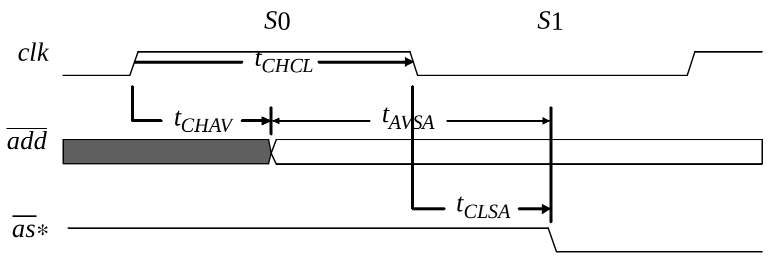


Figure 2.5.6   Timing relationship between *add* and *as*.

To get an intuitive idea of the problem, consider the following example taken from a microprocessor data transfer cycle. The address lines *add* are used to select a particular device. In order to avoid incorrect selection while the address lines are changing, a strobe signal *as* is used to indicate when the *add* lines contain a valid address. The timing relationships between the *add* and $as*$ signals and the clock are shown in Figure 2.5.6. There are four signal transitions of interest: $clk_0$ and $clk_1$, the rising and falling clock transitions respectively, $add+_v$, that indicates the moment when the address lines are guaranteed to be in a valid state, and $as+$, the high-to-low transition on the negative-logic signal $as*$ (refer to Section 2.3.2).

Typical values of the timing parameters are given in Table 2.5.1. There are four timing parameters specified in the table. For instance $t_{CHAV}$ is the duration from the rising edge of the clock (state 0) and the instant when all the address lines are valid. A minimum and a maximum value are given for each timing parameter due to variations in temperature and fabrication process. The manufacturer guarantees that a chip will perform within the minimum and maximum given times.

| Symbol | Timing parameter | Min | Max | Unit |
|---|---|---|---|---|
| $t_{\mathrm{CHAV}}$ | Clock high to Address valid | 0 | 40 | ns |
| $t_{\mathrm{CLSA}}$ | Clock low to *as* asserted | 0 | 40 | ns |
| $t_{\mathrm{AVSA}}$ | Address valid to *as* asserted | 20 | - | ns |
| $t_{\mathrm{CHCL}}$ | Clock high to Clock low | 35 | 45 | ns |

Table 2.5.1. Timing specifications (Motorola MC68030)

Figure 2.5.7 shows a graphical representation of the timing diagram shown in Figure 2.5.6. One can identify two clock transitions, which are indexed by the corresponding states, the instant when the address lines contain a valid address, and the assertion of the address strobe signal. The four timing parameters of Table 2.5.1 label the edges in the graph.

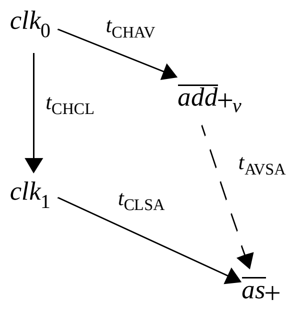

$$\begin{tikzcd}
clk_0 \arrow[r, "t_{\mathrm{CHAV}}"] \arrow[d, "t_{\mathrm{CHCL}}"] & \overline{add}+_v \arrow[d, dashed, "t_{\mathrm{AVSA}}"] \\
clk_1 \arrow[r, "t_{\mathrm{CLSA}}"] & \overline{as}+
\end{tikzcd}$$

Figure 2.5.7 Signal transition graph.

With the exception of $t_{AVSA}$, which is the object of our discussion, the other timing parameters describe causal relationships between the corresponding signal transitions. For instance, the range of $t_{CHCL}$ in Table 2.5.1 specifies that the (high) pulse width of the input clock signal should be between 35ns and 45ns. Timings $t_{CHAV}$ and $t_{CLSA}$ describe the delay from the respective clock transition to $add+_v$ and $as+$.

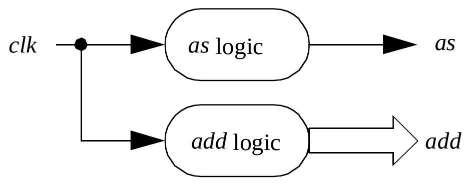


Figure 2.5.8 Generation of the address lines and address strobe signals.

The fourth timing parameter, $t_{AVSA}$, specifies that $add+_v$ always precedes $as+$ by at least 20 ns. One may be tempted to add a causal edge from $add+_v$ to $as+$ (as shown by the dashed edge in Figure 2.5.7). However it turns out that $t_{AVSA}$ is not a propagation delay. To understand this, one needs to refer to Figure 2.5.8 where a possible implementation of the circuit that generates signals *add* and *as* is depicted. Two logic blocks drive *as* and *add* respectively to their appropriate values (taking as inputs the clock and other internal signals which are not shown for the sake of simplicity).

Let us call $d_1$ and $d_2$ the delays from $clk_0$ to $as+$ and $add+_v$ respectively. From the parameters in Table 2.5.1, it can be inferred that $d_1 \in [0, 40]$ and $d_2 \in [35, 85]$ without taking $t_{AVSA}$ into consideration. It is clear that there are some combinations of values for $d_1$ and $d_2$ for which $t_{AVSA}$ is not satisfied. For example if $d_1 = 40$ and $d_2 = 35$. Notice that we have chosen the maximum value for $d_1$ and the minimum value for $d_2$.

However the sources of delay variation (*e.g.* process fabrication tolerances, temperature effects, *etc*.) are *likely* to affect both delays in the same direction, making the above choice of values for the delays improbable. Thus we say that $d_1$ and $d_2$ are not independent of each other but are *correlated*. That is, the probability that $d_1$ take certain value is not independent of the value that $d_2$ takes. Figure 2.5.9 shows a joint probability density function of two variables. Note that for different values of *x*, the range of values that *y* can take varies. A similar behavior is expected between $d_1$ and $d_2$, *i.e.*, the values of the delays are not independent of each other.

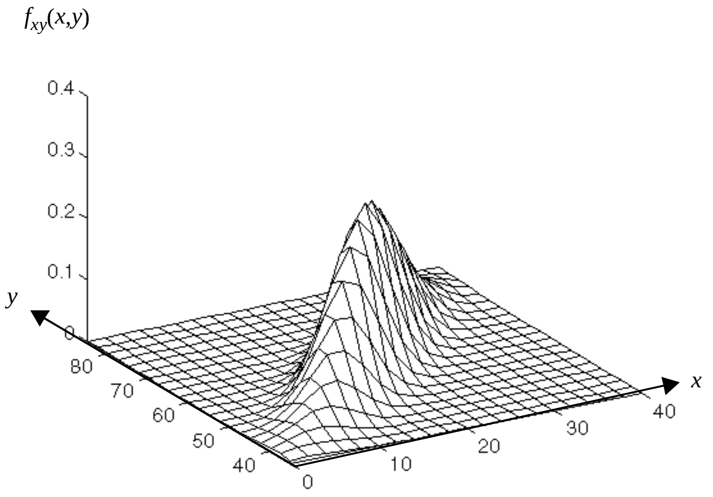


Figure 2.5.9   Joint probability density function.

Once the joint probability density function is known, it is possible to perform a timing analysis of a design [48]. However if complete information about the joint probability density function is not available, a simplification is possible if the boundary values of the function are known. The general idea is to look just at the possible values that the delays can take, disregarding the probability information. In that case a worst-case analysis can be performed. The effect of the specialization can be seen as the projection of the probability density function $f_{x1\ldots xn}(x_1,\ldots,x_n)$ into the hyperplane $x_1\ldots x_n$.

In Figure 2.5.10 the shaded area describes the projection of the probability density function $f_{d1,d2}(d_1,d_2)$, which better expresses the intention of the timing information provided by the manufacturer (c.f. Table 2.5.1), namely that the values that the delays $d_1$ and $d_2$ can take are such that always the transition of the *add* lines to valid precede the assertion of *as* by at least 20 ns (*i.e.*, $d_1 - d_2 \geq 20$ ns). If $d_1$ and $d_2$ were independent, the projection would be the rectangle whose perimeter is depicted by a dashed line. Note that the boundary of the projection in Figure 2.5.10 can be described by linear expressions on $d_1$ and $d_2$. Although arbitrary boundaries can be used (*e.g.*, the dark area shown in Figure 2.5.10), linear boundaries have a clear computational advantage, and lend themselves to a concise description.

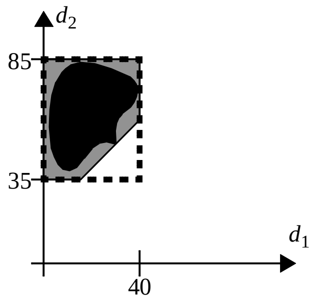


Figure 2.5.10 The projection (dark region) and a linear projection (gray area) of a probability density function for delays $d_1$ and $d_2$.

Now we proceed to formalize the concept of projection of a joint pdf.

**Definition 2.5.4.-** The projection of a joint probability density function $f_{\tau 1 \ldots \tau M}(\tau_1, \ldots, \tau_M)$ is the maximal set $\{(\tau_1, \ldots, \tau_M) \mid f_{\tau 1 \ldots \tau M}(\tau_1, \ldots, \tau_M) > 0\}$ (with respect to inclusion).

Geometrically, the projection of a joint pdf can be represented as a region in the *M*-dimensional euclidean space $R^M$. As mentioned before, it is sometimes convenient to approximate the actual projection by another region which contains the original region but whose representation is easier. Convex regions have the property that any two points in the region can be connected by a line that is included in the region [109]. In this dissertation we only consider a particular type of convex approximations.

**Definition 2.5.5.-** A hyperplane in $R^M$ is the set of points $(\tau_1, \ldots, \tau_M) \in R^M$ such that $\alpha_1 \cdot \tau_1 + \ldots + \alpha_M \cdot \tau_M = \beta$, for the set of given constants $\alpha_1, \ldots \alpha_M, \beta \in R$.

A half-space is the portion of $R^M$ lying on one side of a hyperplane. A closed half-space is a half-space including the hyperplane.

**Definition 2.5.6.-** A polyhedral set, or in short polyhedron, in $R^M$ is the intersection of a finite set of closed half-spaces.

**Definition 2.5.7.-** A *linear projection* of a joint pdf $f_{\tau 1 \ldots \tau M}(\tau_1, \ldots, \tau_M)$ is a polyhedral set *PS* that includes the projection of $f_{\tau 1 \ldots \tau M}(\tau_1, \ldots, \tau_M)$.

In the sequel a polyhedral set shall be referred simply as a polyhedron. It is well known that a polyhedral set is a convex set that can be described by a set of inequalities on the variables $\tau_i$ [104].

Now we address the problem of extracting a linear projection from the correlation edges of an interface specification.

**Definition 2.5.8.-** Given two transitions $t_0, t_n \in T$ of a probabilistic timed Petri net $N = \langle P, T, F, M_0, \Gamma \rangle$, the sequence $t_0 t_1 \ldots t_n$ is called a *directed path* if all the transitions are distinct and for any $i$, $i = 1, \ldots n$, there exists a $p_j$ such that $(t_{i-1}, p_j) \in F$ and $(p_j, t_i) \in F$, for $i = 1 \ldots n$.

**Definition 2.5.9.-** A directed path is called *simple* if for each pair of adjacent transitions $t_{i-1}, t_i$ in the path, there exists only one place $p_i$ such that $(t_{i-1}, p_i) \in F$. The *delay* of a simple directed path is given by $d(t_0, t_n) = \sum_{i=1}^{n} \tau_i$, where $\tau_i$ is the random variable associated with place $p_i$.

**Definition 2.5.10.-** Given three transitions $t_0$, $t_1$, and $t_2 \in T$ of a probabilistic timed Petri net $N = \langle P, T, F, M_0, \Gamma \rangle$, transition $t_0$ is called a *simple fork transition* for $t_1$ and $t_2$ if there exists a simple directed path from $t_0$ to $t_1$ and there exists a simple directed path from $t_0$ to $t_2$. The time separation from $t_1$ to $t_2$, denoted $\tau_{t2} - \tau_{t1}$, is given by:

$$d(t_0, t_2) - d(t_0, t_1).$$

**Definition 2.5.11.-** A timing parameter $\pi_{ij}$ associated with two transitions $t_i$ and $t_j$ of a net $N = \langle P, T, F, M_0, \Gamma\rangle$ is the tuple $\langle t_i, t_j, p_{ij}, \tau_{ij}\rangle$ where $p_{ij}$ is a place of the net such that $(t_i, p_{ij}) \in F$ and $(p_{ij}, t_j) \in F$, and $\tau_{ij} \in \Im+$ is a non-empty non-negative compact real interval ($\Im+$ is the set of compact non-negative real intervals [112]).

A timing parameter $\pi_{ij}$ represents a causal delay from transition $t_i$ to transition $t_j$, modeled by the random variable associated with place $p_{ij}$ whose unique input transition is $t_i$ and whose unique output transition is $t_j$.

**Definition 2.5.12.-** A correlation rule $\phi_{ij}$ associated with two transitions $t_i$ and $t_j$ of a net $N = \langle P, T, F, M_0, \Gamma\rangle$ is the triple $= \langle t_i, t_j, \rho_{ij}\rangle$ where $\rho_{ij} \in \Im$ is a non-empty compact real interval ($\Im$ is the set of compact non-negative real intervals [112]).

A correlation rule describes a time restriction on the time separation $\tau_{t2} - \tau_{t1}$ from $t_1$ to $t_2$, such that $\tau_{t2} - \tau_{t1} \in \rho_{ij}$. A correlation rule is represented graphically as an edge from $t_1$ to $t_2$. To distinguish a correlation edge from delay edges and constraint edges, we use the convention of drawing correlation edges as dashed lines (refer to Figure 2.5.7).

Correlation rules and delay timing parameters can be used to construct a projection of a joint pdf with the interpretation given above. The collection of timing parameters and correlation rules of a specification describes a region of possible values for the delays $\tau$. The following procedure simply describes how to construct such a region.

**Procedure 2.5.1.-** Construction of the projection of a joint pdf of a probabilistic timed Petri net $N = \langle P, T, F, M_0, \Gamma\rangle$ given a set of delay timing parameters $\{\pi_{ij}\}$ and a set of correlation rules $\{\rho_{ij}\}$.

1. For every timing parameter $\pi_{ij} = \langle t_i, t_j, p_{ij}, \tau_{ij}\rangle$ such that $\tau_{ij} = [\tau_{ij,min}, \tau_{ij,max}]$, a pair of linear inequalities $\tau_{ij,min} \le \tau_{ij} \le \tau_{ij,max}$ are generated.

2. For every correlation rule $\phi_{ij} = \langle t_i, t_j, \rho_{ij}\rangle$ such that $\rho_{ij} = [\rho_{ij,min}, \rho_{ij,max}]$, a pair of linear inequalities $\rho_{ij,\ min} \le \tau_{t_i} - \tau_{t_j} \le \rho_{ij,\ max}$.

In this dissertation we further restrict that a correlation edge be placed from transition $t_1$ to transition $t_2$ only if there exists a simple fork transition $t_0$ for $t_1$ and $t_2$. In this case the expressions of the form $\rho_{ij,\ min} \le \tau_{t_i} - \tau_{t_j} \le \rho_{ij,\ max}$ can be computed in a simple way. The following Lemma states that also the region of possible values of the delays $\delta$ according to the specification is a polyhedral set. Thus such region can be thought of as a linear projection of the pdf $f_{\tau 1\ \ldots\ \tau M}(\tau_1, \ldots, \tau_M)$.

**Lemma 2.5.2.-** Let $N = \langle P, T, F, M_0, \Gamma\rangle$ be a probabilistic timed Petri net and let $\Pi = \{\pi_{ij}\}$ be a set of timing parameters and $\Phi = \{\phi_{ij}\}$ be a set of correlation rules. If for each $\rho_{ij}$ from transition $t_i$ to transition $t_j$ there exists a simple transition fork for $t_i$ and $t_j$, then the region defined by both $\Pi$ and $\Phi$ is a polyhedral set.

**Proof.-** Step 1 of Procedure 2.5.1 defines an $M$-dimension cube in $R^M$ described by $\tau_{ij,min} \le \tau_{ij} \le \tau_{ij,max}$, which is clearly a polyhedral set. We now show that the addition of correlation inequalities $\rho_{ij,\ min} \le \tau_{t_i} - \tau_{t_j} \le \rho_{ij,\ max}$ generates a polyhedral set. Consider a correlation rule $\phi_{ij} = \langle t_i, t_j, \rho_{ij}\rangle$ from $t_i$ to $t_j$. If there is a simple fork transition for $t_i$ and $t_j$ then, according to Definition 2.5.10, $\tau_{t_i} - \tau_{t_j}$ can be expressed as the difference $\sum_i \tau_i - \sum_j \tau_j$. Thus a constraint rule specifies:

$$\rho_{ij,\ min} \le \sum_i \tau_i - \sum_j \tau_j \le \rho_{ij,\ max} \qquad \text{(Eq. 2.5.2)}$$

Equation 2.5.2 defines a region in $R^M$ between two hyperplanes. Therefore the region constructed by Procedure 2.5.1, provided that each pair of transitions related by a correlation rule has a simple fork transition, is thus described by a linear set of inequalities. $\square$

Notice that a correlation rule is not part of the flow relation of the Petri net. However it is often convenient to show a correlation rule in an interface specification at the STG level as a different type of edge between two signal transitions. In the sequel we draw a correlation rule as a correlation edge between two signal transitions of the interface specification as shown in Figure 2.5.4 (the correlation edge is labeled with $t_{AVSA}$).

In the following section we show some examples of interface specifications of commercial components which contain time correlation information.

### 2.5.5 Examples of interface specifications

From the previous sub-section, an interface specification of a component consists of two parts: a signal transition graph $\Sigma$ and a set of constraint rules $C$, which is denoted compactly as $\Psi = \Sigma + C$. There is a graphical representation for both STG's and constraint rules. We shall represent in this dissertation interface specifications in graphical form for its more intuitive grasp. However it should be clear that a textual description, which is more convenient for large specifications, is also possible.

Although the fundamentals presented here can be applied to different areas, the examples we use throughout this work are from microprocessor-based systems. In Chapter 3 we shall discuss the interface design problem in the context of microprocessor-based systems. Microprocessor components can be classified into processors (*i.e.*, CPU's, DSP's), memory devices (*e.g.*, RAM, ROM), and I/O devices (*e.g.*, parallel ports, serial ports). Systems are built around standard buses to facilitate the integration of system components.

The interface behavior of a component describes the protocols followed by the ports of the component to be able to communicate with other components. Informally a protocol is a predefined series of events. Events are encoded in hardware components as signal transitions. Interface protocols can be described by interface specifications. As mentioned before, an interface specification has two parts: the description of the internal operation of the component and a set of restrictions on the environment of the component.

The interface behavior of microprocessor components can be quite involved. It is convenient to separate the overall behavior into sub-behaviors. For example a CPU may transfer data to and from other devices, may accept external hardware interrupts, may arbitrate the use of its data transfer lines by other active devices (called masters, refer to Section 3.2.1), or may support cache coherency mechanisms. Each of these *capabilities* is specified by a protocol. The overall component behavior is given by all its capabilities. At this moment we have concentrated on studying protocols of capabilities. An area of future research is to develop extensions to our approach that work with the overall behavior.

Component behavior are usually given in the form of timing diagrams in the manufacturer's data sheets. We prefer to use a formal description, interface specifications, for it allows us to develop formal methods for timing analysis and verification. The internal operation of the single capability protocols that we have studied can be easily described using AOC STG's. In this section we illustrate the read cycles (*i.e.*, the protocol of the read capability) of the Texas Instrument SM64C16 SRAM device, and the Analog Devices SHARC DSP. We shall use other interface protocols in subsequent chapters of this dissertation.

#### 2.5.5.1 SRAM read cycle

The Texas Instrument SM64C16 SRAM device is a static random access memory device organized as 4,096 words by 4 bits. Although this is an early device, the new static RAM chips follow the basic interface protocol of the SM64C16, only with faster timing parameters. The interface protocol of the SM64C16 is representative of other static RAM's with different configurations. There are four group of signals: data lines (4-bit bidirectional ports that are represented by a 4-bit input port $\underline{D}$ and 4-bit output port $\overline{Q}$), address lines (12-bit input port $\underline{A}$), and control lines (1-bit input ports $\underline{E}*$, and $\underline{W}*$), and power lines (*GND* and *VCC*). The '$*$' after a name of a port indicates that it uses negative logic (refer to Section 2.3.2). The power lines do not play any role in the protocol description. $\underline{W}*$ is the write signal, which is '1' if the cycle is read, and '0' if the cycle is write. $\underline{E}*$ is the enable signal, which is '0' if a read or a write cycle is taking place, or '1' if the device is in standby mode.

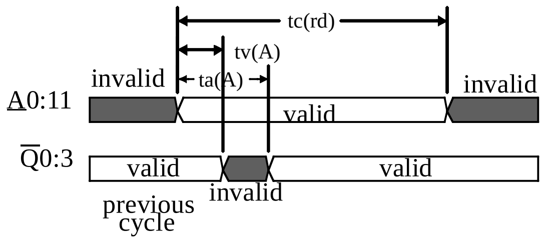


Figure 2.5.11 Timing diagram of an SRAM read cycle from address.

The read protocol or *cycle* of the SM64C16 device is described in the manufacturer's data sheets [120] by two timing diagrams, reproduced in Figures 2.5.11 and 2.5.12. The first timing diagram (refer to Figure 2.5.11) shows the sequence of signal transitions on the data lines (the output port) caused by signal transitions on the address lines (the input port) when the control inputs are: $\underline{W}*$ is '1', or negated, and $\underline{E}*$ is '0', or asserted. Both ports alternate values between a valid state and an invalid state. There are three timing parameters in the timing diagram: $t_{c(rd)}$, $t_{v(A)}$, and $t_{a(A)}$. The first parameter, $t_{c(rd)}$, is a timing constraint that restricts the width of the valid state of the address lines. The other two parameters, $t_{v(A)}$ and $t_{a(A)}$, are propagation delays of a signal transition in the data lines caused by a signal transition in the address lines. In the data sheets of the SM64C16, propagation delays and timing constraints are differentiated (they are called switching characteristics and timing constraints respectively). However in other data sheets (*e.g.* [95]) there is no indication of the type of timing information and some ingenuity from the designer is required to identify the timing relationships.

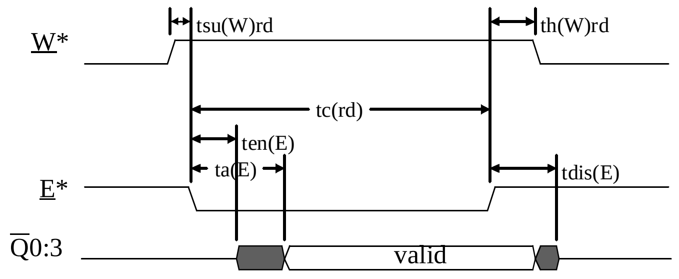


Figure 2.5.12 Timing diagram of an SRAM read cycle from enable.

The second timing diagram (refer to Figure 2.5.12) describes the sequence of signal transitions of the data lines as caused only by the control signals. The address lines must be valid prior to or simultaneously with the high-to-low transition of $\underline{E}*$ (*i.e.*, $\underline{E}*+$, a transition from negated to asserted). The timing constraints are: $t_{c(rd)}$, a restriction on the pulse width of $\underline{E}*$; and $t_{su(W)rd}$ and $t_{h(W)rd}$, a set-up and hold times of $\underline{W}*$ with respect to $\underline{E}*$. The propagation delays are: $t_{en(E)}$, the propagation delay from $\underline{E}*+$ to $Q\uparrow$; $t_{a(E)}$, the propagation delay from $\underline{E}*+$ to $Q+_v$; and $t_{dis(E)}$, the propagation delay from $\underline{E}*-$ to $Q\downarrow$ (for the special signal transition symbols, refer to Table 2.3.1).

The value ranges of the timing parameters are given in Table 2.5.2. A constraint parameter $\Delta_{ij}$ can be transformed into the constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon \rangle$. A delay parameter is transformed to a place of a probabilistic timed Petri net. Such a place is associated with a random variable. The set of random variables of the Petri net are characterized by a joint probability density function (pdf). If all the random variables are assumed to be independent, each random variable is characterized by its pdf. Chip manufacturers count with data that can be used to construct either the joint pdf, or the independent pdf's [23]. In the

| **timing parameter** | **range (ns)** |
|---|---|
| $t_{c(rd)}$ (C) | $[25, \infty)$ |
| $t_{v(A)}$ (D) | $[0, \infty)$ |
| $t_{a(A)}$ (D) | $[0, 25]$ |
| $t_{su(W)rd}$ (C) | $[0, \infty)$ |
| $t_{h(W)rd}$ (C) | $[0, \infty)$ |
| $t_{en(E)}$ (D) | $[5, \infty)$ |
| $t_{a(E)}$ (D) | $[0, 25]$ |
| $t_{dis(E)}$ (D) | $[0, 15]$ |

Table 2.5.2. Timing parameters for the 25 ns version of the SRAM device (C: timing constraint; D: propagation delay).

data sheets, manufacturers only provide the boundaries of the values of the pdf's. Thus a delay parameter $[t_{min}, t_{max}]$ defines an infinite set of pdf's which are non-zero in the interval $[t_{min}, t_{max}]$ and are zero otherwise.

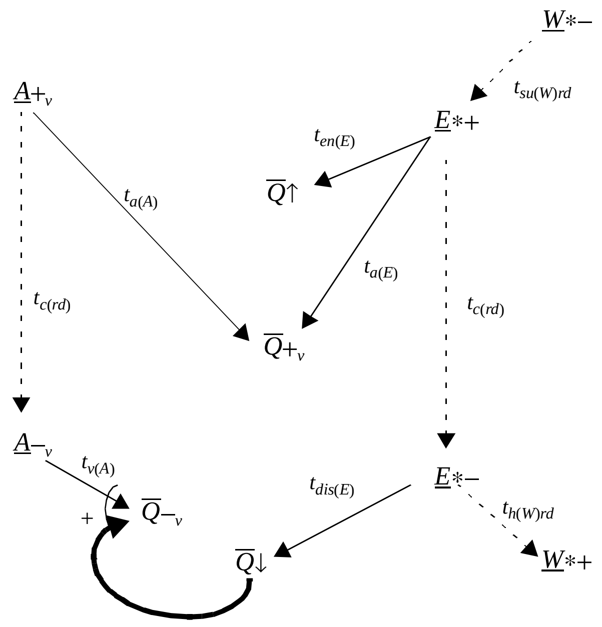


Figure 2.5.13 Partial interface specification of the SRAM read protocol.

One would like to combine the information of the two timing diagrams into a single specification. This is accomplished by the interface specification shown in Figure 2.5.13. As before, solid edges describe delays, and dotted edges describe timing constraints. With the exception of the thick edge, which requires further explanation, the other edges can be easily identified from the timing diagrams in Figures 2.5.11 and 2.5.12. At the end of the read cycle there are two concurrent actions: the data lines become invalid after the address lines become invalid; and the data lines are tri-stated after the enable line is negated. Then the following three scenarios are possible: the data lines first become invalid and then tri-stated, or the data lines become at the same time invalid and tri-stated, or the data lines become tri-stated (forcing the data lines to invalid too). OR causality is invaluable in this case to model the three aforementioned scenarios by stating that the data lines become invalid either after the address lines become invalid or after the data lines become tri-stated due to the negation of the enable signal. If the pdf characterizing the random variable associated with the thick edge/place is the Dirac's impulse function, then the place makes the token visible immediately as desired.

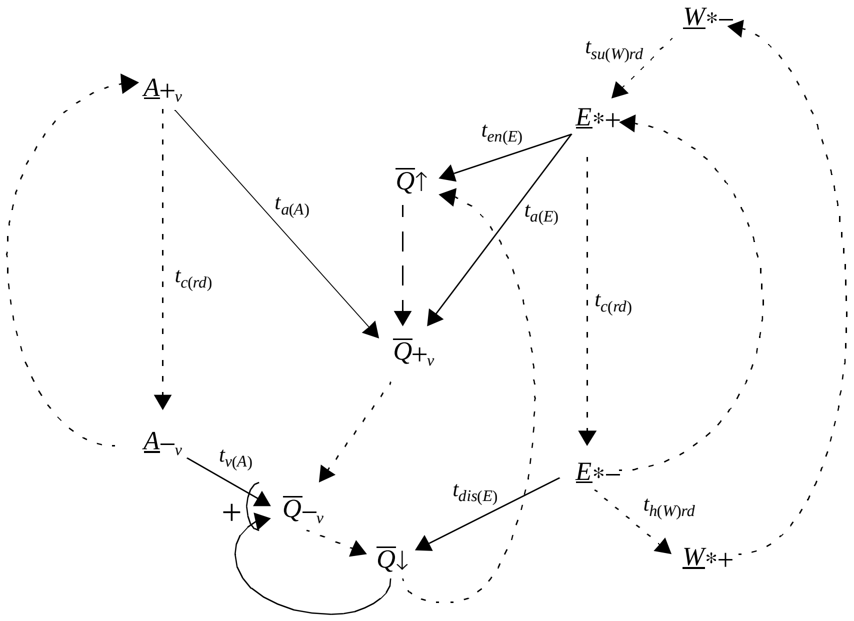


Figure 2.5.14 Interface specification of the SRAM read cycle.

In the timing diagrams some information is implicit, namely the sequencing of states of a signal. That is, a port cannot be in two states simultaneously and thus implements a sequential process. Additional edges that guarantee port sequencing complete the interface specification of the SRAM device, which is shown in Figure 2.5.14. Notice that the delay edges (solid lines) connect an input transition to an output transition (except for the special OR causality edge from $\overline{Q}\downarrow$ to $\overline{Q}-_v$). The added edges to transitions of input ports $\underline{A}$, $\underline{E}*$ and $\underline{W}*$ are clearly timing constraints (with associated interval $\Delta = [0, \infty)$ which indicates a precedence constraint) because the ports are manipulated by the environment.

However the four added edges to transitions of $\overline{Q}$ require further explanation. The edge from $\overline{Q}+_v$ to $\overline{Q}-_v$ is a precedence constraint that checks for an inconsistent specification that assigns value ranges for $t_{c(rd)}$, $t_{v(A)}$, and $t_{a(A)}$ such that transition $\overline{Q}-_v$ occurs after $\overline{Q}+_v$. Using the same argument, one can see that the edge from $\overline{Q}\downarrow$ to $\overline{Q}\uparrow$ must be a precedence constraint. The edge from $\overline{Q}-_v$ to $\overline{Q}\downarrow$ is actually not necessary because the OR causality edge from $\overline{Q}\downarrow$ to $\overline{Q}-_v$ ensures that $\overline{Q}-_v$ will occur not later than $\overline{Q}\downarrow$.

Finally the edge from $Q\uparrow$ to $Q+_v$ is a correlation edge shown as a dashed line (refer to Section 2.5.4). This correlation edge specifies that the value of delays $t_{en(E)}$ and $t_{a(E)}$ are not independent, so that delay $t_{en(E)}$ is always smaller than $t_{a(E)}$. From Table 2.5.2, and our previous discussion on transforming delay parameters to pdf's of random variables, one can see that while $t_{a(E)}$ is a positive number not exceeding 25 ns, $t_{en(E)}$ can be take any value greater than or equal to 5 ns. The correlation edge restricts the values of both $t_{en(E)}$ and $t_{a(E)}$ so that sequentiality of the data lines is always observed. Let us call $\tau_1$ and $\tau_2$ the random variables associated with the delay places labeled respectively with timing parameters $t_{en(E)}$ and $t_{a(E)}$. The linear projection of the joint pdf $f_{\tau 1 \tau 2}(\tau_1, \tau_2)$ obtained according to Procedure 2.5.1 is described by the following set of inequalities:

$5 \le \tau_1 < \infty$
$0 \le \tau_2 \le 25$
$0 \le \tau_2 - \tau_1$

and it is shown in Figure 2.5.15.

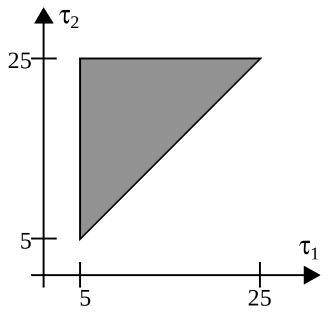


Figure 2.5.15 Projection of the probability density function $f_{\tau 1 \tau 2}(\tau_1, \tau_2)$.

### 2.5.5.2 DSP read cycle

The Analog Devices SHARC (Super Harvard ARchitecture Computer) DSP is a highly parallel high-performance processor. In this section we model the SHARC's read cycle.

The read cycle is described in the SHARC's data sheet [4] by the timing diagram reproduced in Figure 2.5.16. Four groups of ports are involved in the read cycle: the 1-bit <u>*CK*</u> clock input line, the 32-bit input <u>*ADD*</u> address lines, the 48-bit bi-directional *DAT* data lines (as output lines), and the control lines *RD* and <u>*ACK*</u>.

The value ranges of the timing parameters are shown in Table 2.5.2. The timing parameters are classified in the data sheet as switching characteristics and timing requirements. Timing requirements correspond directly to our timing constraints; however switching characteristics specify not only propagation delays but also time correlation data, as explained below.

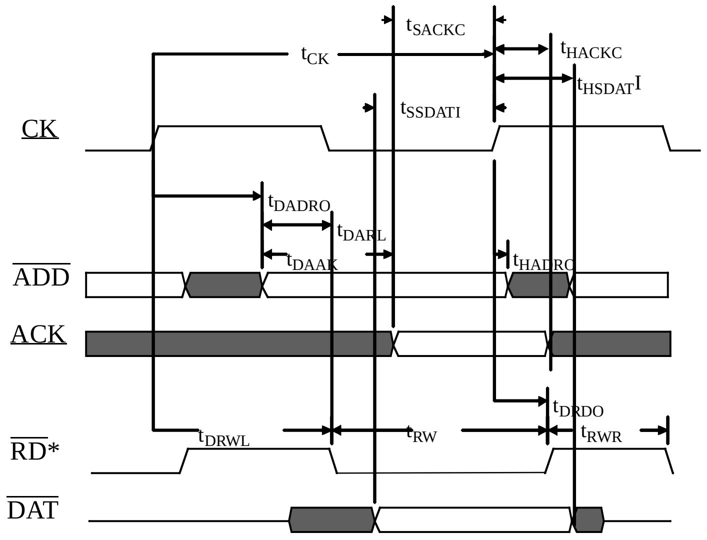


Figure 2.5.16 Timing diagram of the SHARC read cycle.

| **timing parameter** | **range (ns)** |
|---|---|
| $t_{SACKC}$ (C) | [6, ∞) |
| $t_{HACKC}$ (C) | [-1, ∞) |
| $t_{SSDATI}$ (C) | [3, ∞) |
| $t_{HSDATI}$ (C) | [2, ∞) |
| $t_{DAAK}$ (C) | (–∞, 10] |
| $t_{DADRO}$ (D) | [0, 8] |
| $t_{HADRO}$ (D) | [0, ∞) |
| $t_{DRWL}$ (D) | [8, 13] |
| $t_{DRDO}$ (D) | [1, 4] |
| $t_{DARL}$ (R) | [2, ∞) |
| $t_{RW}$ (R) | [13, ∞) |
| $t_{RWR}$ (R) | [6, ∞) |

Table 2.5.3. Timing parameters for the 40 MHz version of the SHARC DSP (C: timing constraint; D: propagation delay; R: correlation data).

The interface specification of the SHARC read cycle is shown in 2.5.17. Once the type of timing information has been determined, the construction of the interface specification from the timing diagram is straightforward. Each timing parameter of the timing diagram is represented by an edge in the specification of the corresponding type (delay, timing constraint or correlation). Sequentiality edges have been added as discussed before. Notice that $CK_1$+ corresponds to $CK_0$+ of the next cycle.

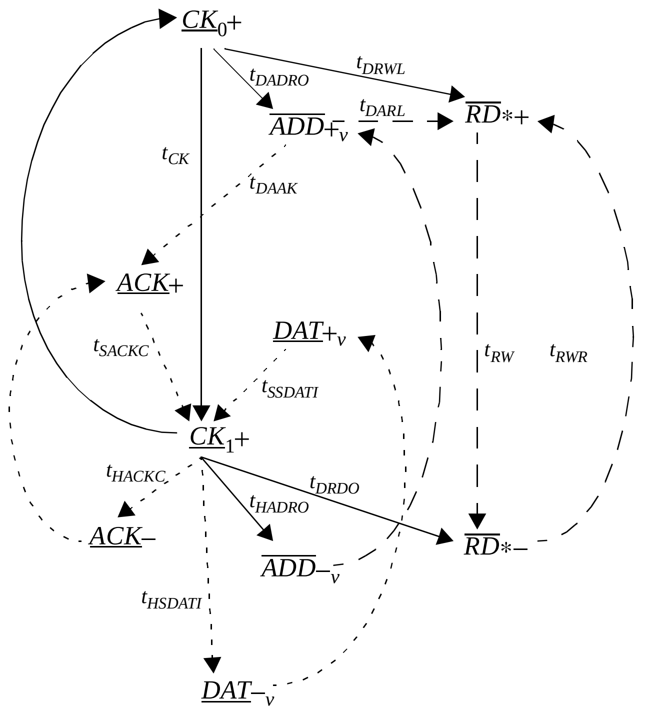


Figure 2.5.17 Interface specification of the SHARC read cycle.

Notice that the lower bound of the timing constraint $t_{DAAK}$ is a negative value. Specification formalisms in which timing constraints are modeled with places of a Petri net (*e.g.,* [124] and [114]) will have problems handling “negative-valued” timing constraints, because tokens in places behave causally.

We now discuss briefly the correlation edges. Consider the edge $t_{DARL}$. As discussed in Section 2.5.4, this edge does not describe a causal (delay) timing relationship from transition $ADD+_v$ to transition $RD+$, which will imply some circuitry that generates $RD+$ using $ADD+_v$ as one of its inputs. The manufacturer can guarantee that $RD+$ always follows $ADD+_v$ because delays $t_{DADRO}$ and $t_{DRWL}$ are not independent (due for example to the proximity of the circuits that generate $RD+$ and $ADD+_v$ in the chip wafer, so that they are affected similarly by fabrication process variations and temperature changes). A similar argument can be applied to understand why the edges $t_{RW}$, $t_{RWR}$, and (the added sequential edge) $t_{ADD\_SEQ}$ specify time correlation data.

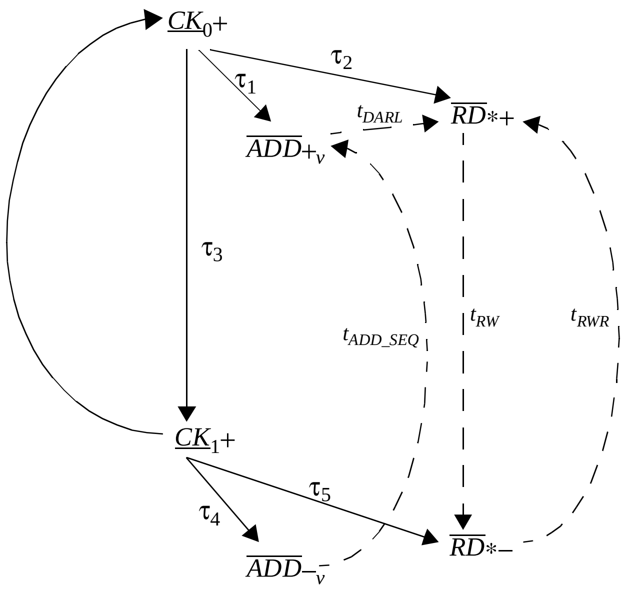


Figure 2.5.18 Construction of a linear projection of the joint pdf $f_{\tau 1\tau 2\tau 3\tau 4\tau 5}(\tau_1, \tau_2, \tau_3, \tau_4, \tau_5)$.

Notice that the associated transitions to each correlation edge have a simple fork transition (refer to Definition 2.5.10). The random variables $\tau_i$ associated with the delay edges of the interface specification are shown in Figure 2.5.18. The expressions generated by step 2 of Procedure 2.5.1 are:

$$\begin{aligned}
&\tau_2 - \tau_1 \subseteq t_{DARL} \\
&\tau_3 + \tau_5 - \tau_2 \subseteq t_{RW} \\
&\tau_2 - \tau_5 \subseteq t_{RWR} \\
&\tau_1 - \tau_4 \subseteq t_{ADD\_SEQ}
\end{aligned}$$

The linear projection of the joint pdf $f_{\tau 1 \tau 2 \tau 3 \tau 4 \tau 5}(\tau_1, \tau_2, \tau_3, \tau_4, \tau_5)$ is a region in $R^5$, so that we do not attempt to visualize it. However, standard techniques can be used for its manipulation [109].

We shall come back to the interface specification of the SRAM and the DSP read cycles in Chapter 4 where we shall use linear projections to construct joint pdf's given some assumptions.

## 2.6 Summary

In this chapter we have presented interface specifications, our formal framework to describe the behavior of hardware components used to build up a system. An interface specification consist of two parts: a description of the component's internal operation, and a restriction on the component's environment. The first part can be described using an interpreted Petri net called timed signal transition graph. The second part can be specified as a set of constraint rules.

In the following chapter we shall address the interconnection of components. There we shall show that the behavioral properties of the system can be inferred by studying a "merged" graph which consists of the interface specifications of the components that comprise the system. Furthermore, if the components cannot be connected directly and glue logic is necessary, we show that the system consisting of the components plus the interface logic can be described as a graph which contains the interface specifications of the interconnected components plus additional edges that represent the interface logic.

# Chapter 3

# Timing and the Interface Design

## 3.1 Introduction

As hardware systems become more complex, the design process shows a trend towards the use of CAD tools that automatically carry out the clerical, time-consuming, and error prone synthesis and verification tasks. Timing is a critical aspect of the design which, except in toy systems, cannot be separated from other parts of the design such as design specification, synthesis, and verification. This is particularly true for high-performance systems, for which achieving the maximum possible throughput is of paramount importance.

In the previous chapter we presented a formal framework to specify the interface behavior of components, that we called interface specifications. Those components are independent modules that can be used to build up more complex systems. The main goals of this chapter are: to show how interface specifications can be used to facilitate the description and validation of systems composed of an aggregation of components; and to provide the foundation of our verification and analysis techniques that will be discussed in Chapters 4 and 5.

In the following section, we discuss the interface design problem, that arises when a hardware system is to be constructed using predefined modules (*e.g.*, off-the-shelf microprocessor components, parameterized libraries, special function cores, *etc.*). Our main contribution is the formulation of the interface design problem as the “merging” of interface specifications. The rest of the chapter discusses the concept of time-consistency as applied to the interface design problem.

## 3.2 Interface design problem

Reuse is a powerful concept that makes it possible to construct more complex systems out of pre-existing ones. In this dissertation we use microprocessor-based systems as one example, although the underlying ideas are applicable to other areas where a system is composed of well-defined sub-components or modules. An essential ingredient that empowers reuse is a clean component interface.

In the case of microprocessor-based systems, a system is composed of off-the-shelf components that can be classified as processors, memory devices, and input/output (or I/O) devices. Single-processor systems are rapidly being superseded by multi-processor systems, with a complex memory hierarchy and high-throughput I/O, to satisfy the demands for increasing power. In the previous chapter, we introduced a formal specification of interface behavior of digital hardware components called interface specifications. In this section we use interface specifications to specify the temporal behavior of complete systems composed of off-the-shelf components and interface, or glue, logic.

### 3.2.1 System integration and interface design

System integration is the process of building systems out of pre-existing components. In microprocessor-based systems, it may not be possible to connect the components directly together. For instance, a memory device may expect a positive logic signal to initiate a read, while the processor may use a negative logic signal. This type of incompatibility is at the physical (electrical in this case) level, and signal conditioning (*e.g.*, an inverter) is involved. A more serious incompatibility arises when a processor initiates a read differently from what the memory device expects. We say that the processor follows a different protocol from the one used by the memory. Interface logic is a circuit that must be placed between microprocessor components to resolve incompatibilities so that the components can exchange information. In this section we introduce some terminology and provide an example to illustrate our ideas.

Microprocessor components are complex modules themselves, and they may perform different operations. For example, a processor typically is capable of transferring data with other components, of accepting external hardware interrupts, of allowing other processors to take over its data transfer lines, *etc.* Such operations are called *capabilities*, *e.g.*, the data transfer capability, the bus arbitration capability, *etc.* The logic and temporal behavior of a capability can be described by a *protocol*. Informally a protocol is a series of *actions*, or atomic operations, that are used to perform a capability. Components are connected to other components through input and output ports (refer to Section 2.3.2). The actions of the protocol are encoded as *signal transitions*, *i.e.* changes on the values of the ports.

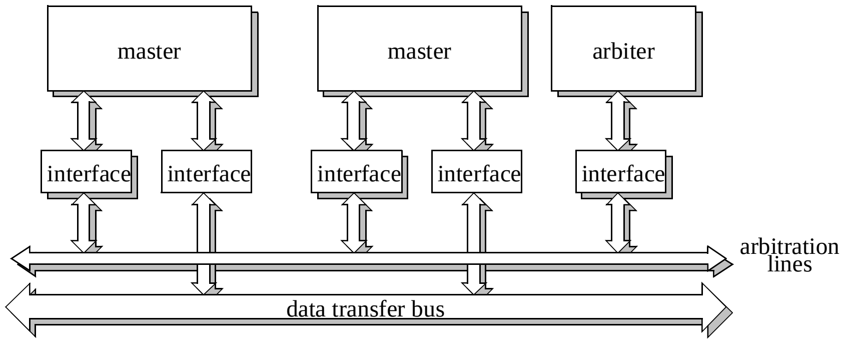


Figure 3.2.1 Multi-master system.

Consider for example a microprocessor system which consists of multiple masters shown in Figure 3.2.1. Among its several capabilities (such as transferring data, accepting hardware interrupts, *etc.*), a master is able to initiate a data transfer via a shared resource: the data transfer bus. This capability is called bus arbitration. The bus arbitration lines are used to guarantee that at most one master requesting the bus takes over the DTB at any given time. A bus arbitration protocol is defined by the standard bus, and each of the masters may use a different bus arbitration protocol. The interface logic placed between a master and the standard bus generates the input actions to both components.

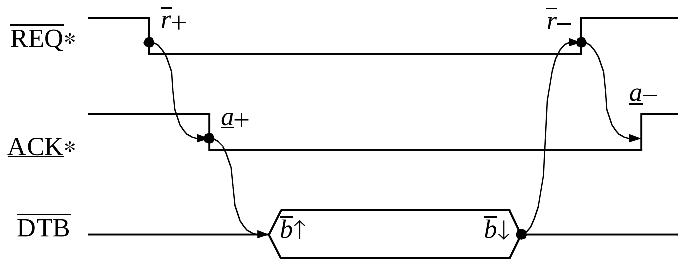


Figure 3.2.2 Master bus arbitration protocol.

Protocols are usually given in the component data sheets in the form of timing diagrams. The timing diagram of a bus arbitration protocol used by a typical master is shown in Figure 3.2.2. According to our notation (refer to Section 2.3.2) in Figure 3.2.2 input signals are underlined and output signals are overlined. The $\overline{\text{REQ}}*$/$\underline{\text{ACK}}*$ signals are control signals which can be asserted and negated, while the $\overline{\text{DTB}}$ is a group of lines that are initially in a high-impedance state, and are driven by the master (when the master takes over the bus).

The bus arbitration protocol can be described by a sequence of actions. The master signals to the arbiter that it wants to use the bus by asserting the output $\overline{\text{REQ}}*$ signal (a '∗' suffix identifies a negative logic signal), and it waits for $\underline{\text{ACK}}*$ to become asserted before it can take over the bus. When the master ends its transaction, it releases $\overline{\text{REQ}}*$. The arbiter then releases $\underline{\text{ACK}}*$ so that another arbitration cycle can take place.

An interface specification embodies the previous protocol description with a precise meaning. The actions of the protocol correspond to signal transitions. For clarity, we use short names to describe the various signal transitions: $\overline{req}+$ ($\overline{req}-$) indicates the change

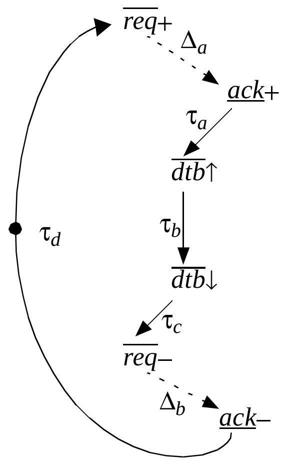


Figure 3.2.3 Interface specification of a bus arbitration protocol.

from asserted to negated (negated to asserted) of the $\overline{\text{REQ}}$* signal; similarly $\underline{ack}$+ ($\underline{ack}$–) indicates the change from asserted to negated (negated to asserted) of the $\underline{\text{ACK}}$* signal; $\overline{dtb}\uparrow$ ($\overline{dtb}\downarrow$) indicates the change from tri-stated to driven (driven to tri-stated) of the $\overline{\text{DTB}}$ lines. The interface specification shown in Figure 3.2.3a describes the bus arbitration protocol. From Section 2.5.3, we know that an interface specification consists of two parts: the internal operation of the component (delays), described by a timed signal transition graph, and a restriction on the behavior of the environment (timing constraints), given as a set of constraint rules. Delays and constraints are graphically represented as solid and dotted edges respectively (correlation edges can also appear in an interface specification as discussed in Section 2.5.4). Each delay edge is associated with a random variable $\tau_i$, and each constraint is labeled by a timing interval $\Delta_i$. For example, in Figure 3.2.3, the random variable $\tau_a$ represents the internal propagation delay in the master from the receiving of the bus grant ($\underline{ack}$+) to the driving of the $\overline{\text{DTB}}$ lines; and the interval $\Delta_a$ specifies that a grant ($\underline{ack}$+) is expected to occur some time after the master issues a request ($\overline{req}$+).

Notice that an interface specification, as discussed in Section 2.5.3, combines two types of information, a specification of the internal operation of the component, given by a timed signal transition graph (with underlying Petri net), and a restriction on the behavior of the environment of the component, given by a set of constraint rules. The underlying net is not connected, and thus, tokens cannot propagate through the net. The main objective of this section and the following section is to introduce the concept of *complete graph*, which informally is a signal transition graph that describes the "complete" operation of a system composed of two or more components, and thus there is an additional requirement that the graph be connected. Please the reader be aware that our usage of complete is different from the concept of complete graphs studied in Graph Theory [73].

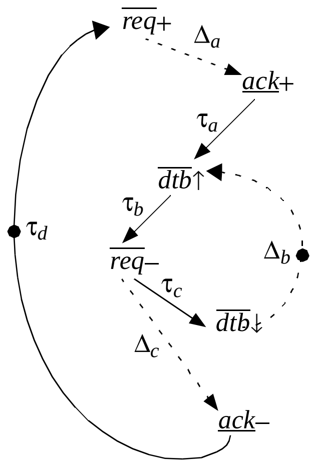


Figure 3.2.4   A bus arbitration protocol variant.

An interesting variation of the protocol shown in Figure 3.2.3, which is called *full handshake*, is the bus arbitration protocol shown in Figure 3.2.4, followed by typical DMA devices. This variation is called a *partial handshake*, for reasons that shall become obvious. The difference is that while in the original protocol, $\overline{req}-$ is generated after the master has released the $\overline{\text{DTB}}$ lines, in the latter $\overline{req}-$ is generated just before the release of the $\overline{\text{DTB}}$ lines. Let us examine the implications of this modification. There is a potential problem in the second protocol, if the arbiter grants the bus to another master before the current master tri-states DTB. The constraint edge $\Delta_c$ checks for this hazard. (Notice that $\Delta_c$ is an example of a constraint edge between two output signal transitions). However if the delay from $\overline{req}$– to a new grant $\underline{ack}$+ is larger than the delay $\tau_c$, then potentially the second protocol can display higher throughput because the arbitration for a new cycle occurs concurrently with the releasing of $\overline{\text{DTB}}$ by the current master.

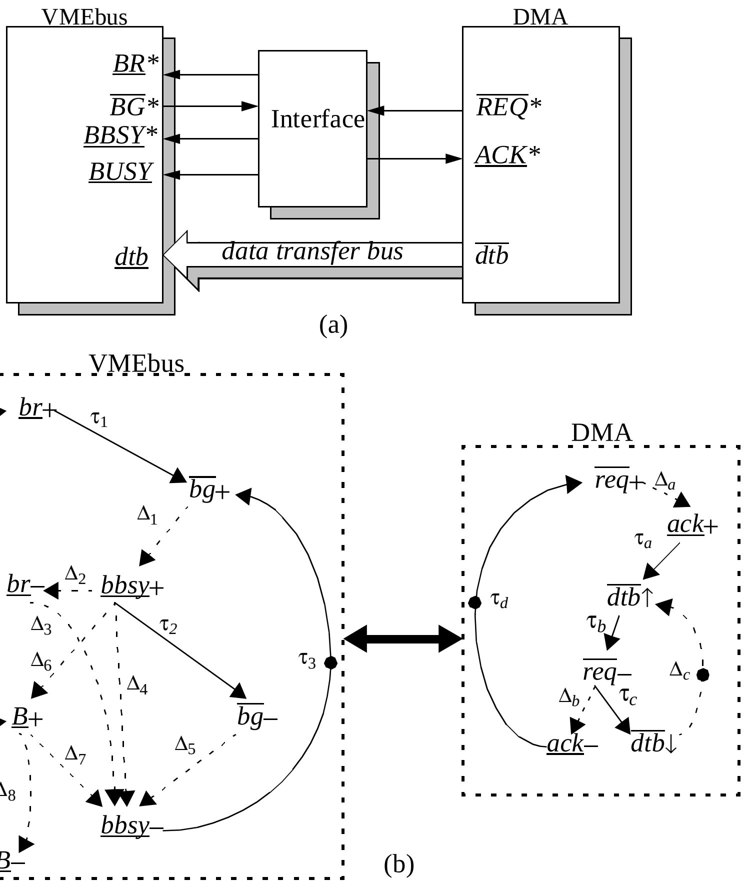


Figure 3.2.5 Bus arbitration interface: (a) structural view; (b) behavioral view.

Suppose that one wants to build up a system with our DMA device around the VME-bus. The VMEbus bus arbitration protocol is more involved [69]; it implements the arbiter's behavior. The VMEbus standard defines three control signals ($\underline{BR}$∗, $\overline{BG}$∗, and $\underline{BBSY}$∗) and one status signal ($\underline{BUSY}$) (refer to Figure 3.2.5). The structural view of the system is shown in Figure 3.2.5a. Notice that the DMA device uses two signals to perform bus arbitration while the VMEbus defines three controls signals and a status signal for bus arbitration. Thus interface logic is required to convert the two different protocols.

For the sake of clarity of the graph representation, we use the following short names: $\underline{br}$ for $\underline{BR}$*, $\overline{bg}$ for $\overline{BG}$*, $\underline{bbsy}$ for $\underline{BBSY}$* and $\underline{B}$ for $\underline{BUSY}$. The VMEbus bus arbitration interface specification is shown in Figure 3.2.5b. After a request is received ($\underline{br}$+) a grant is generated ($\overline{bg}$+). A master being granted the bus acknowledges the grant by asserting the grant-acknowledge signal ($\underline{bbsy}$+) to which the arbiter responds by releasing grant ($\overline{bg}$–). $\underline{bbsy}$ is used to speed up the arbitration similarly to the partial handshake described previously by allowing the arbitration to take place while the last part of the transaction is still in progress. The master must wait until the busy status signal ($\underline{B}$) is negated before driving the bus lines.

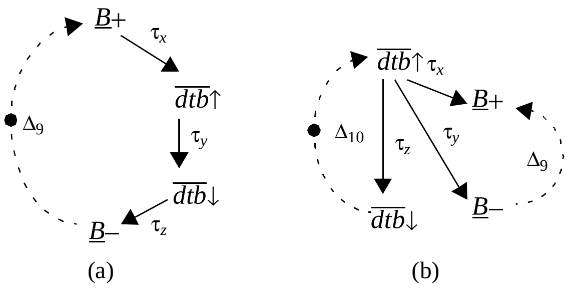


Figure 3.2.6 Bus busy status signal: (a) strobe relation; (b) actual relation.

The last piece of information describes the relationship between the use of the bus ($\overline{dtb}$) and the busy status signal ($\underline{B}$). A desirable behavior is that transitions $\underline{B}$+ and $\underline{B}$– "frame" the utilization of the bus (between $\overline{dtb}\uparrow$ and $\overline{dtb}\downarrow$), as shown in Figure 3.2.6a. This is known as a strobe relation, and has the advantage that $\underline{B}$ becomes negated only after the bus lines have been released by the master, so that by monitoring $\underline{B}$ it is possible to avoid bus collisions. However the VMEbus allows the designer to use one of the DTB lines (the address strobe signal) as the indicator of the status of the bus. Then $\overline{dtb}$ and $\underline{B}$ actually observe the relation shown in Figure 3.2.6b. Thus constraint link $\Delta_{10}$ needs to be added to prevent the possibility of a bus collision. All constraint places $\Delta_i$ in Figures 3.2.5 and 3.2.6 are assigned the interval $[0, \infty)$ except $\Delta_3$ and $\Delta_4$, which according to the VMEbus specifications [69] are assigned intervals $[30, \infty)$ and $[90, \infty)$ respectively, indicating a precedence requirement.

The question that arises is whether it is possible to carry out the design of an interface between two components whose protocols are described by these interface specifications (refer to Figure 3.2.5b). In this dissertation, we shall assume that a designer (human or machine) has produced a design. In the following paragraphs we give an informal discussion of some ideas of how to go about carrying a design needed to substantiate our definition of what we have called a "complete" graph, which is used to describe a system consisting of two (or more) interface specifications together with the interface logic that accomplishes the system integration. We stress that our aim is not to derive a theory about the construction of complete graphs but rather to introduce some basic requirements that we have identified. As a matter of fact in the following sections and chapters we shall assume that a complete graph is given.

First notice that the interface acts as the environment of each component. Therefore the interface must generate the necessary input actions expected by one component using the output actions. One possible approach is to add delay edges from the output signal transitions to the input transitions. For example, a delay edge may be added that connects $\overline{req}+$ to $\underline{br}+$. This is the approach considered in this work. Then the system can be described by a graph which contains the interface specifications of the components that are to be interconnected together with some additional edges corresponding to the interface logic. The resulting graph is called a "complete" graph, which shall be discussed further in the following section.

A procedure that automatically carries out the interface design is beyond the scope of this dissertation, although we believe that this is an important topic which demands further investigation. On another track, the implementation of the interface logic that realizes the added delay edges in the complete graph was explored in [42]. We also leave this problem for future work, although we believe that synthesis techniques that have been used for signal transition graphs are good candidates [79, 97, 123]. In this work, the starting point is a complete graph, and our goal is to investigate the temporal properties of the complete graph. These are the topics of Chapters 4 and 5.

### 3.2.2 Complete graphs

Two components can be interconnected if they share some capabilities (*e.g.*, data transfer, bus arbitration, *etc.*). The behavior of each capability is given by a protocol, which in turn can be formally described by an interface specification. In the previous section we introduced the interface design problem which arises when the protocols followed by two components to implement a capability are not "complementary" of one another (*e.g.*, there is no one-to-one correspondence between the input ports of one component and the output ports of the other component). If that is the case, interface logic is necessary to achieve the interconnection. In the previous section we also hinted at the possibility that a system composed of the components to be interconnected and the interface logic can be described using some sort of interface specification. In this section, we characterize such a representation that we have called a complete graph.

First we shall discuss the issue of describing the purpose of a protocol. This issue has been investigated in the related area of communication protocols where it has been given the name of semantic seed [102]. A semantic seed is intended to capture the semantics (or purpose) of the protocol. For example, the purpose of a bus arbitration protocol is to arbitrate the use of a shared resource, the data transfer bus lines, among several masters so that at most one master can take over the DTB at any given time. In [102] the semantic seed is represented as an automaton. In this work, we use constraint rules, which are more suitable to describe expected relationships (as discussed in Section 2.5.1), to specify the semantics of the protocol; constraint rules can describe non-causal timing relationships such as a hold time of -1 nsec for the input data with respect to the negated transition of the clock in a latch. We call this additional piece of information a semantic specification.

**Definition 3.2.1.-** Given two Petri nets $N_a = \langle P_a, T_a, F_a, M_{a0}, \Gamma_a \rangle$ and $N_b = \langle P_b, T_b, F_b, M_{b0}, \Gamma_b \rangle$, a semantic specification on $N_a$ and $N_b$ is a set of constraint rules $S = \{s_{ij}\}$, where $s_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon \rangle$, such that $t_i, t_j \in T_a \cup T_b$.

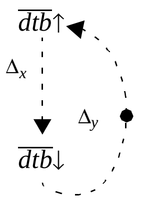


Figure 3.2.7 Bus arbitration semantic specification.

The two nets that Definition 3.2.1 refers to are the underlying Petri nets of the interface specifications that describe the protocols of two components that are to be interconnected. Although a semantic specification defines relationships between pairs of transitions of a Petri net, it can be extended to define relationships between pairs of signal transitions in the obvious way. Thus a semantic specification consists of timing constraints between selected signal transitions of the two protocols. As before, constraint rules can be represented graphically. For example Figure 3.2.7 shows the semantic specification of a bus arbitration protocol. It specifies that any consecutive uses of resource $\overline{dtb}$ must not overlap given that the $\Delta_x$ and $\Delta_y$ are $[0,\infty)$.

Before defining a complete graph, we need to define some basic concepts.

**Definition 3.2.2.-** An undirected path between two transitions $t_0$ and $t_n$ of a Petri net $N = \langle P, T, F, M_0, \Gamma \rangle$ is a sequence of transitions $t_0 t_1 \ldots t_n$ such that, for each

$i = 0,\ldots, n$-1, there exists a possibly non-unique $p_i \in P$ such that either $(t_i, p_i)$ or $(p_i, t_i)$ belongs to $F$ and either $(p_i, t_{i+1})$ or $(t_{i+1}, p_i)$ belongs to $F$.

**Definition 3.2.3.-** A directed path between two transitions $t_0$ and $t_n$ of a Petri net $N = \langle P, T, F, M_0, \Gamma\rangle$ is a sequence of transitions $t_0 t_1 \ldots t_n$ such that, for each $i = 0,\ldots, n$-1, there exists a possibly non-unique $p_i \in P$ such that $(t_i, p_i)$ and $(p_i, t_{i+1}) \in F$.

**Definition 3.2.4.-** A Petri net $N = \langle P, T, F, M_0, \Gamma\rangle$ is connected if there is an undirected path between any two transitions of the net.

A complete graph is an interface specification (refer to Definition 2.5.3) which satisfies a set of conditions as given in the following definition:

**Definition 3.2.5.-** Given two interface specifications $\Psi_a = \langle\Sigma_a, C_{Na}\rangle$ and $\Psi_b = \langle\Sigma_b, C_{Nb}\rangle$, with associated timed signal transition graphs $\Sigma_a = \langle N_a, Y_a, \lambda_a\rangle$ and $\Sigma_b = \langle N_b, Y_b, \lambda_b\rangle$, and a semantic specification $S$ on $N_a$ and $N_b$, a complete graph is the interface specification $\Psi_c = \langle\Sigma_c, C_{Nc}\rangle$ with associated timed STG $\Sigma_c = \langle N_c, Y_c, \lambda_c\rangle$ and set of constraint rules $C_{Nc}$ that ***satisfies*** the following conditions:

1. $N_a$ and $N_b$ are subgraphs of $N_c$;

2. $C_{Na}$, $C_{Nb}$ and $S$ are subsets of $C_{Nc}$;

3. $N_c$ is connected, and for every $t_i \in T_c$ there exists a directed path from a different $t_j \in T_c$;

4. There does not exist a place $p \in P_c \setminus (P_a \cup P_b)$ such that $\lambda_c(\bullet p)$ is a signal transition of an output port, where \ denotes set difference; and

5. $N_c$ is live.

The first two conditions of Definition 3.2.5 ensure that the sub-graphs contained in the interface specifications are part of the complete graph, and that the purpose of the protocols' capability is achieved. Notice that $N_c$ is not necessarily the union of $N_a$, $N_b$ and S for the case that additional signal transitions are required for instance to achieve protocol conversion. The third condition ensures that there are no dangling transitions and that each signal transition is generated by other transition(s). Examples of nets that do not satisfy condition 3 are shown in Figure 3.2.8. The first net is disconnected but there is a directed path that terminates in every transition. The second net is connected but there are no directed paths that terminate in transitions *a* or *b.* Notice that connectivity is only checked on the Petri net part of the complete graph. The fourth condition disallows new delay places which sink into an output transition: output signal transitions of the interface specifications are generated internally by a component and should not be driven by the interface logic. Finally the fifth condition guarantees that every signal transition is live, and thus deadlock does not occur (refer to Section 2.2.1).

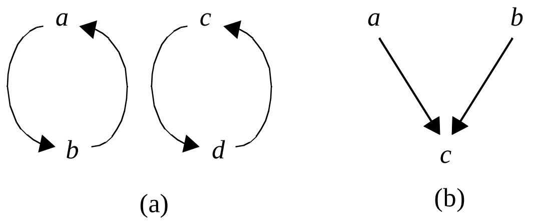


Figure 3.2.8   Examples of nets that do not satisfy condition 3 of Definition 3.2.4.

Given two interface specifications for two components that are to be interconnected, there may be quite a few different ways of designing an interface. Some designs may have nicer properties than others (*e.g.*, a design may be faster or may occupy less silicon area). Our definition of a complete graph aims to describe a minimum set of structural properties that a graph representing an interface design should have. For example that input ports must not be disconnected. The term structural properties refers to properties concerning the connectivity of the graph, and not the behavior of the underlying Petri net. It may be the case that a valid complete graph may not exhibit a correct behavior (*e.g.*, because it violates some of the timing constraints). At the end of this chapter we shall discuss the concept of feasibility of an interface. Informally, an interface is feasible if it implements the environment of each of the components correctly from the point of view of timing. Thus feasibility shall be used to check the correctness of the timing behavior of the interface.

A complete graph derived from the bus arbitration interface specifications shown in Figure 3.2.5 and the semantic specification shown in Figure 3.2.7 is depicted in Figure 3.2.9. The added delay edges are shown as thicker lines and are labeled with random variables $\delta_i$. One can check that the four conditions of Definition 3.2.4 are satisfied: the original interface specifications and the semantic specification are part of the complete graph; the underlying petri net is connected and every signal transition has an incoming delay place; finally none of the added delay edges sinks into an output signal transition as stated in Definition 3.2.5.

Let us recall that the intention of Definition 3.2.5 is to provide necessary, although maybe not sufficient, conditions for a representation of an interface design. Also notice that if components share more than one capability then an interface design may consist of several complete graphs. A procedure to efficiently combine all the complete graphs is left as future work.

## 3.3 Time-consistency of complete graphs

From our previous discussion, a complete graph, like an interface specification, consists of an STG and a set of constraint rules. The STG of a complete graph is a connected net such

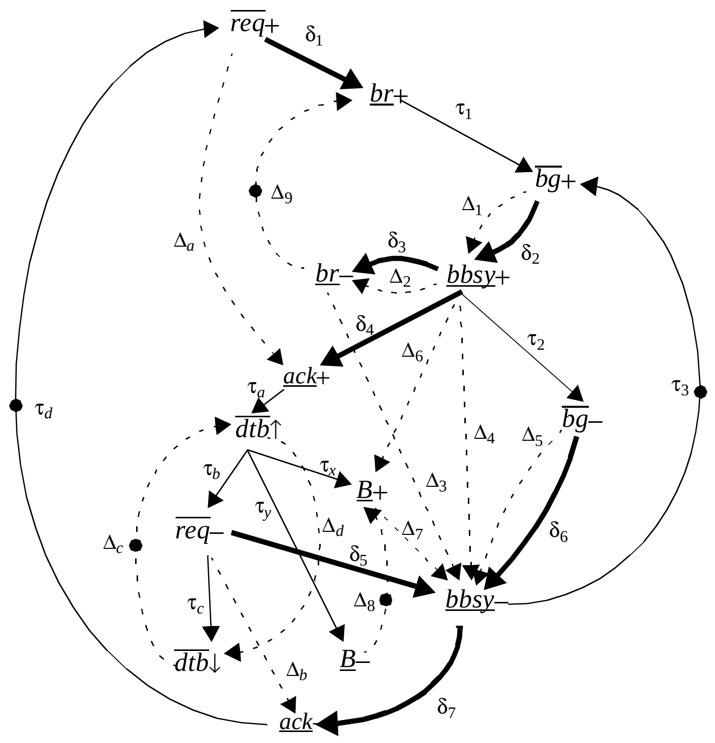


Figure 3.2.9 Bus arbitration interface design.

that every transition is generated by other transition(s); and the complete graph "contains" two component interface specifications and a semantic specification. There are some additional edges in the complete graph in addition to the ones corresponding to the component interface specifications and the semantic specification: those edges describe the interface circuit that realizes protocol conversion. In general the construction of a complete graph may result in an STG which is not AOC (refer to Section 2.4.2). As we already mentioned it is not our aim to develop a theory to construct complete graphs. We shall assume in the sequel that the STG part of a complete graph is AOC. How severe this assumption is we do not know, although from our experience it seems reasonable. Moreover should this assumption prove too limiting in some cases, our approach could provide a basic framework for future development.

In this section, we shall investigate the temporal behavior of complete graphs. As mentioned in the previous section, it is possible that a complete graph may not exhibit a correct temporal behavior. The question we are trying to answer is: Is it possible to determine if all the possible temporal behaviors of a complete graph satisfy all the given timing constraints? We have found conditions under which the answer to the question is positive. First we discuss the timed execution of an AOC STG.

### 3.3.1 STG unfolding

A key problem of our verification and analysis techniques is to find the time separation between two transitions, as it will be discussed in Section 3.3.2. This problem is not trivial for live nets, even for sub-classes of Petri nets using deterministic firing times [110, 117], or using interval data [3, 67]. In this section we describe the execution of a signal transition graph using partial orders.

Partial orders have been used to describe the operational semantics of concurrent systems [54, 12]. In contrast to the interleaving semantics of reachability graphs (refer to Section 2.2.1), partial orders avoid the state explosion that occurs in highly concurrent systems by not having to represent all possible interleavings of a net. The STG unfolding presented in this section is a partial order technique.

We shall need the following definitions adapted from [113] in subsequent sections:

**Definition 3.3.1.-** Let $A$ be a set. A binary relation $\rho \subseteq A \times A$ is called a similarity relation if and only if $\forall\, a,b \in A$:

1. $a \,\rho\, a$
2. $a \,\rho\, b \Rightarrow b \,\rho\, a$

**Definition 3.3.2.-** Let $A$ be a set. A binary relation $\rho \subseteq A \times A$ is called a partial order if and only if $\forall\, a,b \in A$:

1. $\neg(a \, \rho \, a)$

2. $a \, \rho \, b \wedge b \, \rho \, c \Rightarrow a \, \rho \, c$

Thus a similarity relation is reflexive and symmetric, and a partial order is irreflexive and transitive. Notice that conditions 1 and 2 of a partial order imply anti-symmetry (*i.e.*, $a \, \rho \, b \Rightarrow \neg(b \, \rho \, a)$). Let us denote a partial order by '<'.

**Definition 3.3.3.-** Let $A$ be a partially ordered set.

1. Let $li \subseteq A \times A$ be given by $a \; li \; b$ if and only if $a < b$ or $b < a$ or $a = b$

2. Let $co \subseteq A \times A$ be given by $a \; co \; b$ if and only if $\neg(a \; li \; b)$ or $a = b$.

Relations *li* and *co* stand for linear (precedence) and concurrent respectively. It can be shown that *li* and *co* are similarity relations [113].

**Definition 3.3.4.-** A subset $B \subseteq A$ is called a *region* of a similarity relation ρ if and only if:

1. $\forall \; a,b \in B$: $a \, \rho \, b$

2. $\forall \; a \in A$: $a \notin B \Rightarrow \exists \; b \in B$: $\neg(a \, \rho \, b)$

**Definition 3.3.5.-** Let $A$ be a partially ordered set, and let $B \subseteq A$. $B$ is called a *line* if $B$ is a region of *li*. $B$ is called a *cut* if $B$ is a region of *co*.

Figure 3.3.1 shows the graphical representation of a partial ordered set, or poset, where there is a directed edge from $a$ to $b$ if and only if $a < b$ and there is no $c$ such that $a < c < b$. The similarity relations *li* and *co* are also depicted (if ρ is a similarity relation, then a graphical representation draws an undirected edge from $a$ to $b$ if and only if $a \, \rho \, b$). The *li* relation formalizes the idea of a path in a poset; that is, if $a \; li \; b$ then there is an directed path from $a$ to $b$ or from $b$ to $a$. The cut relation describes events that are concurrent; if $a \; co \; b$ then $a$ and $b$ can occur simultaneously.

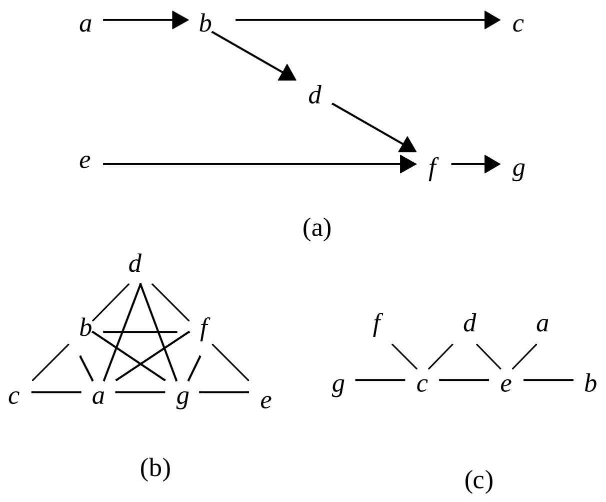


Figure 3.3.1 A poset (a) and its relations (b) *li*; and (c) *co*.

The connectedness and liveness properties of the signal transition sub-graph of a complete graph imply the presence of cycles (*i.e.*, directed paths which start and end at the same transition). Acyclic graphs, graphs without cycles, are easier to analyze [58]. The unfolding of a signal transition graph is an acyclic graph in which different occurrences of the transitions are considered as different nodes of the graph. The notion of the unfolded graph of an STG corresponds to the notion of processes and occurrence nets in Condition/ Event Systems [113].

So far we have represented the operational behavior of an interface specification, or a complete graph, graphically using its AOC di-graph $\Omega = \langle T_\Omega, P_\Omega, M_{\Omega 0}, \Gamma_\Omega, ct_\Omega, Y, \lambda_\Omega\rangle$, and we have used the equivalent STG $\Sigma = \langle N, Y, \lambda\rangle$ with associated net $N = \langle P, T, F, M_0, \Gamma\rangle$ for the development of the theory. The STG unfolding procedure that we shall describe below is more easily understood in terms of $\Omega$. Notice however that the translation of the results to the STG is straightforward (refer to Section 2.3.3). The left arrow $\leftarrow$ is used to denote assignment, and **0** is the special root transition of the di-graph.

**Procedure 3.3.1.-** The unfolding of an AOC di-graph $\Omega$ is the di-graph $\Omega_U = \langle T_{U\Omega}, P_{U\Omega}, \Gamma_{U\Omega}, ct_{U\Omega}, Y, \lambda_{U\Omega}\rangle$ constructed from $\Omega$ according to the following steps:

```
$T_{U\Omega} \leftarrow \{\mathbf{0}\}, P_{U\Omega} \leftarrow \{\}$
Define the index function $idx : T_\Omega \rightarrow \aleph$
$\forall\, t \in T_\Omega, idx(t) \leftarrow 0$
$U \leftarrow \{t_j : M_{\Omega 0}((t_i, t_j)) = 1 \wedge (t_i, t_j) \in P_\Omega\}$
$\forall\, t_j \in U, idx(t_j) \leftarrow idx(t_j) + 1$
$V \leftarrow \{t_{j(idx(tj))} : t_j \in U\}, W \leftarrow \{(\mathbf{0}, t_{j(idx(tj))}) : t_j \in U\}$
$T_{U\Omega} \leftarrow T_{U\Omega} \cup V, P_{U\Omega} \leftarrow P_{U\Omega} \cup W$
$\forall\, t_j \in U \wedge (t_i, t_j) \in P_\Omega, \Gamma_{U\Omega}((\mathbf{0}, t_{j(idx(tj))})) \leftarrow \Gamma_\Omega((t_i, t_j))$
$\forall\, t_j \in U, ct_{U\Omega}(t_{j(idx(tj))}) \leftarrow ct_\Omega(t_i)$
$\forall\, t_j \in U, \lambda_{U\Omega}(t_{j(idx(tj))}) \leftarrow \lambda_\Omega(_j)$
do forever
    $U' \leftarrow \{t_j : t_i \in U \wedge (t_i, t_j) \in P_\Omega\}$
    $\forall\, t_j \in U', idx(t_j) \leftarrow idx(t_j) + 1$
    $V \leftarrow \{t_{j(idx(tj))} : t_j \in U'\}$
    $W_{\mathrm{I}} \leftarrow \{(t_{i(idx(ti))}, t_{j(idx(tj))}) : t_j \in U' \wedge t_i \neq t_j \wedge (t_i, t_j) \in P_\Omega\}$
    $W_{\mathrm{II}} \leftarrow \{(t_{i(idx(ti)-1)}, t_{i(idx(ti))}) : (t_i, t_i) \in P_\Omega\}$
    $T_{U\Omega} \leftarrow T_{U\Omega} \cup V, P_{U\Omega} \leftarrow P_{U\Omega} \cup W_{\mathrm{I}} \cup W_{\mathrm{II}}$
    $\forall\, t_j \in U' \wedge (t_i, t_j) \in P_\Omega, \Gamma_{U\Omega}((t_{i(idx(ti))}, t_{j(idx(tj))})) \leftarrow \Gamma_\Omega((t_i, t_j))$
    $\forall\, t_j \in U', ct_{U\Omega}(t_{j(idx(tj))}) \leftarrow ct_\Omega(t_j)$
    $\forall\, t_j \in U', \lambda_{U\Omega}(t_{j(idx(tj))}) \leftarrow \lambda_\Omega(t_j)$
    $U \leftarrow U'$
end do
```

Procedure 3.3.1 constructs $\Omega_U$ as follows: First it sets the root transition **0**, which represents time $\tau = 0$. The function *idx* keeps track of the number of occurrences of each transition. The transitions $t_i$ enabled by the initial marking of $\Omega$ are added to $T_{U\Omega}$ as nodes $t_{i(1)}$, that is the first occurrence of each transition, and $idx(t_i)$ is increased by 1; edges $(\mathbf{0}, t_{i(1)})$ are added to $P_{U\Omega}$. The new nodes (edges) added to $\Omega_U$ are labeled with the signal transitions and causality type (random variables) of the original nodes (edges) in $\Omega$. The same process is repeated indefinitely for transitions $t_j$ to which that transitions $t_i$ are connected.

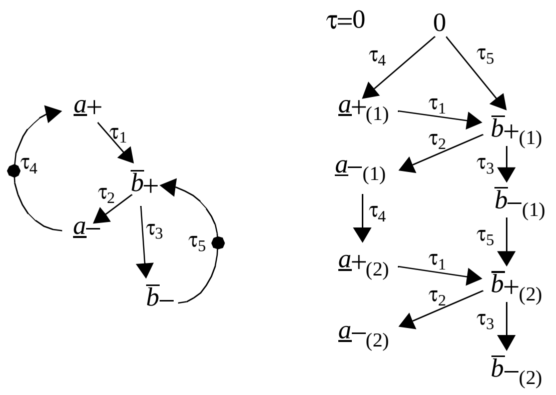


Figure 3.3.2 Signal transition graph and a partial view of its infinite acyclic unfolding.

Consider for example the STG shown in Figure 3.3.2. The initial marking assigns one token to each of the edges labeled with random variables $\tau_4$ and $\tau_5$. Those tokens are assumed to arrive at the places at time $\tau = 0$. Thus from the root transition 0, two edges labeled $\tau_4$ and $\tau_5$ are connected to two nodes labeled $\underline{a}+_{(1)}$ and $b+_{(1)}$ respectively. From $\underline{a}+_{(1)}$, there is an edge $\tau_1$ to $b+_{(1)}$, and from $b+_{(1)}$ there are two edges $\tau_2$ and $\tau_3$ connected to $\underline{a}+_{(1)}$ and $b-_{(1)}$ respectively. Thus from $\underline{a}+_{(1)}$, edge $\tau_1$ is added to $b+_{(1)}$, and edges $\tau_2$ and $\tau_3$ are added to $\underline{a}-_{(1)}$ and $b-_{(1)}$ respectively.

Notice that the unfolding of the AOC STG is an infinite acyclic graph.

**Lemma 3.3.1.-** Given an unfolding $\Omega_U$, let ρ be a binary relation on $T_{U\Omega}$ such that $t_{i(m)}\ \rho\ t_{j(n)}$ if and only if $(t_{i(m)}, t_{j(n)}) \in P_{U\Omega}$. Then the transitive (but not reflexive) closure of ρ, denoted by ρ*, is a partial order.

**Proof.-** From Definition 3.3.2, a partial order is a binary relation which is transitive and irreflexive. Relation ρ* is by construction transitive. Then we shall show that ρ* is irreflexive, that is $(t_{i(m)}, t_{i(m)}) \notin P_{U\Omega}$ for $t_{i(m)} \in T_{U\Omega}$. To show this fact, suppose that $(t_{i(m)}, t_{i(m)}) \in P_{U\Omega}$ for a transition $t_{i(m)} \in T_{U\Omega}$. The set $P_{U\Omega}$ is constructed in the step $P_{U\Omega} \leftarrow P_{U\Omega} \cup W_{\mathrm{I}} \cup W_{\mathrm{II}}$. By construction, $(t_{i(m)}, t_{i(m)})$ could only be added to $P_{U\Omega}$ in $W_{\mathrm{II}}$

which takes care of the case that $(t_i, t_i) \in P_\Omega$ in the original AOC di-graph. But explicitly the construction of W guarantees that if $(t_i, t_i) \in P_\Omega$, then $(t_{i(m-1)}, t_{i(m)})$ is added to $P_{U\Omega}$. □

Thus ρ* is a partial order on the infinite set $T_{U\Omega}$, and the AOC di-graph can be thought of as the graphical representation of ρ*. In the sequel we shall call ρ* the precedence relation < of the unfolding $\Omega_U$.

## 3.3.2 Time-consistency

From a timing perspective, a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$ describes two different aspects: a net execution given by the signal transition graph $\Sigma_c$, and a set of timing constraints $C_{Nc}$ which specify a desired behavior. The connectivity properties of the signal transition graph $\Sigma_c$ of a complete graph assure that; this is not the case with interface specifications, whose input signal transitions are not generated by other signal transitions (*i.e.*, there are no delay edges incoming to input signal transitions). Time-consistency is a property of a complete graph whose net execution satisfies its timing constraints. Notice that time-consistency is not a property of the interface specifications.

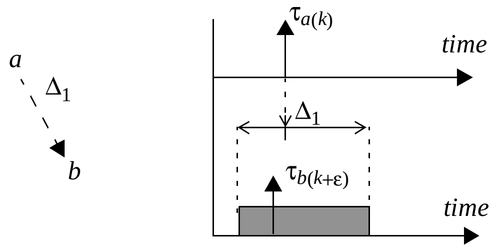


Figure 3.3.3 Constraint rule for transitions *a* and *b*.

As discussed in Section 2.5.1, a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon \rangle$ on two transitions of the net $N = \langle P, T, F, M_0, \Gamma \rangle$ defines a time window $\Delta_{ij}$ with respect to the $k$-th occurrence of the constraining transition $t_i$ during which the ($k$+ε)-th occurrence of the constrained transition $t_j$ is allowed (refer to Figure 3.3.3).

**Definition 3.3.6.-** Given a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$, a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon \rangle$ of $C_{Nc}$ is said to be *satisfied* if for every possible execution of the underlying Petri net $N_c$ of $\Sigma_c = \langle N_c, Y_c, \lambda_c \rangle$, it is true that:

$$\tau_{t_{j(k+\varepsilon)}} - \tau_{t_{i(k)}} \in \Delta_{ij} \qquad \text{(Eq. 3.3.1)}$$

for all $k > 0$, where $\tau_{t_{i(k)}}$ is the time of the $k$-th occurrence of transition $t_i$. Otherwise $c_{ij}$ is said to be *violated*.

Eq. 3.3.1 is called the constraint equation for constraint rule $c_{ij}$. The expression $\tau_{t_{j(k+\varepsilon)}} - \tau_{t_{i(k)}}$ denotes the time separation from the $k$-th occurrence of transition $t_i$ to the ($k$+ε)-th occurrence of transition $t_j$ (recall that ε can be either 0 or 1).

**Definition 3.3.7.-** A complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$ is said to be time-consistent if every constraint rule $c_{ij} \in C_{Nc}$ is satisfied.

To compute the time separation of Eq. 3.3.1, we can unfold the STG starting from the initial marking, and then check that Eq. 3.3.1 applies. Notice that we have to do this for an infinite number of occurrences (*i.e.*, $k > 0$); moreover, because the occurrences $\tau_{t_{i(k)}}$ are not deterministic, we have to consider a possibly infinite ensemble of occurrences $\tau_{t_{i(k)}}$ (refer to Section 2.3.4).

One point to note is that in general if there is no trivial constraint (*i.e.*, a constraint with associated interval $(-\infty, +\infty)$ ) between two transitions, they should have a common ancestor, otherwise the constraint would not be satisfied because the time occurrence of the transitions would be independent of one another and their time separation can be arbi-

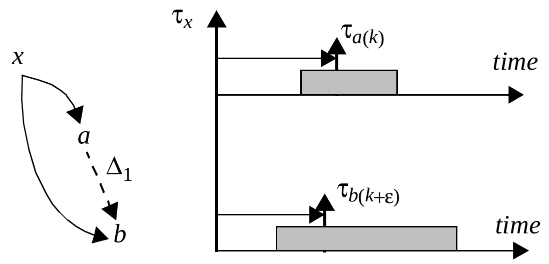


Figure 3.3.4 Time separation between transitions *a* and *b* with respect to a common ancestor transition.

trarily large. Suppose for the moment that such a common ancestor exists (refer to Figure 3.3.4). If the common ancestor is within finite reach, then one does not have to check an infinite number of time separations.

In the following section we shall show that such a common ancestor does exist in an unfolding of an AOC STG. Rather than finding $\tau_{t_{i(k)}}$ and $\tau_{t_{j(k+\varepsilon)}}$ separately and then computing Eq. 3.3.1 for each possible $\tau_{t_{i(k)}}$ and $\tau_{t_{j(k+\varepsilon)}}$, we could determine Eq. 3.3.1 with respect of the common ancestor.

Consider for example the time separation $\tau_{b+(1)} - \tau_{\underline{a}+(1)}$ in Figure 3.3.2, The common ancestor to both $\underline{a}+_{(1)}$ and $b+_{(1)}$ is the root transition 0. Transition $\underline{a}+_{(1)}$ occurs at time $\tau_{\underline{a}+(1)} = \tau_4$. Transition $b+_{(1)}$ occurs when there are visible tokens at both edges $\tau_1$ and with $\tau_5$, that is at $\tau_{b+(1)} = \max(\tau_4+\tau_1, \tau_5)$. Then $\tau_{b+(1)} - \tau_{\underline{a}+(1)} = \max(\tau_4+\tau_1, \tau_5) - \tau_4$.

We shall discuss more thoroughly our algebraic approach to computing Eq. 3.3.1 in Section 3.3.4. Notice that we still have to deal with an infinite number of occurrences. In the following section we introduce a notable type of common ancestor, the cycle-invariant fork transition, that can solve this problem.

### 3.3.3 Fork transitions

The satisfaction of timing constraints plays an important role in our approach. As mentioned in the previous section, the constraint equation must check that the constraint rule is satisfied over all executions of the net, that is, for all $k > 0$. In this section we investigate under which circumstances the constraint equation becomes independent of the occurrence index $k$. First we formalize the concept of common ancestor of two transitions.

A common ancestor of two transitions $a$ and $b$ from which the left-hand side expression $\tau_{t_{j(k+\varepsilon)}} - \tau_{t_{i(k)}}$ of the constraint equation (Eq. 3.3.1) can be computed is called a *fork transition*.

**Definition 3.3.8.-** Given the unfolding $\Omega_U$ of an AOC di-graph, a transition $x$ is called a fork transition of transitions $t_{i(k)}$ and $t_{j(k+\varepsilon)}$ if in the partial order $<$ induced by the unfolding, $x < t_{i(k)}$ and $x < t_{j(k+\varepsilon)}$, and every line of $<$ containing either $t_{i(k)}$ or $t_{j(k+\varepsilon)}$ contains also $x$.

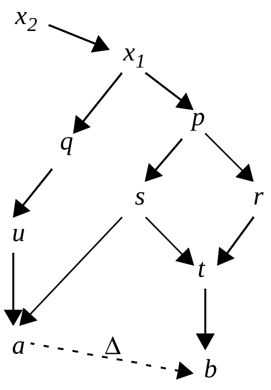


Figure 3.3.5 Fork transition $x$ of transitions $a$ and $b$.

According to Section 3.3.1, a line is a region of the *li* relation generated by the precedence relation $<$. Thus a line of $<$ corresponds to a single path in the AOC di-graph. Consider for example the di-graph shown in Figure 3.3.5. Transition $s$ is not a fork transition of $a$ and $b$, because there are lines of $<$ containing $a$ but not $s$ (*e.g.*, line $\{a, u, q, x_1, x_2\}$). Transition $x_1$ is a fork transition of $a$ and $b$. Notice that a fork transition may not be unique. For example, $x_2$ is also a fork transition of $a$ and $b$.

**Lemma 3.3.2.-** Every pair of transitions of an unfolding $\Omega_U$ have at least one fork transition.

**Proof.-** From the construction of $\Omega_U$ by Procedure 3.3.1, it is easy to see that the root transition 0 precedes every transition. Also every path starts at transition 0, therefore every line of $<$ contains transition 0. □

As mentioned in the previous section, we would like to compute $\tau_{t_{j(k+\varepsilon)}} - \tau_{t_{i(k)}}$ with respect to a common ancestor of $t_{i(k)}$ and $t_{j(k+\varepsilon)}$, rather than computing each possible $\tau_{t_{i(k)}}$ and $\tau_{t_{j(k+\varepsilon)}}$. As shown in Figure 3.3.5, not every ancestor is suitable. For instance, had transition $s$ in Figure 3.3.5 been chosen as an ancestor, then the effect of transition $u$ on the firing of $a$ would have been ignored, as it would have been the effect of $r$ on the firing of $b$.

Thus a fork transition is a synchronization point, or time origin, from which the time separation $\tau_{t_{j(k+\varepsilon)}} - \tau_{t_{i(k)}}$ can be computed. From this point of view, it should be clear that the root transition 0 is a fork transition for every pair of transitions. However we are interested in the fork transition closest to the transitions $t_{i(k)}$ and $t_{j(k+\varepsilon)}$. Furthermore, we are interested in a fork transition that "moves" with $t_{i(k)}$ and $t_{j(k+\varepsilon)}$.

**Definition 3.3.9.-** A pair of transitions $t_i$ and $t_j$ of a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$ has a *cycle-invariant fork transition* if in the unfolding $\Omega_U$ of $\Sigma_c$ there exist two integers, $\lambda$ and $M$, and a transition $x_{(k-\lambda)}$, such that $x_{(k-\lambda)}$ is a fork transition of $t_{i(k)}$ and $t_{j(k+\varepsilon)}$ for all $k \geq M$.

**Definition 3.3.10.-** Given a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$ and a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon \rangle$ such that $c_{ij} \in C_{Nc}$, the ordered pair $(t_i, t_j)$ is said to be a pair of transitions constrained by $c_{ij}$, written as $(t_i \rightarrow t_j)c_{ij}$.

**Definition 3.3.11.-** A complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$ is said to be cycle-invariant if every pair of transitions $(t_i, t_j)$ of $\Sigma_c$ such that $(t_i \rightarrow t_j)c_{ij}$, where $c_{ij} \in C_{Nc}$, has a cycle-invariant fork transition.

There must be a cycle-invariant fork transition for every constrained pair of transitions, which must be checked according to the interface and semantic specifications.

**Definition 3.3.12.-** A pair of transitions $t_{i(k)}$ and $t_{j(k+\varepsilon)}$ of an unfolding $\Omega_U$ are said to be repeatable if there exist an integer $M$ for which the probability density function of the time separation $\tau_{t_{j(k+\varepsilon)}} - \tau_{t_{i(k)}}$ is invariant for $k \geq M$.

**Lemma 3.3.3.-** If a pair of constrained transitions $(t_i \rightarrow t_j)c_{ij}$ is repeatable, then there is a finite procedure that can check if the constraint rule $c_{ij}$ is satisfied.

**Proof.-** First check the constraint equation for $c_{ij}$ for all $0 < k < M$. According to Definition 3.3.12, the left-hand side of the constraint equation for $c_{ij}$ is invariant for $k \geq M$, so that it is sufficient to check the constraint equation for $k = M$ to determine if the constraint is satisfied for $k \geq M$. □

It is clear from the previous presentation that it is desirable to find if two constrained transitions are repeatable. In the following section we shall show that a cycle-invariant fork transition is a necessary condition for repeatability.

### 3.3.4 Computing constraint equations

In this section we give a procedure to express constraint equations in terms of the random variables associated with the edges of the AOC di-graph.

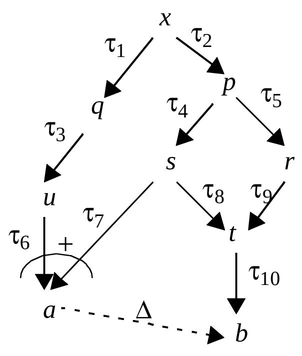


Figure 3.3.6 Unfolding for transitions *a* and *b* from their fork transition.

The characterization of a fork transition *x* of two transitions *a* and *b* given by Definition 3.3.8 makes it suitable to be considered as a time origin, because all the paths of the unfolding of the STG from the root transition 0 to either *a* or *b* must pass through *x*. In this section we try to compute the time separations $\tau_a - \tau_x$, from *x* to *a*, and $\tau_b - \tau_x$ from *x* to *b*, in terms of the random variables associated with the edges (refer to Figure 3.3.6). Because *x* is a common reference point for both *a* and *b*, then the left-hand side of Eq. 3.3.1 can be written as:

$$\tau_b - \tau_a = (\tau_b - \tau_x) - (\tau_a - \tau_x) \qquad \text{(Eq. 3.3.2)}$$

The computation of $\tau_b - \tau_x$, or $\tau_a - \tau_x$, in term of the random variables associated with the edges must obey the firing semantics of AND and OR causality (refer to Section 2.4.1) that we reproduce below.

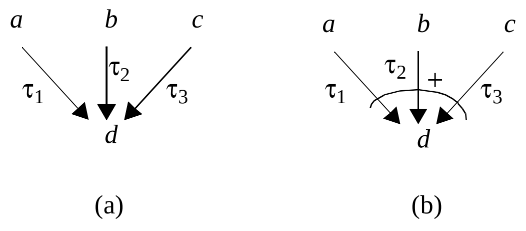


Figure 3.3.7 (a) AND causality; (b) OR causality.

If transition *d* is AND-caused by transitions *a*, *b*, and *c* (refer to Figure 3.3.7a), then the occurrence time of *d* is:

$$\tau_d = \max(\tau_a + \tau_1, \tau_b + \tau_2, \tau_c + \tau_3) \qquad \text{(Eq. 3.3.3)}$$

where $\tau_a$, $\tau_b$, and $\tau_c$ are the occurrence times of transitions *a*, *b*, and *c* respectively.

Similarly, if transition *d* is OR-caused by transitions *a*, *b*, and *c* (refer to Figure 3.3.7b), then the occurrence time of *d* is:

$$\tau_d = \min(\tau_a + \tau_1, \tau_b + \tau_2, \tau_c + \tau_3) \qquad \text{(Eq. 3.3.4)}$$

Finally if there is only one transition that causes *d*, say *a*, then:

$$\tau_d = \tau_a + \tau_1 \qquad \text{(Eq. 3.3.5)}$$

and the causality type is not important.

A procedure to compute the time separation, say $\tau_a - \tau_x$, is as follows:

**Procedure 3.3.2.-**

1. Expand $\tau_a$ according to the causality type of transition *a*, by using Eq. 3.3.3, Eq. 3.3.4, or Eq. 3.3.5.

2. Recursively expand the occurrence times of transitions that appear in the expression for t, until the fork transition is reached.

At the end of the procedure, the expression for $\tau_a$ contains min/max/linear terms on the random variables $\tau_i$ and $\tau_x$. Moreover, each linear term that contains $\tau_x$ is of the form $\tau_x + \ldots + \tau_i$.

Once Procedure 3.3.2 is applied to find an expression for $\tau_b$ in terms of $\tau_i$ and $\tau_x$, the expression for the time separation $\tau_b - \tau_a$ can be computed from Eq. 3.3.2, where $\tau_b \quad \tau_x$ and $\tau_a - \tau_x$ are given by the expressions produced for $\tau_b$ and $\tau_a$ by Procedure 3.3.2 in which the terms $\tau_x$, corresponding to the reference point, have been removed.

For example, consider the di-graph shown in Figure 3.3.6. The expression for $\tau_a$ is $\min(\tau_u + \tau_6, \tau_s + \tau_7)$, which is expanded into $\min(\tau_q + \tau_3 + \tau_6, \tau_p + \tau_4 + \tau_7)$, and finally into $\min(\tau_x + \tau_1 + \tau_3 + \tau_6, \tau_x + \tau_2 + \tau_4 + \tau_7)$. Similarly the final expression for $\tau_b$ is $\tau_2 + \tau_{10} + \max(\tau_x + \tau_4 + \tau_8, \tau_x + \tau_5 + \tau_9)$. Then,

$$\tau_b - \tau_a = \tau_2 + \tau_{10} + \max(\tau_4 + \tau_8, \tau_5 + \tau_9) - \min(\tau_1 + \tau_3 + \tau_6, \tau_2 + \tau_4 + \tau_7) \qquad \text{(Eq. 3.3.6)}$$

In Chapter 4 we shall give a procedure to obtain the probability density function of Eq. 3.3.2. Now we can prove the following lemma.

**Lemma 3.3.4.-** If a pair of transitions $t_i$ and $t_j$ of a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$ has a cycle invariant fork transition then $t_i$ and $t_j$ are repeatable.

**Proof.-** Consider transitions $t_i$ and $t_j$. By Definition 3.3.9, transitions $t_{i(k)}$ and $t_{j(k+\varepsilon)}$ of the unfolding $\Omega_U$ have a transition $x_{(k-\lambda)}$ for each $k \geq M$ (refer to Figure 3.3.8). We can find equations for the time separations $\tau_{ti(k)} - \tau_{x(k-\lambda)}$, from $x_{(k-\lambda)}$ to $t_{i(k)}$, and $\tau_{tj(k+\varepsilon)} - \tau_{x(k-\lambda)}$ from $x_{(k-\lambda)}$ to $t_{j(k+\varepsilon)}$ using Procedure 3.3.2. These two time separations are invariant for $k \geq M$, because by construction of the unfolding $\Omega_U$ from the AOC di-graph $\Omega$ (refer to

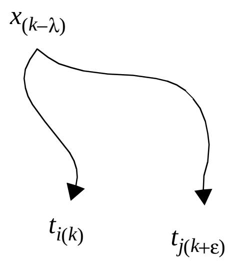


Figure 3.3.8 Fork transition for $k \geq M$.

Procedure 3.3.1) the same sub-graph must exist from $x_{(k-\lambda)}$ to $t_{i(k)}$ and $t_{j(k+\varepsilon)}$. Thus a cycle-invariant fork transition is a synchronization point for both transitions $t_{i(k)}$ and $t_{j(k+\varepsilon)}$. Then

$$\tau_{t_{j(k+\varepsilon)}} - \tau_{t_{i(k)}} = (\tau_{t_{j(k+\varepsilon)}} - \tau_{x_{(k-\lambda)}}) - (\tau_{t_{i(k)}} - \tau_{x_{(k-\lambda)}}) \qquad \text{(Eq. 3.3.7)}$$

applies for all $k \geq M$. Eq. 3.3.7 implies that the probability density function of the time separation $\tau_{tj(k+\varepsilon)} - \tau_{ti(k)}$ is invariant for $k \geq M$. Therefore $t_i$ and $t_j$ are repeatable. □

The existence of a cycle-invariant fork transition is only a sufficient condition. That means that a pair of transitions without a cycle-invariant fork transition may be repeatable. This subject has been studied in the literature (*cf.* [6], and [67]). One example is the STG shown in Figure 3.3.9. It can be shown that for any pair of transitions there is no cycle-invariant fork transition. Such type of nets may exhibit repeatable behavior, however they may also exhibit non-repeatable periodic behavior with arbitrarily long transients by just changing the delays, and thus they are difficult to analyze. In the interface specification of components that we have studied, we have never found this type of net. We conjecture that the reason is that their behavior is time-dependent, that is, by changing the timing parameters, it is possible for the same system to exhibit radically different behaviors, something that is not desirable in a protocol.

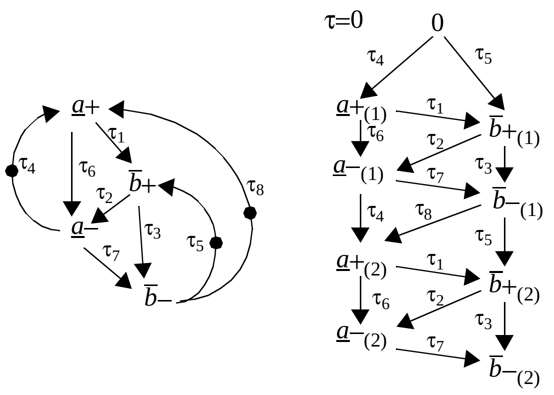


Figure 3.3.9 An STG whose transitions do not have a cycle-invariant fork transition.

So far we have assumed that a (cycle-invariant) fork transition has been found. In the following section we present an algorithm that can find a fork transition, if one exists, of two transitions in a finite unfolding of an AOC STG.

## 3.3.5 Procedure to find fork transitions

The procedure that is discussed in this section finds a fork transition of two transitions $t_{i(k)}$ and $t_{j(k+\varepsilon)}$ of a finite unfolding of an AOC STG, if one exists, otherwise flags that no fork transition could be found. The finite unfolding is a finite graph $\Omega_{Uf}$ produced by Procedure 3.3.1 by expanding the AOC STG until transitions $t_{i(k)}$ and $t_{j(k+\varepsilon)}$ are added to $\Omega_{Uf}$, for a $k = M$.

The fork transition procedure requires that the unfolding $\Omega_{Uf}$ be sorted in topological order. A topological ordering or sort of an acyclic di-graph assigns a level to each node of the graph.

A di-graph can be represented as a pair $G = \langle V, E \rangle$, where $V$ is a set of nodes and $E \subseteq V \times V$ is a set of edges. A node $a$ of a directed graph is called a root node if there does not exist a node $b$ such that $(b, a)$ is an edge of the graph. The in-degree of node $b$ is the number of edges ($a_i$, $b$) of the graph. The topological level of a node $a \in V$ in $G$ is computed as follows: the root node is assigned to level 0; then recursively if $A$ is the set $\{a_j \mid \text{level of } a_j \text{ is } i\}$, the nodes in set $\{b_k \mid (a_j, b_k) \in E \text{ and } a_j \in A\}$ are assigned to level $i$+1. Notice that a topological level is defined only for acyclic di-graphs.

The following procedure computes the topological level of the nodes of a graph $G$ [73]. The $N = |V|$ nodes of $G$ are named 1, 2, …, $N$. $M = |E|$ is the number of edges.

**Procedure 3.3.3.-** Topological level of the nodes of an acyclic di-graph $G = \langle V, E\rangle$:

```
for $j \leftarrow 1$ to $N$, $c[j] \leftarrow 0$
for each edge $(i, j) \in E$, $c[j] \leftarrow c[j] + 1$
$Level[0] \leftarrow \{j \in V \mid c[j] = 0\}$
$k \leftarrow 0$
repeat
    $L \leftarrow Level[\mathrm{k}]$, $L' \leftarrow \emptyset$
    for each node $i \in L$ do
        for each edge $(i, j)$ do
            $c(j) \leftarrow c(j) - 1$
            if $c(j) = 0$ then $L' \leftarrow L' \cup \{j\}$
    $k \leftarrow k + 1$
    $Level[k] \leftarrow L'$
until $L = \emptyset$
return $Level$
```

The procedure first computes the in-degree $c(j)$ of node $j$. Level[0] is the set of root nodes. The level variable $k$ is set to zero. Iteratively: set $L$ is set to $Level[0]$ and $L'$ is set to empty; the in-degrees of the nodes to which the nodes in $L$ are connected are decreased by one; if the in-degree of a node becomes zero, the node is added to $L'$; the level variable is increased by one; and $Level[k]$ is set to $L$. The iterations continue until $L$ is empty.

The Procedure 3.3.3 finds a topological sort of $G$ in running time O($N + M$) [73].

Consider for example the AOC STG shown in Figure 3.3.10. For the procedures given in this section, the causality type and the time label of an edge are not important, and thus they are ignored. Notice that node $e$ is not connected to other nodes.

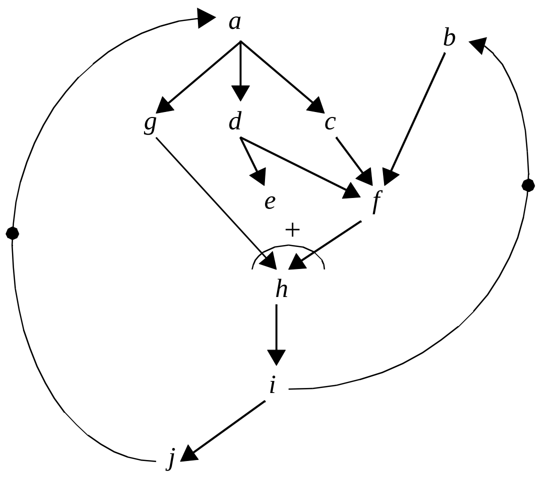


Figure 3.3.10 AOC signal transition graph.

A finite unfolding for $k = 2$, called 2-unfolding, of the AOC STG of Figure 3.3.10 is shown in Figure 3.3.11. The corresponding topological sort is given in Table 3.3.1.

A topological sort has the following property:

**Lemma 3.3.5.-** Let $<$ be the partial order induced by a topological sorted di-graph $G = \langle V, E \rangle$. If, for $a, b \in V$, $a < b$ and $a$ is assigned to level $l_a$ and $b$ is assigned to level $l_b$, then $l_a < l_b$.

**Proof.-** If $a < b$ then there is a set of nodes $\{e_i \in V \mid (a, e_1), (e_1, e_2), \ldots, (e_n, b) \in E\}$. From the computation of topological level, if $(e_i, e_j) \in E$ and $e_i$ is assigned to level $i$ then $e_j$ is assigned to level $i + 1$. A topological sort is only defined for acyclic di-graphs, then $l_a < l_b$. □

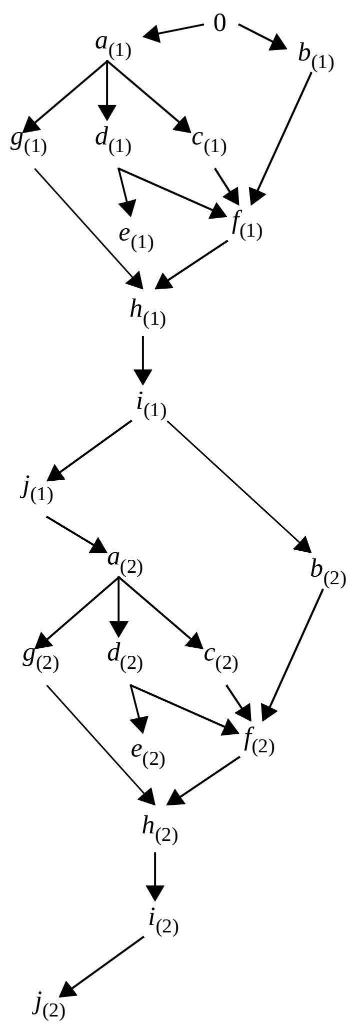


Figure 3.3.11   2-unfolding of the STG of Figure 3.3.10.

| **level** | **nodes** |
|---|---|
| 0 | 0 |
| 1 | $a_{(1)}, b_{(1)}$ |
| 2 | $c_{(1)}, d_{(1)}, g_{(1)}$ |
| 3 | $e_{(1)}, f_{(1)}$ |
| 4 | $h_{(1)}$ |
| 5 | $i_{(1)}$ |
| 6 | $j_{(1)}, b_{(2)}$ |
| 7 | $a_{(2)}$ |
| 8 | $c_{(2)}, d_{(2)}, g_{(2)}$ |
| 9 | $e_{(2)}, f_{(2)}$ |
| 10 | $h_{(2)}$ |
| 11 | $i_{(2)}$ |
| 12 | $j_{(2)}$ |

Table 3.3.1. Topological sort of the 2-unfolding of Figure 3.3.11.

Given a *k*-unfolding of an AOC STG, represented by the di-graph $G = \langle V, E\rangle$, where $V$ is the set of transitions and $E$ is the set of edges of the unfolding, the following procedure finds a fork transition of a selected pair of transitions $a$ and $b$ of $G$. The procedure is an application of a best-first search, where the topological levels of $G$ are used as the cost function [129].

**Procedure 3.3.4.-** Algorithm that finds a fork transition of a pair of transitions $a, b \in E$ of an acyclic di-graph $G = \langle V, E\rangle$, where $G$ has been topologically sorted.

```
$OPEN \leftarrow \{a, b\}$
while $|OPEN| > 1$ do
    sort $OPEN$ in descending order on the topological level
    $j \leftarrow$ first($OPEN$)
    $OPEN \leftarrow OPEN \setminus \{j\} \cup \{i \mid (i, j) \in E\}$
end while
return $OPEN$
```

The procedure first initializes set *OPEN* to contain the transitions *a* and *b*. Then until the size of *OPEN* is less than two, it selects as *j* the highest element of *OPEN*, removes *j* from *OPEN*, and adds all the direct ancestor transitions of *j* to *OPEN*.

At termination of Procedure 3.3.4, *OPEN* has either one or zero elements. We shall show in Theorem 3.3.7 that *OPEN* actually has always one element, and that such element is a fork transition. In the case of *k*-unfoldings of AOC STG's, the root transition 0 will be returned instead if there is no cycle-invariant fork transition.

We need the following lemma to prove the correctness of Procedure 3.3.4.

**Lemma 3.3.6.-** If $x$ and $y$ are two distinct fork transitions of transitions $a$ and $b$, with topological level $l_x$ and $l_y$ respectively, then $l_x \neq l_y$.

**Proof.-** We have to show that distinct fork transitions $x$ and $y$ cannot have the same level. Suppose they have the same level; by the definition of fork transition (refer to Definition 3.3.8) there exists a line containing say $x$ and $a$ but not $y$ otherwise that would mean by Lemma 3.3.5 that $l_x \neq l_y$; this is a violation of Definition 3.3.8 which requires that every line containing $a$ also contains $y$. Then if $x$ and $y$ are distinct either $x < y$ or $y < x$. □

The following theorem proves the correctness of Procedure 3.3.4.

**Theorem 3.3.7.-** When applied to a *k*-unfolding of an AOC STG, Procedure 3.3.4 terminates with a fork transition of transitions $a$ and $b$ or the root transition 0.

**Proof.-** Suppose that Procedure 3.3.4 does not terminate; that implies that transitions are continuously being added to *OPEN*; because the k-unfolding has a finite set of transitions, the only way of continuously adding transitions to *OPEN* is to add transitions previously removed; however when a transition is removed, it has the highest topological level in *OPEN*; that implies that none of the transitions left in *OPEN* is preceded by the removed transition; then it is not possible to add a removed transition again to *OPEN*, which is a contradiction, therefore Procedure 3.3.4 terminates. When Procedure 3.3.4 terminates the size of *OPEN* is either 0 or 1. Suppose it is 0, then the size of *OPEN* changed from 2 or greater to 0 in one iteration of the while loop; however *OPEN* is decreased only by removing one element, first(*OPEN*), which is a contradiction. Therefore when Procedure 3.3.4 terminates *OPEN* has size 1. Now suppose that the unique element $y$ of *OPEN* when Procedure 3.3.4 terminates is not a fork transition; by Lemma 3.3.2 one knows that there is at least one fork transition, let us call $x$ the closest fork transition to $a$ and $b$. Firstly suppose that $x > y$, then by Lemma 3.3.6 the topological level of $x$ is greater than the topological level of $y$; but OPEN is sorted in descending order so that $x$ would be in front of $y$; furthermore $x$ would be the only transition at topological level $l_x$ otherwise there would be a line containing either $a$ or $b$ but not $x$ thus violating Definition 3.3.8; then Procedure 3.3.4 would have returned $x$ which is a contradiction, then $x < y$. Secondly consider the case that $x < y$ (note that by both Lemma 3.3.6 and the definition of fork transition, if $x$ and $y$ are distinct, they cannot have the same topological order); note that then there must exist a path from $x$ to either $a$ or $b$ that does not include $y$ (refer to Figure 3.3.12); without loss of generality, let us say that it is a path from $x$ to $a$ given by $x \ldots v\, w \ldots a$; let us choose $v$ and $w$ such that their topological levels $l_v$ and $l_w$ respectively are $l_v < l_y \le l_w$, where $l_y$ is the topological level of $y$; such an assignment always exist, because there is also a path $x \ldots y \ldots a$, and together with path $x \ldots v\, w \ldots a$ imply the following relations between the topological levels of the transitions on each path: $l_x < l_y < l_a$ and $l_x < l_v < l_w < l_a$; because $l_y \le l_w$, $w$ must have been in *OPEN* before the termination of Procedure 3.3.4. But then when the direct ancestors of $w$ were added to *OPEN*, $v$ must have been added to *OPEN*, and because $l_v < l_y$, $v$ cannot be removed from *OPEN* before $y$, which is a contradiction. Therefore $y$ must be a fork transition. Finally Lemma 3.3.2 states that the root transition 0 is a fork transition for every pair of transitions, then if the only fork transition of transitions $a$ and $b$ is 0, Procedure 3.3.4 returns {0}. □

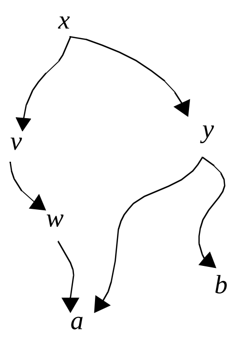


Figure 3.3.12 Construction for Theorem 3.3.7.

Before we can determine the time complexity of Procedure 3.3.4, we have to introduce the counting sort procedure adapted from [33]. The inputs of counting sort is a positive integer *k*, and a matrix *A* that contains |*A*| (possibly repeated) integers in the range [0, *M*] to be sorted. The result of counting sort is a matrix *B* of the same size as *A* which contains the elements of *A* sorted in descending order.

**Procedure 3.3.5.-** Counting sort

```
for $i \leftarrow 0$ to $M$,
        $c[i] \leftarrow 0$
for $j \leftarrow 1$ to $|A|$,
        $c[A[j]] \leftarrow c[A[j]] + 1$
for $i \leftarrow M-1$ to 0,
        $c[i] \leftarrow c[i+1] + c[i]$
for $j \leftarrow 1$ to $|A|$,
        $B[c[A[j]]] \leftarrow A[j]$
        $c[A[j]] \leftarrow c[A[j]] - 1$
```

Procedure 3.3.5 uses a counting array of size $M$+1 to store in $c[j]$ the number of integers in *A* which are greater than or equal to integer *j*. The first for loop initializes *c*. The second for loop counts the number of times integer *j* appears in *A*. The third for loop places in $c[j]$ the number of integers in *A* which are greater than or equal to integer *j*. The last for loop places the sorted integers of *A* in *B*. Notice that multiple occurrences of an integer *j* are allowed in *A*. The running time of counting sort is $O(|A| + M)$ [33]. We are going to use counting sort to sort *OPEN* in topological descending order; then $M$ is the maximum topological level of a $k$-unfolding, and both $M$ and $|A|$ are $O(|V|)$, where $V$ is the set of transitions of the $k$-unfolding. Therefore counting sort has a running time $O(|V|)$ to sort *OPEN* in Procedure 3.3.5.

The following lemma determines the running time of Procedure 3.3.4.

**Lemma 3.3.8.-** Procedure 3.3.4 has a running time $O(|V|^2)$.

**Proof.-** The operations inside the while loop of Procedure 3.3.4 have the following running times: We use counting sort to implement the sort operation and thus the sort operation has a running time of $O(|V|)$. Selecting the first element of *OPEN* can be done in $O(1)$. Removing $j$ from *OPEN* and adding the direct ancestors of $j$ to *OPEN* can be done in $O(1) + O(|V|)$, because the number of direct ancestors of a transition is $O(|V|)$. Thus the running time of one iteration of the while loop is $O(|V|)$. The while operation can be executed $O(|V|)$ times, because a transition can be added only once to *OPEN*, and when it is removed from *OPEN*, it cannot be added to *OPEN* again. □

Now we study under which conditions the fork transition returned by Procedure 3.3.4 is a cycle-invariant fork transition.

**Definition 3.3.13.-** An AOC STG $\Sigma = \langle N, Y, \lambda \rangle$ is called *simple* if every transition of the net is associated either with a distinct signal transition $a \in A(Y)$ or the silent transition $\varepsilon$, where $A(Y)$ is the alphabet of $Y$ (refer to Section 2.3.3).

**Theorem 3.3.9.-** If Procedure 3.3.4 terminates with a fork transition different from the root transition 0 when applied to find the fork transition of transitions $a_{(k)}$ and $b_{(k+\varepsilon)}$ of a $(k+\varepsilon)$-unfolding of a simple AOC STG, then the fork transition is cycle-invariant.

**Proof.-** Let us call $x_{(k-\lambda)}$ the fork transition of $a_{(k)}$ and $b_{(k+\varepsilon)}$ that is different from the root transition 0. Let $paths(x_{(k-\lambda)}, a_{(k)})$ be the set of all transition paths

$x_{(k\text{-}\lambda)} \ldots y_{(k\text{-}\lambda 1)}\, z_{(k\text{-}\lambda 2)} \ldots a_{(k)}$ from $x_{(k\text{-}\lambda)}$ to $a_{(k)}$, and let $path_i(x_{(k\text{-}\lambda)}, a_{(k)}) \in paths(x_{(k\text{-}\lambda)}, a_{(k)})$. Similarly let $paths(x_{(k\text{-}\lambda)}, b_{(k+\varepsilon)})$ be the set of all paths from $x_{(k\text{-}\lambda)}$ to $b_{(k+\varepsilon)}$. For each path $path_i(x_{(k\text{-}\lambda)}, a_{(k)})$, there is a (possibly not simple [58]) non-indexed path $path_i(x, a) = x \ldots y\, z \ldots a$ in the AOC STG. Now consider the $(k+\varepsilon+1)$-unfolding, which is constructed from the $(k+\varepsilon)$-unfolding by adding *one cycle* of the simple AOC STG (*i.e.*, one set of transitions and edges; notice that the signal transitions are not repeated). Consider first $a_{(k+1)}$; construct a path $path_i(x_{(k\text{-}\lambda+1)}, a_{(k+1)})$ by incrementing the indices $k$ of $path_i(x_{(k\text{-}\lambda)}, a_{(k)})$ by one. Path $path_i(x_{(k\text{-}\lambda+1)}, a_{(k+1)})$ must appear in the $(k+\varepsilon+1)$-unfolding because its corresponding non-indexed path $x \ldots y\, z \ldots a$ is a path of the AOC STG. Similarly construct $paths(x_{(k\text{-}\lambda+1)}, b_{(k+\varepsilon+1)})$ from $paths(x_{(k\text{-}\lambda)}, b_{(k+\varepsilon)})$. Moreover $x_{(k\text{-}\lambda+1)}$ must be a fork transition for $a_{(k+1)}$ and $b_{(k+\varepsilon+1)}$. Suppose $x_{(k\text{-}\lambda+1)}$ is not a fork transition of $a_{(k+1)}$ and $b_{(k+\varepsilon+1)}$, then there is a line containing $a_{(k+1)}$ or $b_{(k+\varepsilon+1)}$ but not $x_{(k\text{-}\lambda+1)}$; but that would imply that there is a line containing $a_{(k)}$ or $b_{(k+\varepsilon)}$ but not $x_{(k\text{-}\lambda)}$, which is a contradiction. If $x_{(k\text{-}\lambda+1)}$ is a fork transition then it is a cycle-invariant fork transition. □

Theorem 3.3.9 suggests a simple procedure to find a fork transition. For a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon\rangle$ of a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc}\rangle$, obtain the $(\varepsilon+1)$-unfolding, and apply Procedure 3.3.4 to find a fork transition of $t_{i(k)}$ and $t_{j(k+\varepsilon)}$. If the root transition 0 is returned, then add one cycle to the unfolding, otherwise the returned transition is the sought cycle-invariant fork transition. We have not found a tight upper-bound on the number of cycles required to guarantee that a cycle-invariant fork transition will be found. The number of transitions in the AOC STG is an upper-bound but we believe it is not tight. A more likely candidate, we conjecture, is the size of the cuts of the unfolding. A cut is a snapshot of the unfolding, and thus gives an indication of the degree of concurrency. For instance all the cuts of a sequential process (a total ordering) have size 1, and the cycle-invariant fork transition of two transitions $a$ and $b$, is $a$ if $a < b$, $b$ otherwise, which can be found in the $(\varepsilon+1)$-unfolding.

## 3.4 Summary

In this chapter we introduced the interface design problem, that arises when two components, that are to be interconnected to construct a system, require interface logic to be able communicate. The case in which protocol conversion is necessary is particularly interesting. Protocols can be described by interface specifications (refer to Chapter 2) which describe the internal operation and the desired environment of a component.

We have shown that one can view the interface design as the “merging” of two interface specifications with additional delay edges that generate the input signal transitions (the environment) of the interface specifications. This merged graph is called a complete graph. A complete graph consists of a live net, and a set of constraint rules. We developed the concept of time-consistency that checks if the set of constraints rules is satisfied by all possible executions of a complete graph.

Typically there is an infinite number of possible executions of a net (we are dealing with dense, or continuous, time). Thus it is important to find procedures that can determine if a complete graph is time-consistent. The concept of a cycle-invariant fork transition established that it is possible to check all the net executions by analyzing a finite unfolding of the net. Not all nets have cycle-invariant fork transitions; however we have found that all the interface specifications that we have studied possess that property. Moreover, nets that do not have cycle-invariant fork transitions may exhibit fundamentally different behavior if the time (delay) parameters of the net are slightly modified.

In the next chapter we shall present our probabilistic interface timing verification procedure which is heavily founded on the concepts developed in this and the previous chapters.

# Chapter 4

# Probabilistic interface timing verification

## 4.1 Introduction

Once the implementation of a digital system has been completed, one would like to check that the system operates correctly. One approach is to check one instance of the system, against a suite of tests which are intended to detect any malfunctioning. This approach suffers two major drawbacks. Firstly the chosen instance of the system may happen not to be representative of the production run. Secondly the suite of tests, unless exhaustive, may fail to detect a problem. If a problem eluded the checking procedure, eventually it would show up later on at a time when it would be very expensive to fix. Thus there is a market force that is driving system checking towards formal techniques.

Formal verification attempts to prove mathematically that the implementations of a system are going to function correctly under all circumstances. In this sense, it is equivalent to exhaustive system checking. It is no surprise that usually formal verification is computationally very expensive. And until recently [19, 31, 83], its application was limited to toy systems. However due to the potential benefits, and the development of efficient formal verification techniques, industry has been more receptive and is adopting formal verification techniques, at least in parallel with standard checking, in their design methodology.

The aim of this dissertation is to develop formal timing verification techniques to certify that a system composed of sub-components and interface logic satisfies timing constraints given in the interface specifications of the components. In the previous chapter we have presented a formal representation of the interface design, called a complete graph, that is amenable to formal timing verification. In this chapter we shall present a procedure that relies on probability theory which can be used to certify that an ensemble of instantiations represented by the complete graph satisfies all the timing constraints.

An important remark is that our approach uses a probabilistic, rather than a statistical, analysis, as differentiated in [100]. Statistical techniques are essentially Monte Carlo methods that are based on statistical sampling, and thus are close in spirit to simulation. A probabilistic technique propagates the probability measure directly through the system. Clearly a probabilistic approach is preferable, although sometimes it is not feasible due to the complexity of the problem. The main contribution of this chapter is the development of a probabilistic technique to compute time separation between transitions that propagate the probability density functions of the delays exactly.

## 4.2 Verification problem formulation

A complete graph is a formal description of a system composed of two components and interface logic (refer to Section 3.2.2). There are two parts of that description: the specification of the operation of the components that make up the system, and the description of the timing constraints that each component imposes on its environment for proper operation.

In the previous chapter we characterized time-consistency of a complete graph as a constraint satisfaction problem. A complete graph is time-consistent if and only if the set of all the timing constraints are satisfied by every possible execution of the net describing the operation of the system. We pointed out the difficulty involved in checking time-consistency, namely the infinite number of possible executions of the net. To avoid that problem we identified a sub-class of nets with a remarkable property whose behavior is *repeatable* (refer to Definition 3.3.12). Moreover, the existence of a cycle-invariant fork transition for each constraint, being a structural property, implies that it can be easily verified.

In this section we show that for repeatable nets there is a simple procedure to solve the interface timing verification problem that we define below.

**Definition 4.2.1.-** Given a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$ with associated timed STG $\Sigma_c = \langle N_c, Y_c, \lambda_c \rangle$ and set of constraint rules $C_{Nc}$, the interface timing verification problem is the problem of determining if every constraint rule $c_{ij} \in C_{Nc}$ is satisfied by every execution of the STG.

In other words, the goal of the interface timing verification problem is to find if a complete graph is time-consistent. From our discussion of Chapter 3, it is clear that the interface timing verification problem for the general case may involve checking an infinite number of executions. Fortunately, as mentioned above, for repeatable nets it is possible check if a constraint is satisfied in a finite number of steps.

Recall that a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon \rangle$ defines a time window $\Delta_{ij}$ with respect to the $k$-th occurrence of the constraining transition $t_i$ during which the ($k$+ε)-th occurrence of the constrained transition $t_j$ is allowed. The constraint rule is satisfied if for all occurrence indices $k > 0$, the following constraint equation for $c_{ij}$ is true:

$$\tau_{tj(k+\varepsilon)} - \tau_{ti(k)} \subseteq \Delta_{ij} \qquad \text{(Eq. 4.2.1)}$$

If a net is repeatable, then the time separation $\tau_{tj(k+\varepsilon)} - \tau_{ti(k)}$ is invariant for $k \geq M$. Thus it is possible to check Eq. 4.2.1 for $k \geq M$ by checking Eq. 4.2.1 for $k = M$. In Chapter 3, we identified a structural condition on the net, the presence of a *cycle-invariant fork transition* (refer to Definition 3.3.9), that guarantees repeatability. Furthermore, as discussed in Section 3.3.4, one can express the left-hand side of Eq. 4.2.1 with respect to a cycle-invariant fork transition $x$ as:

$$\tau_{tj(k+\varepsilon)\Diamond x} - \tau_{ti(k)\Diamond x} \subseteq \Delta_{ij} \qquad \text{(Eq. 4.2.2)}$$

where each of $\tau_{tj(k+\varepsilon)\Diamond x}$ and $\tau_{tj(k+\varepsilon)\Diamond x}$ are expressions containing linear/min/max terms on a set of random variables that defines the timing behavior of the net.

The set of random variables is characterized by a joint probability density function (or pdf). Our goal is to propagate the probability information of the random variables to the constraint equation. Let us denote the constraint equation by $z_{ij} = \tau_{tj(M+\varepsilon)} - \tau_{ti(M)}$. It is clear that $z_{ij}$ is a random variable. If the pdf of $z_{ij}$ is known, then the constraint rule is satisfied if and only if all the values of $z_{ij}$ lie within $\Delta_{ij}$.

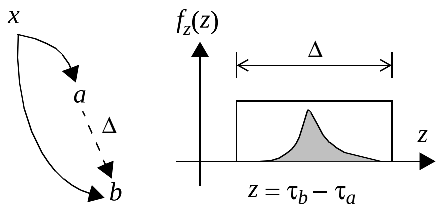


Figure 4.2.1 Checking if $z = \tau_b - \tau_a$ satisfies the constraint $\Delta$.

Consider for example Figure 4.2.1. There is a constraint rule from transition $a$ to transition $b$. Let us define $z = \tau_b - \tau_a$. If the pdf of $z$, $f_z(z)$, is given by the shaded area, and the interval $\Delta$ of the constraint “covers” the pdf, then the constraint is satisfied because all the values that the time separation $z$ can take are within the allowed time window $\Delta$. Let us formalize these ideas.

**Definition 4.2.2.-** Given a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon\rangle$ of a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc}\rangle$, the *k-th time separation* from $t_i$ to $t_j$ is the random variable $z_{ij}(k) = \tau_{tj(k+\varepsilon)} - \tau_{ti(k)}$, for a positive integer $k$.

**Definition 4.2.3.-** Given a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon\rangle$ of a complete graph $\Psi_c = \langle\Sigma_c, C_{Nc}\rangle$, the *cover of the k-th time separation* over $\Delta_{ij}$, denoted by $I_{ij}(k)$, is given by:

$$I_{ij}(k) \equiv \int_{\Delta_{ij}} f_{z_{ij}(k)}(z_{ij}(k)) dz_{ij}(k) \qquad \text{(Eq. 4.2.3)}$$

where $f_{z_{ij}(k)}(z_{ij}(k))$ is the probability density function of the $k$-th time separation.

The cover of the time separation is the area of the pdf of the time separation that is within the interval $\Delta_{ij}$.

To understand the significance of a cover, let us briefly introduce pdf's. The interpretation of a probability density function is as follows: Let $x$ be a random variable with pdf given by $f_x(x)$. For a sufficiently small $\Delta x$,

$$f_x(x_0)\, \Delta x \approx \text{Prob}\{x_0 < x < x_0 + \Delta x\} \qquad \text{(Eq. 4.2.4)}$$

The equality is approached as $\Delta x \rightarrow 0$. Thus the probability that $x$ takes a value in a small interval is proportional to $f_x(x)$. Clearly if the Probability is zero, then $f_x(x)$ is also zero. Thus the cover of the time separation is the probability that the time separation lies within the window $\Delta_{ij}$. Thus if $I_{ij}(k) = 1$ it is certain that the constraint is satisfied.

Without the existence of a cycle-invariant fork transition one would have to check an infinite number of covers.

**Definition 4.2.4.-** Given a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon\rangle$ of a complete graph $\Psi_c = \langle\Sigma_c, C_{Nc}\rangle$, and a cycle-invariant fork transition of $t_i$ and $t_j$ for $k \geq M$, the *invariant time separation* from $t_i$ to $t_j$ is the random variable $z_{ij} = \tau_{tj(n+\varepsilon)} - \tau_{ti(n)}$, for any $n \geq M$.

**Definition 4.2.5.-** Given a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon\rangle$ of a complete graph $\Psi_c = \langle\Sigma_c, C_{Nc}\rangle$ of transitions $t_i$ and $t_j$, the *cover of the invariant time separation* over $\Delta_{ij}$, denoted by $I_{ij}$, is the integral of the probability density function of the invariant time separation, denoted by $f_{z_{ij}}(z_{ij})$, over the constraint interval $\Delta_{ij}$:

$$I_{ij} \equiv \int_{\Delta_{ij}} f_{z_{ij}}(z_{ij}) dz_{ij} \qquad \text{(Eq. 4.2.5)}$$

**Theorem 4.2.1.-** Given a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon\rangle$ of a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc}\rangle$, and an invariant fork transition of $t_i$ and $t_j$ for $k \geq M$, the constraint equation is satisfied for $k \geq M$ if and only if the cover of the invariant time separation $I_{ij}$ is 1.

**Proof.-** If $I_{ij} = 1$, then all the values of $z_{ij}$ are within $\Delta_{ij}$. It is easy to show that the converse is also true. □

The constraint satisfaction procedure that solves the interface timing verification problem can be stated as follows:

**Procedure 4.2.1.-** Given a cycle-invariant complete graph, for every constraint rule $c_{ij} \in C_{Nc}$ do:

1. Compute the cover of the $k$-th time separation $I_{ij}(k)$ for $k = 1, 2, \ldots, M-1$.

2. Compute the cover of the invariant time separation $I_{ij}$.

3. The complete graph is time-consistent if and only if,

$$I_{ij}(k) = 1 \qquad \text{(Eq. 4.2.6)}$$

for $k = 1, 2, \ldots, M-1$, and

$$I_{ij} = 1 \qquad \text{(Eq. 4.2.7)}$$

**Theorem 4.2.2.-** Procedure 4.2.1 solves the interface timing verification problem.

**Proof.-** In a cycle-invariant complete graph there is a cycle-invariant fork transition for every pair of transitions involved in a constraint rule (refer to Definition 3.3.11). Then for each constraint rule there is an invariant time separation. According to Theorem 4.2.1, checking the cover of the invariant time separation $I_{ij}$ guarantees that the constraint rule is satisfied for $k \geq M$. Using a similar argument, checking the cover of the $k$-th time separation $I_{ij}(k)$ for $k = 1, 2, \ldots, M{-}1$, guarantees that the constraint rule is satisfied for $k = 1, 2, \ldots, M{-}1$. □

Thus Procedure 4.2.2 checks a finite number of covers to find if a cycle-invariant complete graph is time-consistent.

The rest of the chapter presents the techniques that we have developed to determine the pdf's of the invariant time separation. We provide several examples to illustrate our ideas. The chapter concludes with a discussion of a reliability analysis for the case that some of the covers are less than 1.

## 4.3 Probability distribution of functions of random variables

The time of occurrence of each transition in a complete graph is a random variable, as it is the time separation between two transitions. In this section we state some results from probability theory [105] that will be needed to compute the time separation between transitions.

### 4.3.1 One function of two random variables

Given two random variables $x$ and $y$ and a scalar function $g(a, b)$ of two real variables $a$ and $b$, the random variable $z = g(x, y)$ is formed. The probability density function of $z$ can be expressed in terms of the joint probability density function $f_{xy}(x, y)$ of the random variables $x$ and $y$, and the function $g$, as discussed below.

Let the random variable $z$ be a given value $Z$. Denote by $D_z$ the region of the $ab$ plane such that $g(a, b) \leq Z$, Then:

$$\{z \leq Z\} = \{g(x, y) \mid g(x, y) \leq Z\} = \{(x, y) \mid (x, y) \in D_z\} \qquad \text{(Eq. 4.3.1)}$$

The probability that the point $(X, Y)$ of the pair of random variables $(x, y)$ is in a region $D_z$ of the $ab$ plane is given by the following integral:

$$Prob\{(x, y) \in D_z\} = \iint_{D_z} f_{xy}(x, y)dx\ dy \qquad \text{(Eq. 4.3.2)}$$

The cumulative probability distribution function of $z$ is given by:

$$F_z(Z) = Prob\{z \leq Z\} = Prob\{(x, y) \in D_z\} \qquad \text{(Eq. 4.3.3)}$$

Thus, to determine $F_z(z)$ one has to find the region $D_z$ and evaluate the integral in Eq. 4.3.2.

The probability density function can be determined similarly. Let $\Delta D_z$ be the region of the $ab$ plane such that $Z \leq g(a, b) < Z + dZ$. Then,

$$\{Z \leq z < Z + dZ\} = \{(x, y) \mid (x, y) \in \Delta D_z\} \qquad \text{(Eq. 4.3.4)}$$

$$f_z(Z)\,dz = Prob\{Z \leq z < Z + dz\} = \iint_{\Delta D_z} f_{xy}(x, y)\,dx\ dy \qquad \text{(Eq. 4.3.5)}$$

## 4.3.2 Statistics of linear/max/min functions

In Section 3.3.4 we have shown that the left-hand side of the constraint equation (Eq. 4.2.1) when computed with respect to the fork transition of transitions $t_{i(k)}$ and $t_{j(k+\varepsilon)}$ is an expression on a subset of the random variables containing min, max and linear terms only. Thus, the constraint equation is an expression of the form $E_1 - E_2$, where each of the $E_i$ is recursively defined as follows in BNF notation:

$$
\begin{aligned}
E \leftarrow\ & nil\ | \\
& \tau_i\ | \\
& E + \tau_i\ | \\
& \max(E, \tau_i)\ | \\
& \min(E, \tau_i)
\end{aligned}
$$

where *nil* means that $E$ is empty. This occurs if either $t_{i(k)}$ or $t_{j(k+\varepsilon)}$ coincide with the fork transition (only one of them could be the fork transition, assuming that $t_{i(k)}$ and $t_{j(k+\varepsilon)}$ are distinct).

In this section we summarize the application of Eqs. 4.2.3 and 4.2.6 to the special functions that can appear in a constraint equation derived from a fork transition.

Let $x$ and $y$ be two random variables with joint pdf $f_{xy}(x, y)$.

**1.** $\boldsymbol{z = x + y}$

The region $D_z$ of the $ab$ plane such that $a + b \le Z$, and the region $\Delta D_z$, given by $Z \le a + b < Z + dZ$ are shown in Figure 4.3.1.

After integrating over the corresponding region, Eqs. 4.2.3 and 4.2.6 can be written as:

$$F_z(z) = \int_{-\infty}^{\infty} \int_{-\infty}^{z-y} f_{xy}(x, y)dx\ dy \qquad \text{(Eq. 4.3.6)}$$

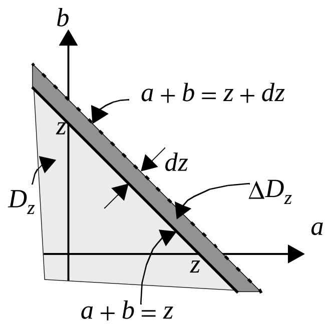


Figure 4.3.1 Probability regions for $z = x + y$.

$$f_z(z) = \int_{-\infty}^{\infty} f_{xy}(z - y, y)dy \qquad \text{(Eq. 4.3.7)}$$

If $x$ and $y$ are independent random variables, *i.e.*, $f_{xy}(x, y) = f_x(x) \cdot f_y(y)$, then Eq. 4.3.7 becomes the convolution of the individual pdf's:

$$f_z(z) = \int_{-\infty}^{\infty} f_x(z - y) f_y(y) dy \qquad \text{(Eq. 4.3.8)}$$

**2. $z = y - x$**

The region $D_z$ of the $ab$ plane such that $b - a \le Z$, and the region $\Delta D_z$, given by $Z \le b - a < Z + dZ$ are shown in Figure 4.3.2.

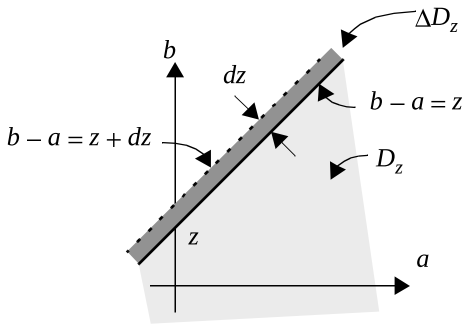


Figure 4.3.2 Probability regions for $z = x - y$.

After integrating over the corresponding region, Eqs. 4.2.3 and 4.2.6 can be written as:

$$F_z(z) = \int_{-\infty}^{\infty}\int_{-\infty}^{z+x} f_{xy}(x, y)dy\ dx \qquad \text{(Eq. 4.3.9)}$$

$$f_z(z) = \int_{-\infty}^{\infty} f_{xy}(x, x+z)dx \qquad \text{(Eq. 4.3.10)}$$

If $x$ and $y$ are independent random variables then Eq. 4.3.10 becomes the cross-correlation of the individual pdf's:

$$f_z(z) = \int_{-\infty}^{\infty} f_x(x)f_y(x+z)dx \qquad \text{(Eq. 4.3.11)}$$

**3. $z = \max(x, y)$**

The region $D_z$ of the $ab$ plane such that $\max(a, b) \le Z$, and the region $\Delta D_z$, given by $Z \le \max(a, b) < Z + dZ$ are shown in Figure 4.3.3.

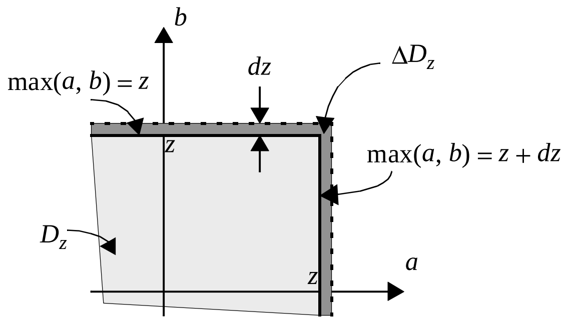


Figure 4.3.3 Probability regions for $z = \max(x, y)$.

After integrating over the $D_z$, Eq. 4.2.3 can be written as:

$$F_z(z) = F_{xy}(z, z) \qquad \text{(Eq. 4.3.12)}$$

and $f_z(z)$ is obtained after differentiating Eq. 4.3.12:

$$f_z(z) = \frac{d}{dz} F_{xy}(z, z) \qquad \text{(Eq. 4.3.13)}$$

where

$$F_{xy}(x, y) = \int_{-\infty}^{y} \int_{-\infty}^{x} f_{xy}(\alpha, \beta)\, d\alpha\, d\beta \qquad \text{(Eq. 4.3.14)}$$

is the cumulative joint distribution function of $x$ and $y$.

If $x$ and $y$ are independent random variables, Eqs. 4.3.13 becomes:

$$f_z(z) = f_x(z) F_y(z) + f_y(z) F_x(z) \qquad \text{(Eq. 4.3.15)}$$

**4. $z = \min(x, y)$**

The region $D_z$ of the $ab$ plane such that $\min(a, b) \le Z$, and the region $\Delta D_z$, given by $Z \le \min(a, b) < Z + dZ$ are shown in Figure 4.3.4.

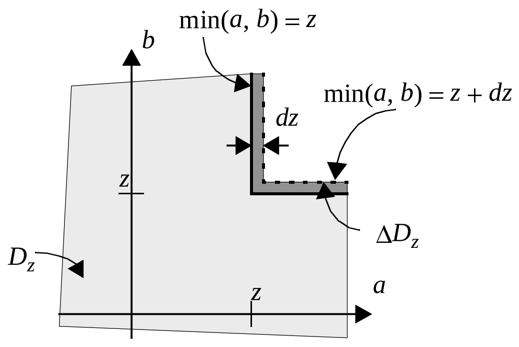


Figure 4.3.4 Probability regions for $z = \min(x, y)$.

After integrating over the $D_z$, Eq. 4.2.3 can be written as:

$$F_z(z) = F_x(z) + F_y(z) - F_{xy}(z, z) \qquad \text{(Eq. 4.3.16)}$$

and $f_z(z)$ is obtained after differentiating Eq. 4.3.16:

$$f_z(z) = \frac{d}{dz}(F_x(z) + F_y(z) - F_{xy}(z, z)) \qquad \text{(Eq. 4.3.17)}$$

If $x$ and $y$ are independent random variables, $f_z(z)$ is commonly expressed in terms of the reliability function $R_x(x)$ defined as:

$$R_x(X) = \text{Prob}\{x \ge X\} = 1 - F_x(X) \qquad \text{(Eq. 4.3.18)}$$

Then Eq. 4.3.17 becomes:

$$f_z(z) = f_x(z)R_y(z) + f_y(z)R_x(z) \qquad \text{(Eq. 4.3.19)}$$

Notice that the expressions for the +, max, and min operators are commutative. It is easy to show that they are also associative.

## 4.3.3 Point conditional probability

The results of the previous section can be used directly to compute the time separations $\tau_{tj(k+\varepsilon)\Diamond x}$ and $\tau_{ti(k)\Diamond x}$ with respect to a fork transition $x$. However to compute $\tau_{tj(k+\varepsilon)\Diamond x} - \tau_{ti(k)\Diamond x}$ we have to address the issue of the so-called reconvergence fan-out [28]. The effect of reconvergence fan-out is due to the presence of common random variables in expressions $\tau_{tj(k+\varepsilon)\Diamond x}$ and $\tau_{ti(k)\Diamond x}$. In a correct analysis, the value used for a common random variable that appears in both expressions should be the same. As we shall discuss below, even if the delay random variables are independent, the topology of a net may introduce correlation.

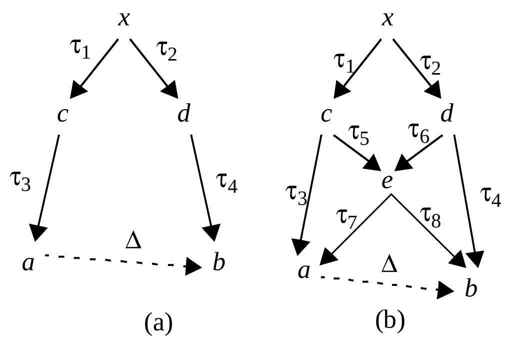


Figure 4.3.5 Two partial unfoldings (a) *a* and *b* independent; (b) *a* and *b* correlated.

Consider for example the two unfoldings shown in Figure 4.3.5. For the sake of clarity, let us assume that the random variables $\tau_i$ associated to the edges of the unfoldings are independent. To check constraint $\Delta$, one has to compute the time separation from transition *a* to transition *b* with respect to the fork transition *x*, given by:

$$\tau_{b\Diamond x} - \tau_{a\Diamond x} \subseteq \Delta \qquad \text{(Eq. 4.3.20)}$$

where both expressions $\tau_{a\Diamond x}$ and $\tau_{b\Diamond x}$ are functions of the $\tau_i$.

For the unfolding shown in Figure 4.3.5a, the time separation $\tau_{b\Diamond x} - \tau_{a\Diamond x}$ is simply $z = \{\tau_2 + \tau_4\} - \{\tau_1 + \tau_3\}$. Application of Eqs. 4.3.8 and 4.3.11 obtains the pdf of $z$ from the pdf's of the random variables $\tau_1$ to $\tau_4$. An interesting property of this unfolding is that $\tau_{b\Diamond x} = \{\tau_2 + \tau_4\}$ and $\tau_{a\Diamond x} = \{\tau_1 + \tau_3\}$ are two independent random variables.

For the unfolding shown in Figure 4.3.5b, the time separation $z = \tau_{b\Diamond x} - \tau_{a\Diamond x}$ is given by the following expression:

$$z = \max(\tau_2 + \tau_4, \tau_8 + \tau_{e\Diamond x}) - \max(\tau_1 + \tau_3, \tau_7 + \tau_{e\Diamond x}) \qquad \text{(Eq. 4.3.21)}$$

where $\tau_{e\Diamond x} = \max(\tau_1 + \tau_5, \tau_2 + \tau_6)$ is the occurrence time of transition *e* with respect to the fork transition *x*.

Notice that $\tau_{e\Diamond x}$ appears in both max terms. It is clear that to compute $z$ for a particular set of values of the random variables $\tau_i$, one has to make sure that the same value for $\tau_{e\Diamond x}$ is used in both max terms. This implies that the two max terms in Eq. 4.3.21 are not independent of one another, even if the $\tau_i$ are independent. Thus common random variables $\tau_i$ in expressions $\tau_{a\Diamond x}$ and $\tau_{b\Diamond x}$ introduce correlation in random variable $z$.

To take into account the dependency introduced by common terms we use point conditional probability [106]. Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be two vectors of random variables $\tau_i$ and let $f_{\boldsymbol{xy}}(\boldsymbol{x}, \boldsymbol{y})$ be a function of $\boldsymbol{x}$ and $\boldsymbol{y}$. The point conditional probability of $f_{\boldsymbol{xy}}(\boldsymbol{x}, \boldsymbol{y})$ provided that $\boldsymbol{x} = \boldsymbol{X}$, written $f_{\boldsymbol{y}|\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{y})$, is given by the following equation:

$$f_{\boldsymbol{y}|\boldsymbol{x}}(\boldsymbol{x}, \boldsymbol{y}) = \frac{f_{\boldsymbol{xy}}(\boldsymbol{x}, \boldsymbol{y})}{f_{\boldsymbol{x}}(\boldsymbol{x})} \qquad \text{(Eq. 4.3.22)}$$

if $f_{\boldsymbol{x}}(\boldsymbol{x}) \neq 0$.

Consider again the time separation $z = \tau_{b\Diamond x} - \tau_{a\Diamond x}$. To simplify the notation in the following paragraphs, let us denote $a = \tau_{a\Diamond x}$, and $b = \tau_{b\Diamond x}$. To compute $z$, one can use Eq. 4.3.10, which necessitates the joint probability distribution $f_{ab}(a, b)$.

To determine the joint pdf $f_{ab}(a, b)$ we use the following procedure based on point conditional probability (Eq. 4.3.22). Let us denote by $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ two vectors of random variables formed from the set of random variables $\tau_i$ on which random variables $a$ and $b$ respectively depend. The dependency of random variables $a$ and $b$ on vectors $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ can be explicitly written as $a = a(\boldsymbol{x}_a)$, and $b = b(\boldsymbol{x}_b)$. Let us denote by $\boldsymbol{x}_\cap$ the vector containing all the variables common to both $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$.

The probability density function $f_{ab|\boldsymbol{x}\cap}(a, b, \boldsymbol{x}_\cap)$ describes the joint pdf of $a$ and $b$ for a given value of $\boldsymbol{x}_\cap$. For a fixed $\boldsymbol{x}_\cap$, $f_{a|\boldsymbol{x}\cap}(\boldsymbol{x}_a, \boldsymbol{x}_\cap)$ and $f_{b|\boldsymbol{x}\cap}(\boldsymbol{x}_b, \boldsymbol{x}_\cap)$ are independent of one another. Thus one can write $f_{ab|\boldsymbol{x}\cap}(a, b, \boldsymbol{x}_\cap)$ as:

$$f_{ab|\boldsymbol{x}\cap}(a, b, \boldsymbol{x}_\cap) = f_{a|\boldsymbol{x}\cap}(a, \boldsymbol{x}_\cap)\, f_{b|\boldsymbol{x}\cap}(b, \boldsymbol{x}_\cap) \qquad \text{(Eq. 4.3.23)}$$

In Section 4.5 we shall show that it is straightforward to compute $f_{a|\boldsymbol{x}\cap}(\boldsymbol{x}_a, \boldsymbol{x}_\cap)$ and $f_{b|\boldsymbol{x}\cap}(\boldsymbol{x}_b, \boldsymbol{x}_\cap)$. Finally, the joint pdf of $a$ and $b$ is given by:

$$f_{ab}(a, b) = \int_{-\infty}^{\infty} \dots \int_{-\infty}^{\infty} f_{ab\boldsymbol{x}_\cap}(b, a, \boldsymbol{x}_\cap) d\boldsymbol{x}_\cap \qquad \text{(Eq. 4.3.24)}$$

where (using Eq. 4.3.22):

$$f_{ab\boldsymbol{x}_\cap}(a, b, \boldsymbol{x}_\cap) = f_{ab|\boldsymbol{x}_\cap}(a, b, \boldsymbol{x}_\cap) f_{\boldsymbol{x}_\cap}(\boldsymbol{x}_\cap) \qquad \text{(Eq. 4.3.25)}$$

Thus point conditional probability allows us to take into account the reconvergence fan-out due to common delays in the occurrence times of transitions $a$ and $b$ with respect to fork transition $x$. Once Eq. 4.3.24 has been obtained, the application of Eq. 4.3.10 yields the desired pdf of the time separation from $a$ to $b$.

## 4.4 Reliability analysis

In this section we take a look at the case when a complete graph is not time-consistent. As we shall discuss, it is for this case that our procedure can provide invaluable insight.

In Section 4.2 we introduced the cover of the time separation, which computes for a constraint $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon\rangle$ the area under the pdf of the time separation between the transitions over the interval $\Delta_{ij}$. Our interface timing verification procedure checks that the cover of the time separation is 1, that means that every possible value that the time separation can take lies within the constraint window $\Delta_{ij}$. In this section we extend the yes/no answer produced by Procedure 4.2.1 so that a designer can have, in case that the complete graph is not time-consistent, a measure of the deviation from time-consistency.

**Definition 4.4.1.-** The set of reliability factors of a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon\rangle$ of a cycle-invariant complete graph $\Psi_c = \langle \Sigma_c, C_{Nc}\rangle$ is the set of covers of the time separation $\{I_{ij}(k) \mid k = 1, 2, \ldots, M-1\} \cup \{I_{ij}\}$.

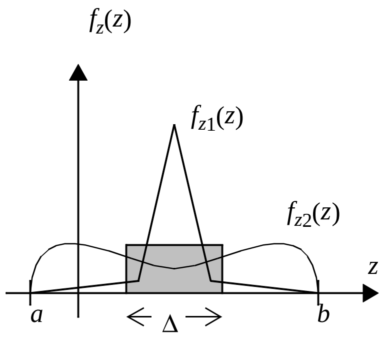


Figure 4.4.1 Reliability factor.

In essence, a cover of a time separation associated to a constraint represents the probability that the constraint is satisfied by the net. The verification procedure of Section 4.2 requires that the constraint be satisfied 100%. By keeping the values of the covers, rather than just checking if they are 1 or not, it is possible to qualify a design. Consider for example two possible pdf's of a time separation for the same constraint shown in Figure 4.4.1. Neither of the pdf's satisfy the constraint. However, the probability that $f_{z1}(z)$ would violate the constraint is significantly smaller than the probability that $f_{z2}(z)$ would violate the constraint.

This is particularly important when the joint pdf that characterizes the set of delay random variables describes an ensemble of components. The reliability factor would provide some indication about the reliability of a design. For instance, an estimation of the number of finished boards that would be returned. Moreover, the $[\tau_{min}, \tau_{max}]$ are likely approximations of the actual ranges such that, say 99.5% of the components show a delay in that range.

The reliability factor of a constraint is actually a set of values. To determine a reliability figure of a design, one may want to develop a strategy to combine the reliability factors of all constraints of the complete graph into a single figure. A simple approach is to use the minimum of the values of the reliability factor set to represent a constraint and then choose the minimum of the constraints' values to represent the design. However different applications may require different strategies and we have preferred to leave this decision to the designer.

## 4.5 Examples

In this section we look at some examples to illustrate our probabilistic timing interface verification procedure. Our first example considers the case in which all the delays are independent. A second example considers the case in which some delays are not independent. Then we analyze a read interface design involving the SHARC DSP and an SRAM memory chip (the interface specifications of these two components, which contain correlation data, were presented in Chapter 2). Finally we explore the relation between the traditional interval representation of delays and our probabilistic representation by analyzing some special cases.

### 4.5.1 Example with independent random variables

In this example we shall explore the effect of using different pdf's. We have chosen uniform and Gaussian functions as representative pdf's. For the sake of simplicity, the random variables are assumed to be independent. In the following example we shall look into the effect of correlation.

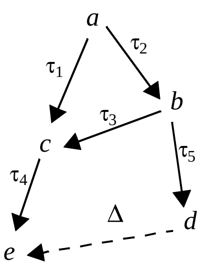


Figure 4.5.1 Constraint satisfaction by a net unfolding.

Consider the net unfolding shown in Figure 4.5.1. The time separation from $d$ to $e$ with respect to the fork transition $a$ is the random variable $z = \tau_e - \tau_d$, where $\tau_e = max(\tau_1, \tau_2 + \tau_3) + \tau_4$, and $\tau_d = \tau_2 + \tau_5$. The delays are assumed to be independent, and their projections (refer to Section 2.5.4) can be described by the following intervals: for $\tau_1$, [0, 90]; for $\tau_2$, [0, 100]; and or $\tau_3$, $\tau_4$, and $\tau_5$, [10, 20].

First we tackle the task of finding the joint pdf that characterizes the delays of the unfolding. Because the delays are independent, the joint pdf has the following form:

$$f_{\tau 1 \tau 2 \tau 3 \tau 4 \tau 5}(\tau_1, \tau_2, \tau_3, \tau_4, \tau_5) = f_{\tau 1}(\tau_1)\, f_{\tau 2}(\tau_2)\, f_{\tau 3}(\tau_3)\, f_{\tau 4}(\tau_4)\, f_{\tau 5}(\tau_5) \qquad \text{(Eq. 4.5.1)}$$

where $f_{\tau i}(\tau_i)$ is the pdf of the independent random variable $\tau_i$.

As mentioned in Section 2.5.4, the projection of a pdf describes an infinite number of possible pdf's. First let us assume that it is known that the pdf's of the $\tau_i$'s are uniform. That defines unequivocally the pdf's given their projections. For example, the pdf of $\tau_1$ is shown in Figure 4.5.2.

The next step is to determine the pdf of the time separation $z = \tau_{e\Diamond a} - \tau_{d\Diamond a}$ from $d$ to $e$ with respect to $a$. The expressions for each of the terms involved in $z$ are

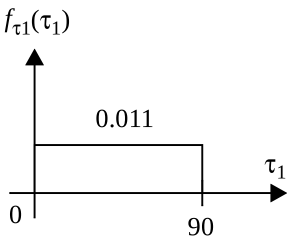


Figure 4.5.2 Probability density function $f_{\tau 1}(\tau_1)$ of $\tau_1$.

$\tau_{e\Diamond a} = \max(\tau_1, \tau_2 + \tau_3) + \tau_4$ and $\tau_{d\Diamond a} = \tau_2 + \tau_5$. The random variable $\tau_2$ is common to both terms, thus one must use point conditional probability as discussed in Section 4.3.3 to find $f_{\tau d\Diamond a|\tau 2}(\tau_{d\Diamond a}, \tau_2)$ and $f_{\tau e\Diamond a|\tau 2}(\tau_{e\Diamond a}, \tau_2)$ to obtain the joint pdf $\mathrm{f}_{\tau d\Diamond a \tau e\Diamond a}(\tau_{d\Diamond a}, \tau_{e\Diamond a})$ required in Eq. 4.3.10 to compute $f_z(z)$. Figure 4.5.3 shows the intermediate step to find the conditional pdf of $x$ given a fixed $\tau_2$ where $x = max(\tau_1, \tau_2 + \tau_3)$.

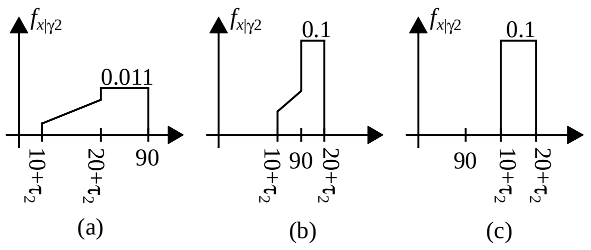


Figure 4.5.3 Probability density function of $f_{x|\tau 2}(x, \tau_2)$, $x = max(\tau_1, \tau_2 + \tau_3)$, for: (a) $0 \leq \tau_2 \leq 70$; (b) $70 \leq \tau_2 \leq 80$; (c) $80 \leq \tau_2 \leq 100$.

The resulting pdf of $z$ is shown in Figure 4.5.4. The projection of $f_z(z)$ is the interval [0, 100] (not clearly shown in the plot due to resolution). Hence any constraint interval $\Delta$ that includes [0, 100] is satisfied by the net unfolding.

Let us consider now the case in which the pdf's of the $\tau_i$ are Gaussian distributions, such that 99.7% of their values lie within the above intervals, *i.e.*, within three times the standard deviation $\sigma$ to each side of the mean. For example, the mean and standard deviation of the Gaussian distribution corresponding to $\tau_1$ are 45 and 15 respectively. The

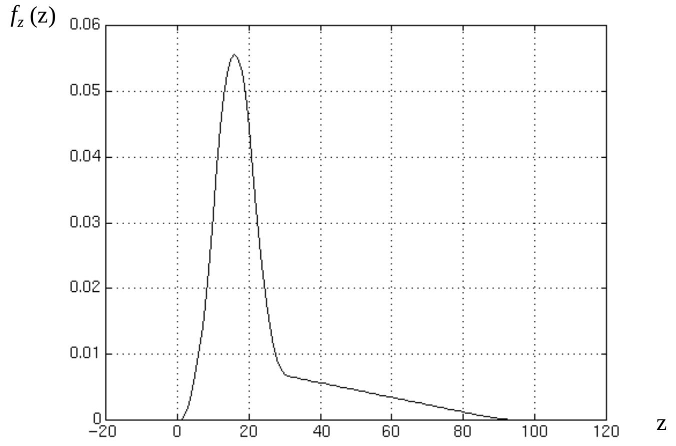


Figure 4.5.4 Probability density distribution of $z = \tau_{e\lozenge a} - \tau_{d\lozenge a}$: uniform pdf's.

resulting pdf of $z$ is shown in Figure 4.5.5. Notice the similarities of this pdf to the pdf shown in Figure 4.5.4. This is due to the filtering effect of the operations performed on the random variables (*i.e.*, convolution, and correlation).

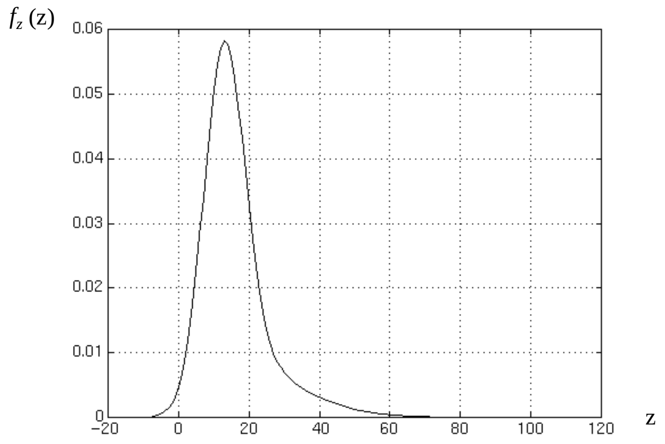


Figure 4.5.5 Probability density distribution of $z = \tau_{e\lozenge a} - \tau_{d\lozenge a}$: Gaussian pdf's.

The area under $f_z(z)$ within the interval [0, 100] (*i.e.*, $F_z(100) – F_z(0)$) is 0.997, therefore a constraint interval $\Delta = [0, 100]$ is satisfied by the unfolding in 99.7% of its executions.

In both cases, for uniform and Gaussian pdf's, it is clear that a tighter constraint would be satisfied by the unfolding with a lower probability. To investigate the effect of the reliability factor $r$ on a constraint for the time-separation distribution shown in Figure 4.5.5, consider the variable constraint $[0, d_{max}]$. Figure 4.5.6 shows the plot of $r$ vs. $d_{max}$. Note the two main regions in the plot: for low values of $d_{max}$, a variation in $d_{max}$ causes a noticeable change in $r$; starting from $d_{max} \approx 25$, the slope asymptotically diminishes towards zero. For $d_{max} \geq 80$, $r \approx 1$.

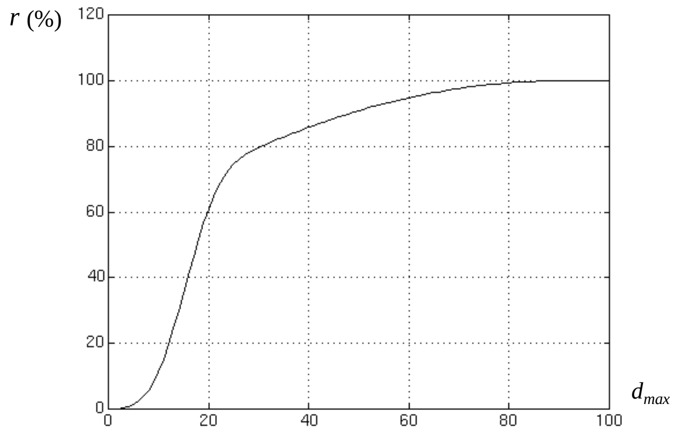


Figure 4.5.6 Reliability figure *r*.

## 4.5.2 Example with correlated random variables

In our second example we investigate the impact of ignoring correlation data in the verification procedure.

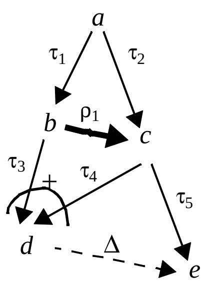


Figure 4.5.7 Partial unfolded graph with correlation between transitions *b* and *c*.

Consider for example the partial unfolded graph shown in Figure 4.5.7. Transition *d* will occur as soon as the first of *b* or *c* occurs (OR causality). Suppose the delays have projections with the following lower/upper bounds: $\tau_1$ and $\tau_2$ in [0,20]; $\tau_3$ in [10,50]; $\tau_4$ in [0,60]; and $\tau_5$ in [10,30]. Moreover $\tau_1$ and $\tau_2$ are related by a correlation edge $\rho_1$ whose range is [-5,5]. Correlation $\rho_1$ states that for all possible values of $\tau_1$ and $\tau_2$, $\tau_1 - \tau_2 \in \rho_1$, which can be written as $-5 \le \tau_1 - \tau_2 \le 5$. Note that non-causality is not implied by $\rho_1$, as transitions *b* and *c* always occur after transition *a*. The other delays are assumed to be independent. Finally let us assume that the pdf's are uniformly distributed so that we can reconstruct a joint pdf for the delays from the given projections. The joint probability density function $f_{\tau1\tau2}(\tau_1,\tau_2)$ is shown in Figure 4.5.8. The joint pdf that characterizes the delays of the unfolding is thus given by:

$$f_{\tau1\tau2\tau3\tau4\tau5}(\tau_1, \tau_2, \tau_3, \tau_4, \tau_5) = f_{\tau1\tau2}(\tau_1, \tau_2)\, f_{\tau3}(\tau_3)\, f_{\tau4}(\tau_4)\, f_{\tau5}(\tau_5) \qquad \text{(Eq. 4.5.2)}$$

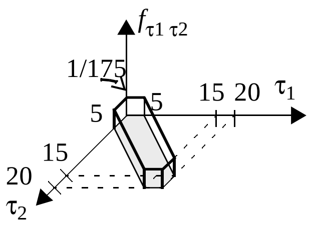


Figure 4.5.8 Joint probability density function of delays $\tau_1$ and $\tau_2$.

To check constraint $\Delta$ one must find the time separation from *e* to *d*, which is given by the random variable $z = \tau_{d\Diamond a} - \tau_{e\Diamond a}$, where $\tau_{d\Diamond a} = min\ (\tau_1+\tau_3,\ \tau_2+\tau_4)$, and $\tau_{e\Diamond a} = \tau_2 + \tau_5$, relative to the fork transition *a*. Figure 4.5.9 shows the probability density function of *z*.

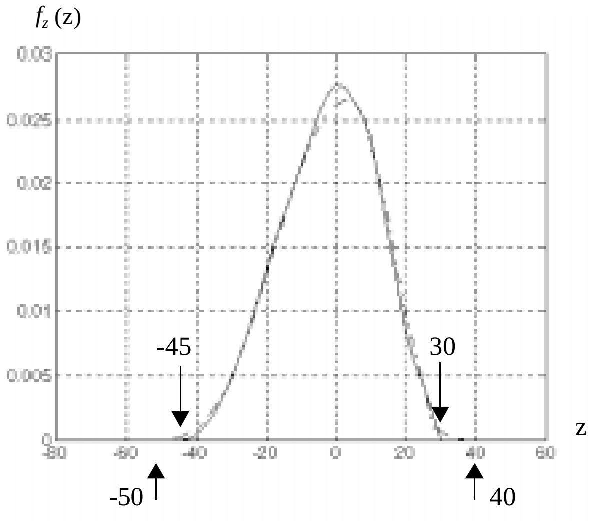


Figure 4.5.9 Probability density distribution of $z = \tau_{e\Diamond a} - \tau_{d\Diamond a}$: continuous line, with correlation; dashed line, without correlation.

The bounds on the time separation *z* from *d* to *e* are [-45,30]. Therefore any constraint $\Delta$ such that $[-45,30] \subseteq \Delta$ would be satisfied. If $\Delta$ is not satisfied, the probability that the constraint can be violated by the interface circuit can be determined by computing the cover of the time separation. Suppose that $\Delta = [-30,30]$; then $I = 0.9703$, that is $\Delta$ would be violated in about 3% of the executions.

If correlation $\rho_1$ is not taken into consideration, it can be found that the bounds on *z* are [-50,40] (refer to Figure 4.5.9) thus yielding pessimistic results.

As a side remark, we comment on the simpler problem of obtaining just the bounds on the time separation of the events related by the constraint. One might think that it could be possible to determine the constraint bounds by computing the time separation for every combination of the extreme (*i.e.* min/max) values of the delays $\tau_i$. This in general does not hold. For instance in the previous example the -45 bound occurs only in the following cases:

1. $\tau_1 + \tau_3 \geq \tau_2 + \tau_4$, and $\tau_4 = 55$, $\tau_5 = 10$.

2. $\tau_1 + \tau_3 \leq \tau_2 + \tau_4$, and $\tau_1 = 5$, $\tau_2 = 0$, $\tau_3 = 50$, $\tau_5 = 10$ or $\tau_1 = 25$, $\tau_2 = 5$, $\tau_3 = 50$, $\tau_5 = 10$.

In general taking correlation into consideration tightens up the bounds on the time separation $z = \tau_{d\Diamond a} - \tau_{e\Diamond a}$. To see that consider the linear projections shown in 4.5.10. If the random variables are independent the linear projection is given by the rectangular projection, indicating that the values that each r.v. can take does not depend on the values of the other one. If they are correlated, this is described by the hexagonal shape, indicating that there is some depedency between the values of the two r.v.

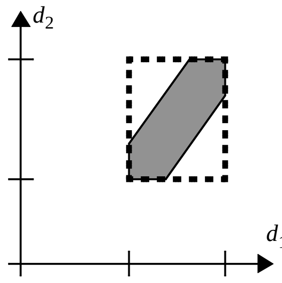


Figure 4.5.10 Two linear projections of pdf's of two random variables: independent (dotted boundary) and correlated (gray area).

Because the projection for the case of independent r.v. properly contains the projection for the case of correlated r.v., the possible values for the time separation $z = \tau_{d\Diamond a} - \tau_{e\Diamond a}$ for the latter case is always a subset of the possible values for the time separation $z = \tau_{d\Diamond a} - \tau_{e\Diamond a}$ for the former case.

### 4.5.3 Memory read interface example

In our last example we shall examine a read interface design involving a DSP and a RAM device. The interface specifications of these two components were covered in Chapter 2, where it was shown that the operational part of the interface specification included time correlation data, *i.e.*, some of the delays were not independent.

Consider a high-performance system comprising the SHARC DSP and some fast, no-wait state, RAM. The DSP's clock cycle is 25 nanoseconds. The designer of this system would like to choose a ram chip that is not an overkill, because it will increase the cost of the system, but at the same time she would like to certify that the system will not fail. Our interface timing verification procedure is a tool that can help her in making the right choices.

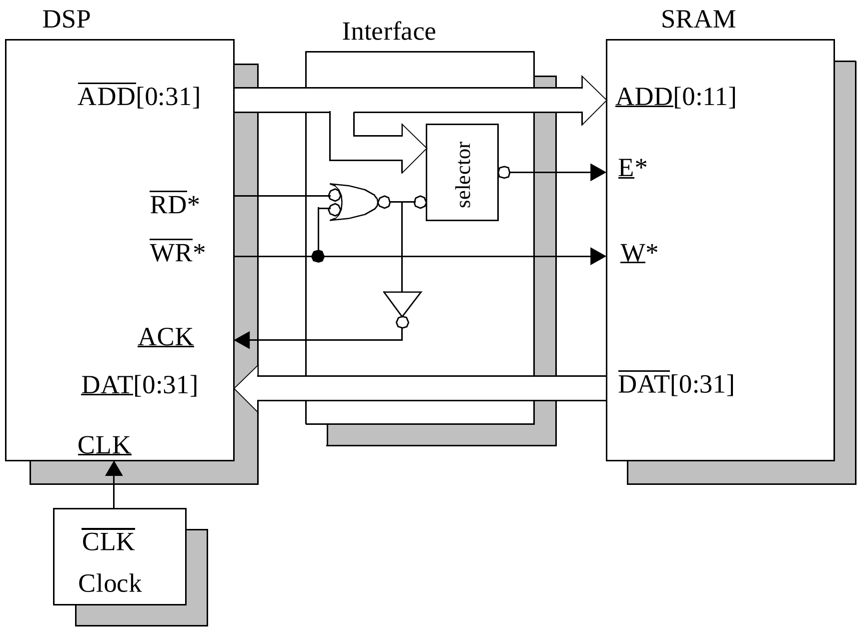


Figure 4.5.11 Interface read design.

Figure 4.5.11 shows a block diagram (a structural view) of a system composed of a DSP, an SRAM component and interface logic. The interface logic consists of a selector that generates the enable signal of the SRAM, and of the generation of the *ack* signal. Due to the design requirements, the *ack* signal is generated as soon as possible to avoid incurring wait states.

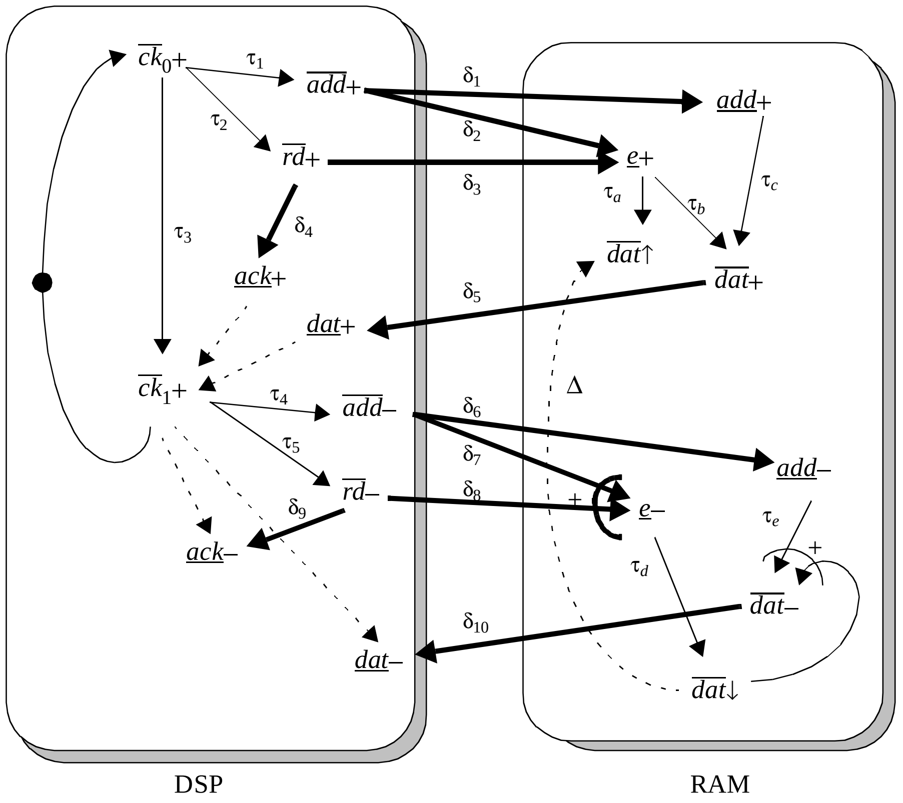


Figure 4.5.12 Complete graph representing the interface read design.

The complete graph describing the read cycle of this design is shown in Figure 4.5.12. Notice that the complete graph provides a behavioral view of the design. The complete graph contains the interface specifications for the read cycle of the DSP and the SRAM. The semantic specification that defines a data transfer, given by the sequence *dat*+ → <u>*dat*</u>+ → *dat*– → <u>*dat*</u>– -> *dat*+, is already included in the interface specifications. Finally there are some additional delay edges, shown as thick lines, which correspond to the interface logic. For example the two delays $\delta_2$ and $\delta_3$ represent the propagation delays from the address lines and the address strobe signal *rd* in the DSP, through the selection block in the interface block, to the enable signal *e* in the SRAM. Notice that those delays are able to represent also interconnection delays; this is very important in the new sub-micron technologies for which interconnection delays are comparable to gate delays, and thus not negligible.

To certify the design one must check that each one of the constraints is satisfied. In this example we shall focus our attention on one constraint, which is usually overlooked: the back-to-back cycle constraint that monitors that the data lines in the previous cycle are tri-stated before a new piece of data is placed in the data bus during the current cycle. The interval associated with the constraint is $\Delta = [0, \infty)$.

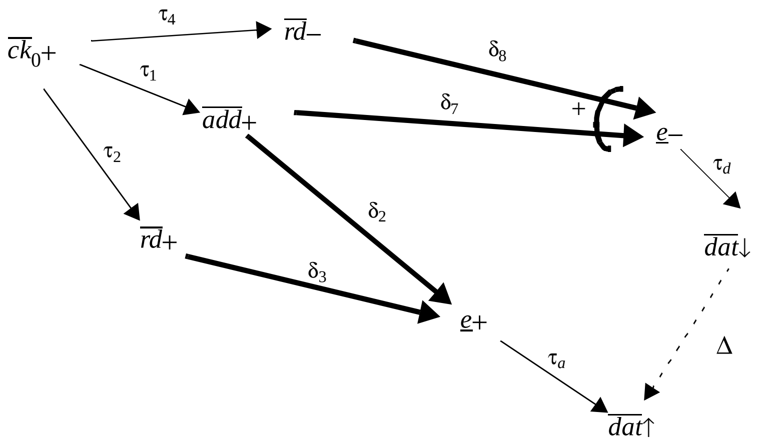


Figure 4.5.13 Back-to-back cycle constraint Δ.

Figure 4.5.13 shows a partial net unfolding for constraint Δ. Notice that $ck_1+$ of the previous cycle coincides with $ck_1+$ of the current cycle.

The bounds of the delays are as follows: $\tau_1 \in [0, 8]$, $\tau_2 \in [8, 13]$, $\tau_4 \in [0, 5]$, $\tau_a \in [5, 7]$, $\tau_d \in [0, 8]$, $\delta_2 \in [2, 5]$, $\delta_3 \in [2, 4]$, $\delta_7 \in [3, 6]$, and $\delta_8 \in [1, 5]$. Delays $\tau_1$, $\tau_2$, and $\tau_4$ are correlated, according to the following inequalities:

$$\tau_2 - \tau_1 \geq 5$$
$$\tau_2 - \tau_4 \geq 8$$

Moreover, the interface delays are also assumed to be correlated according to the following inequalities:

$$-1 \leq \delta_8 - \delta_3 \leq 1$$
$$-1 \leq \delta_7 - \delta_2 \leq 1$$

We assume that the joint pdf is uniform. The joint pdf that characterizes the random variables of the net unfolding $f_{\tau1\tau2\tau4\delta2\delta3\delta7\delta8\tau a\tau d}(\tau_1, \tau_2, \tau_4, \delta_2, \delta_3, \delta_7, \delta_8, \tau_a, \tau_d)$ is given by:

$$f_{\tau1\tau2\tau4}(\tau_1, \tau_2, \tau_4)\ f_{\delta2\delta7}(\delta_2, \delta_7)\ f_{\delta3\delta8}(\delta_3, \delta_8)\ f_{\tau a}(\tau_a)\ f_{\tau d}(\tau_d) \qquad \text{(Eq. 4.5.3)}$$

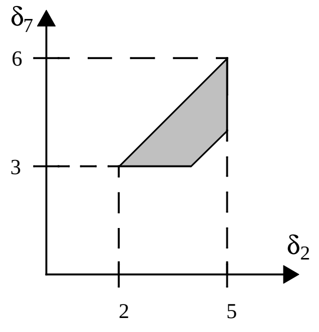


Figure 4.5.14 Projection of $f_{\delta2\delta7}(\delta_2, \delta_7)$.

The projection of $f_{\delta2\delta7}(\delta_2, \delta_7)$ is shown in Figure 4.5.14. The projection of $f_{\delta2\delta7}(\delta_2, \delta_7)$ can be similarly obtained. The projection of $f_{\tau1\tau2\tau4}(\tau_1, \tau_2, \tau_4)$ is shown in Figure 4.5.15. To obtain the actual uniform pdf from its projection, one first must find the hypervolume of the projection. In this dissertation we consider projections described by set of inequalities (*i.e.*, polytopes [104]). The problem of finding efficient algorithms to compute the volume of a polytope has received recently considerable attention in the literature [32, 75, 59, 80, 18]. We have used to compute the volume of the polytopes in our examples Prof. Fukuda's *cdd* code [56], which we gratefully acknowledge.

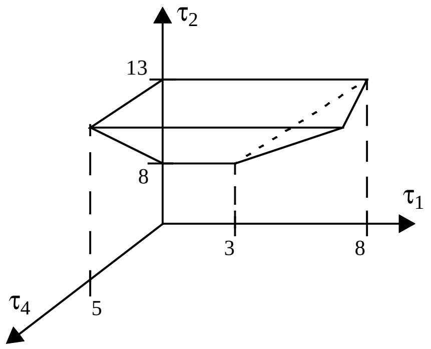


Figure 4.5.15 Projection of $f_{\tau 1 \tau 2 \tau 4}(\tau_1, \tau_2, \tau_4)$.

The constraint equation for $\Delta$ can be written as:

$$\tau_{dat\downarrow} - \tau_{dat\uparrow} \subseteq \Delta \qquad \text{(Eq. 4.5.4)}$$

which can be expressed in terms of the delays with respect to the fork transition $ck_0+$ as follows:

$$\{ \max(\tau_2+\delta_3, \tau_1+\delta_2) + \tau_a \} - \{ \min(\tau_4+\delta_8, \tau_1+\delta_7) + \tau_d \} \subseteq \Delta \qquad \text{(Eq. 4.5.5)}$$

We consider two cases: (a) the delays are assumed to be independent, which implies that there is no correlation; and (b) the delays are correlated. For both cases we have to compute $f_{\tau dat\downarrow \tau dat\uparrow}(\tau_{dat\downarrow}, \tau_{dat\uparrow})$. Figures 4.5.16 and 4.5.17 show $f_{\tau dat\downarrow \tau dat\uparrow}(\tau_{dat\downarrow}, \tau_{dat\uparrow})$ for cases (a) and (b) respectively. Notice the symmetry of the pdf for case (a), in contrast to the skewed pdf corresponding to case (b). This effect is due to the phenomenon of time correlation among the delays.

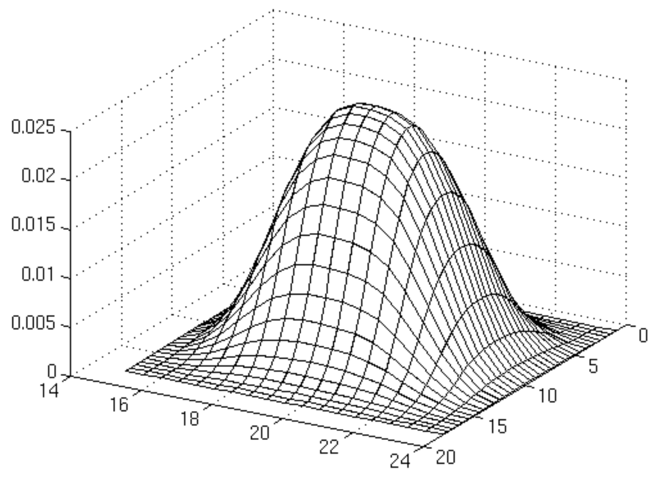


Figure 4.5.16 Joint pdf $f_{\tau dat\downarrow\ \tau dat\text{¦}\downarrow}(\tau_{dat\downarrow}, \tau_{dat\text{¦}\downarrow})$ without correlation.

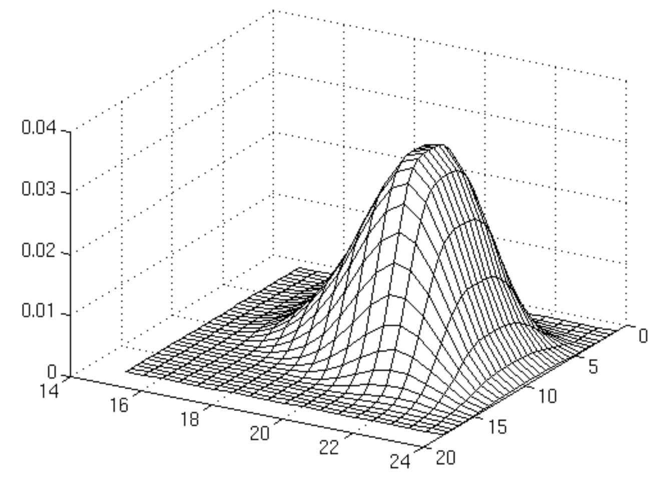


Figure 4.5.17 Joint pdf $f_{\tau dat\downarrow\ \tau dat\text{¦}\downarrow}(\tau_{dat\downarrow}, \tau_{dat\text{¦}\downarrow})$ with correlation.

The pdf of the time separation $z = \tau_{dat\downarrow} - \tau_{dat\uparrow}$ for both cases is shown in Figure 4.5.18. The continuous curve in Figure 4.5.18 corresponds to case (a) and the dashed curve corresponds to case (b). The projection of the pdf for case (a) is the interval [-3, 23], while for case (b) is the interval [4, 21]. The constraint $\Delta = [0, \infty)$ is satisfied by

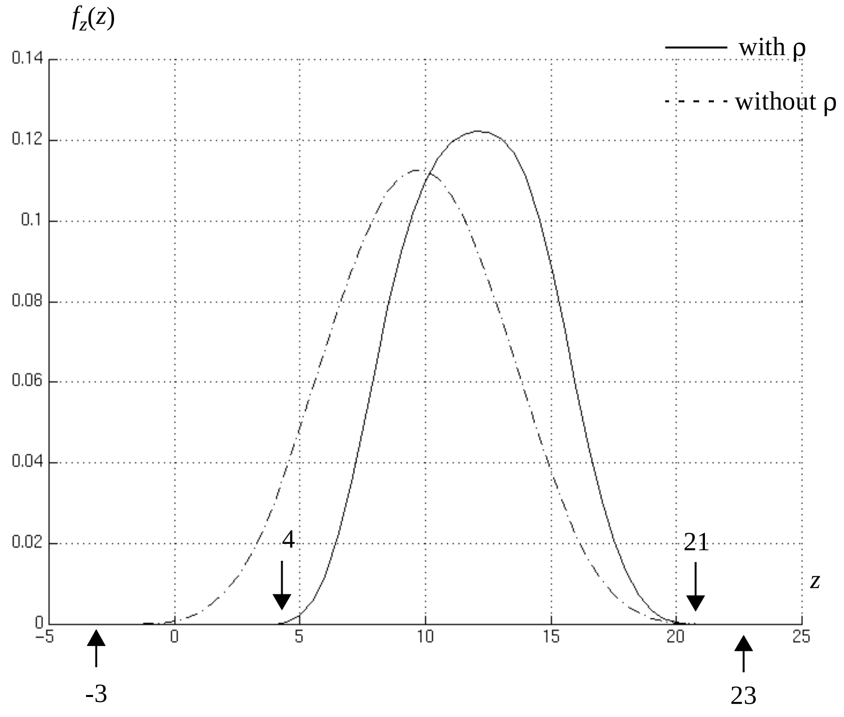


Figure 4.5.18   Probability density function of the time separation.

case (b) but not by case (a). Moreover, our procedure allows the designer to quantify the probability that the timing constraint Δ would be violated in case (a), which turns out to be 2.3%.

Let us interpret the previous two cases as follows: case (a) corresponds to the analysis of a circuit for which correlation among delays is ignored, while in case (b) correlation is taken into consideration. Under this view, an analysis that neglects timing correlation would yield an incorrect conclusion. In general, as discussed in Section 2.5.4, timing correlation reduces the projection of the delays (cf. Figure 4.5.14).

In the following section we present several examples that show the relationship between the probabilistic analysis that we have developed in this chapter and traditional interval analysis techniques.

## 4.5.4 Special cases

In this section we present a suite of special cases in order to show the relationship that exists between an interval arithmetic (or bounded-delay) analysis [112] and our probabilistic approach for the case that the pdf's are bounded (a bounded pdf is zero outside a bounded interval).

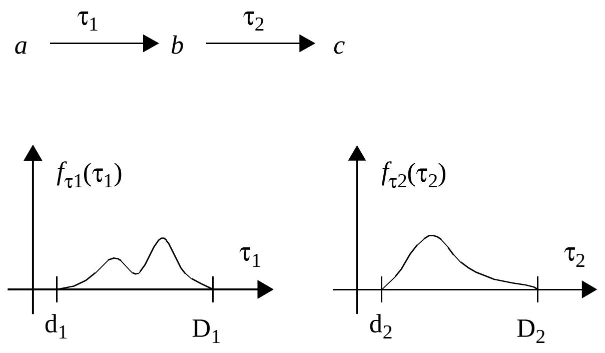


Figure 4.5.19 Sequencing.

We start with the sequencing of two transitions (refer to Figure 4.5.19): transition *a* causes transition *b* which causes transition *c*. Let us find the time separation *z* from *a* to *c*. Clearly the fork transition is *a* itself, thus $z = \tau_1 + \tau_2$. For the sake of simplicity let us assume that $\tau_1$ and $\tau_2$ are independent. Rather than using specific pdf's for $\tau_1$ and $\tau_2$, we shall only assume that the pdf's are bounded, *i.e.*, that the pdf's are non-zero only in the interval $[d_i, D_i]$ (refer to Figure 4.5.19). Notice that a non-empty interval implies that $d_i \leq D_i$.

Using the above assumptions, it can be seen that $f_z(z)$ can be computed using Eq. 4.3.8, which denotes convolution. Figure 4.5.20 shows the integrand of Eq. 4.3.8 for a particular *z*. We are not interested in the actual shape of $f_z(z)$, but in its bounds. It is easy to check that the integrand is non-zero for $z \in [d_1 + d_2, D_1 + D_2]$. Hence $f_z(z)$ is non-zero in

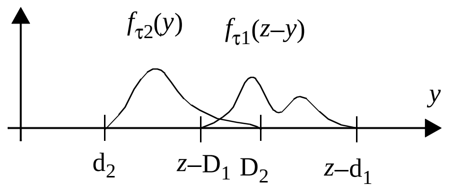


Figure 4.5.20 Convolution.

that interval (refer to Figure 4.5.21). This coincides with the result of the addition of intervals $[d_1, D_1]$ and $[d_2, D_2]$.

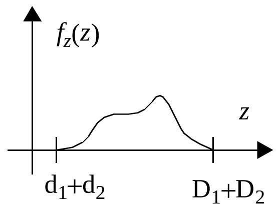


Figure 4.5.21 Sequencing pdf.

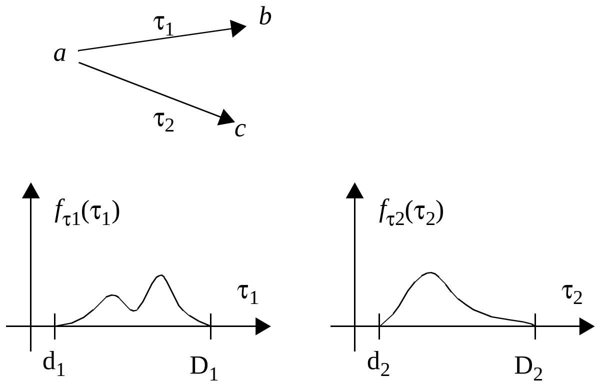


Figure 4.5.22 Time separation.

The second special case is a simple time separation between two transitions (refer to Figure 4.5.22): transition *a* causes both transitions *b* and *c*. Let us find the time separation $z$ from *c* to *b*. The fork transition is *a*, thus $z = \tau_1 - \tau_2$. Let us assume again that $\tau_1$ and $\tau_2$ are independent and that we only know that the pdf's are bounded.

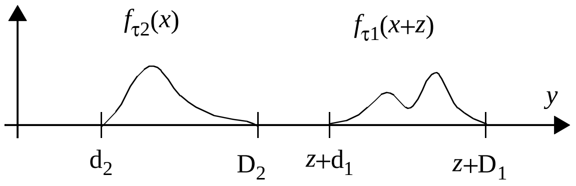


Figure 4.5.23   Time separation construction.

Using the above assumptions, it can be seen that $f_z(z)$ can be computed using Eq. 4.3.11, which denotes the operation of correlation (not to be confused with the correlation data in our interface specifications). Figure 4.5.23 shows the integrand of Eq. 4.3.11 for a particular $z$. The integrand is non-zero for $z \in [d_1 - D_2, D_1 - d_2]$. Hence $f_z(z)$ is non-zero in that interval (refer to Figure 4.5.24). This coincides with the result of the subtraction of intervals $[d_1, D_1]$ and $[d_2, D_2]$.

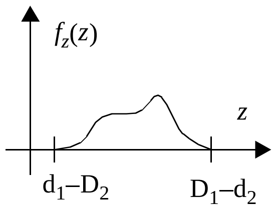


Figure 4.5.24   Time separation pdf.

The third special case is simple AND causality (refer to Figure 4.5.25): transition *c* is caused by both transitions *a* and *b*. Let us assume that both transitions *a* and *b* occurred

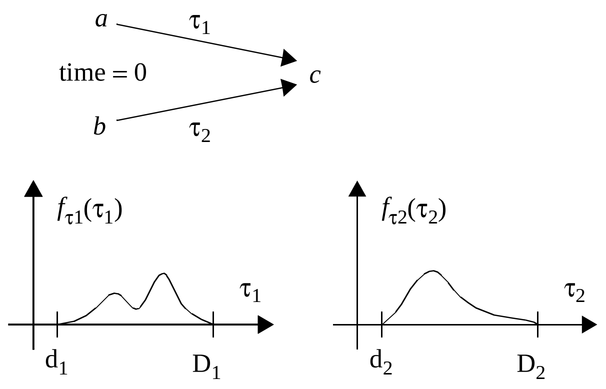


Figure 4.5.25 AND causality.

at exactly the same time $\tau_0$. Let us find the time occurrence $z$ of $c$ with respect to $\tau_0$. Thus $z = \max(\tau_1, \tau_2)$. As before, $\tau_1$ and $\tau_2$ are independent and we only know that the pdf's are bounded.

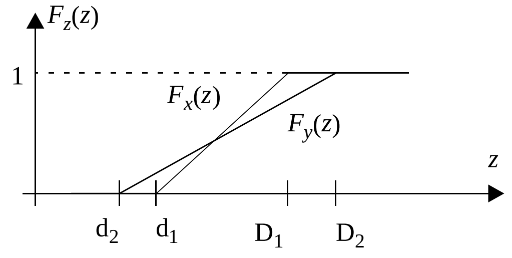


Figure 4.5.26 AND causality construction.

Hence $f_z(z)$ is $f_x(z)F_y(z) + f_y(z)F_x(z)$ (refer to Eq. 4.3.15). Figure 4.5.26 shows $F_x(z)$ and $F_y(z)$. It is easy to check that $f_z(z)$ is non-zero for $z \in [\max(d_1, d_2), \max(D_1, D_2)]$ (refer to Figure 4.5.27). This coincides with the result of the max operation on intervals $[d_1, D_1]$ and $[d_2, D_2]$.

The final special case is simple OR causality (refer to Figure 4.5.28): transition $c$ is OR-caused by both transitions $a$ and $b$. Let us assume that both transitions $a$ and $b$ occurred

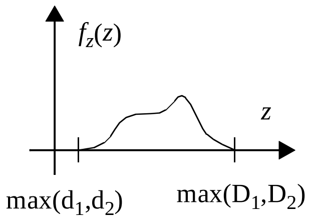


Figure 4.5.27   AND causality pdf.

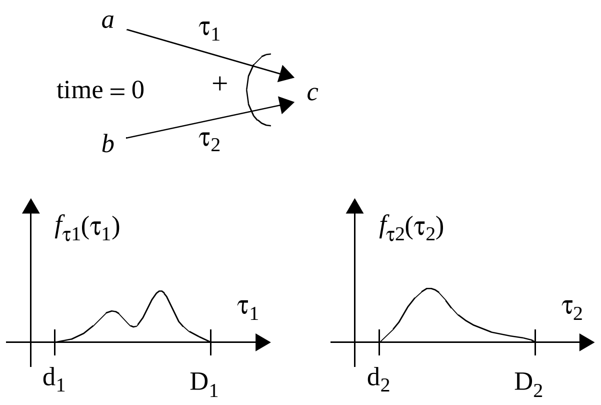


Figure 4.5.28   OR causality.

at $\tau_0$. The time occurrence $z$ of $c$ with respect to $\tau_0$ is $z = \min(\tau_1, \tau_2)$. As before, $\tau_1$ and $\tau_2$ are independent and we only know that the pdf's are bounded.

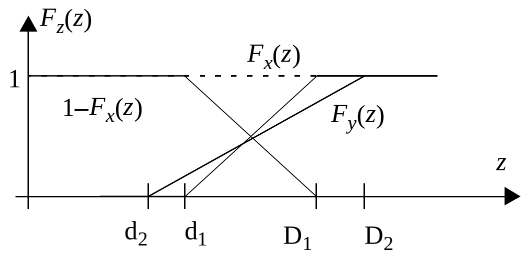


Figure 4.5.29   OR causality construction.

Hence $f_z(z)$ is $f_x(z)(1 - F_y(z)) + f_y(z)(1 - F_x(z))$ (refer to Eq. 4.3.19). Figure 4.5.29 shows $F_x(z)$ and $F_y(z)$. It can be shown that $f_z(z)$ is non-zero for

$z \in [\min(d_1, d_2), \min(D_1, D_2)]$ (refer to Figure 4.5.30). This coincides with the result of the min operation on intervals $[d_1, D_1]$ and $[d_2, D_2]$.

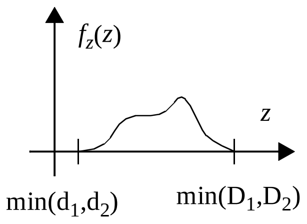


Figure 4.5.30 OR causality pdf.

More complicated graphs cannot be expressed using plain interval arithmetic alone, due to correlation either implied by the joint pdf of the delays of the net, or introduced by the topology of the net (reconvergence fan-out). Of course the interval analysis techniques can be extended to handle more complicated situations.

## 4.6 Summary

In this chapter we have presented a probabilistic interface timing verification procedure that not only can check if every constraint of a cycle-invariant complete graph is satisfied, but if a constraint is not satisfied, it returns a reliability factor which is a measure of the probability that a constraint will be satisfied.

Thus our verification procedure provides more information than traditional interface timing verification techniques, which can be invaluable when evaluating the tradeoffs of a design. Our verification procedure is based on a probabilistic framework that gives a natural interpretation to the time correlation data that crop up in component data sheets. We also show in this chapter that by ignoring time correlation, a pessimistic result may be obtained. In high-speed designs, where every nanosecond counts, it is important to determine tighter bounds: a pessimistic result may seem to indicate a timing constraint violation where none exists.

The verification procedure presented in this chapter requires that all the delays be known. This is not the case prior to the interface logic synthesis. However it is possible to adapt our procedure so that instead of checking for constraint satisfaction, it determines the values of the delays that satisfy all the constraints. In the following chapter we shall present this variation that we have called timing analysis for synthesis.

# Chapter 5

# Timing Analysis for Synthesis

## 5.1 Introduction

During the system integration phase of the design flow, interface logic may be required to construct a system that uses off-the-shelf components. In Chapter 2 we presented a suitable formal specification to describe the interface behavior of the components, while in Chapter 3 we addressed the issue of describing the glue, or interface, logic whose function is to connect the components together, and in Chapter 4 we discussed a timing verification procedure that, once the necessary logic has been implemented, not only checks if the new circuitry meets the timing constraints given in the component specifications, but also gives a measure of the reliability of the system in case some constraints are violated.

In this chapter we shall present an analysis procedure that can be used ahead of the interface logic implementation, or synthesis. The problem that we face is that prior to interface synthesis, the values that the interface delays can take are unknown. One solution is to apply a timing verification procedure for which conservative estimates for the interface delays are used [97, 98]. Our solution is to reformulate the timing interface verification problem as the problem of finding the set of allowed values for the unknown delays so that the constraints are satisfied, which we have called timing analysis for synthesis (TAFS) in [47]. Amon and Borriello [2] suggested a similar idea, that they call symbolic timing verification; however they studied only the convex case for which a solution can be given using standard constraint satisfaction programming techniques [127, 68, 122], and failed to point out that in general “symbolic verification” results in a non-convex problem.

## 5.2 Timing analysis for synthesis problem formulation

Timing analysis for synthesis (TAFS) is a technique that can be used in advance of the interface logic synthesis to determine the values that the interface delays can take if they are to meet the timing constraints given in the interface specifications. The importance of TAFS is that it can break the series of iterations between interface design and implementation that usually occur during the design flow (*i.e.*, a design is implemented, then checked for timing constraint satisfaction, and if violations are encountered, the implementation and/or the design of the interface logic must be redone), because it can be applied on a description of the interface design prior to implementation.

In Chapter 3 we suggested that the interface logic required to integrate off-the-shelf components to build up a system can be described as the “merging” of the interface specifications describing the protocols followed by the components to implement certain capability (*e.g.*, a bus arbitration operation), where the “merging” consisted of adding delay edges, corresponding to the interface logic, to generate the input signal transitions of the specifications. The result of the interface design was a complete graph. To clarify the previous ideas we briefly present the bus arbitration interface design discussed in Chapter 3.

Figure 5.2.1 shows a system comprising a DMA component and the VMEbus. The bus arbitration interface block converts the request-acknowledge protocol used by the DMA component to the more involved request-grant-done bus arbitration protocol defined in the VMEbus standard (refer to Figure 5.2.2).

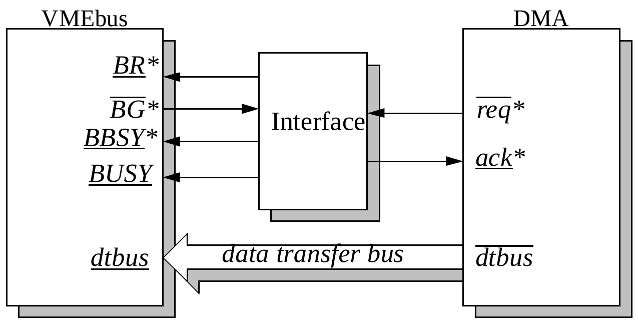


Figure 5.2.1 Structural view of a bus arbitration interface.

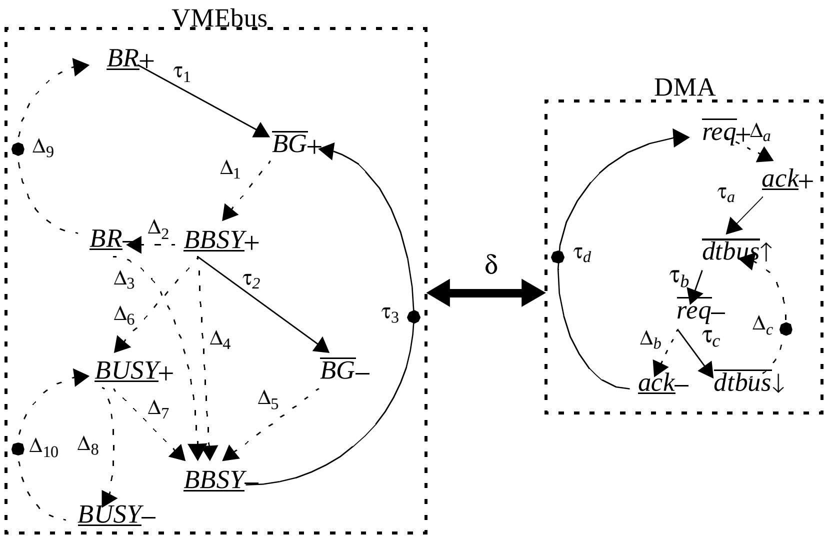


Figure 5.2.2 Interface specifications of the bus arbitration protocols followed by the components of the system shown in Figure 5.2.1.

The complete graph that describes the component protocols and the interface logic is shown in Figure 5.2.3. The interface logic is represented by the interface delay edges shown as thick lines. It is not possible to check if the complete graph is time-consistent without knowing the interface delays δ. In this chapter we formulate timing analysis for synthesis as the problem of finding the tightest bounds on the interface delays δ such that the timing constraints are satisfied. If there is no such set of values then the interface must be redesigned (*e.g.* by modifying some or all of the δ links in the merged graph or by choosing different components). Otherwise those bounds can be used to select an appropriate target technology and guide time-driven synthesis tools. Finally a correct realization of the interface must not exceed the bounds computed by the analysis.

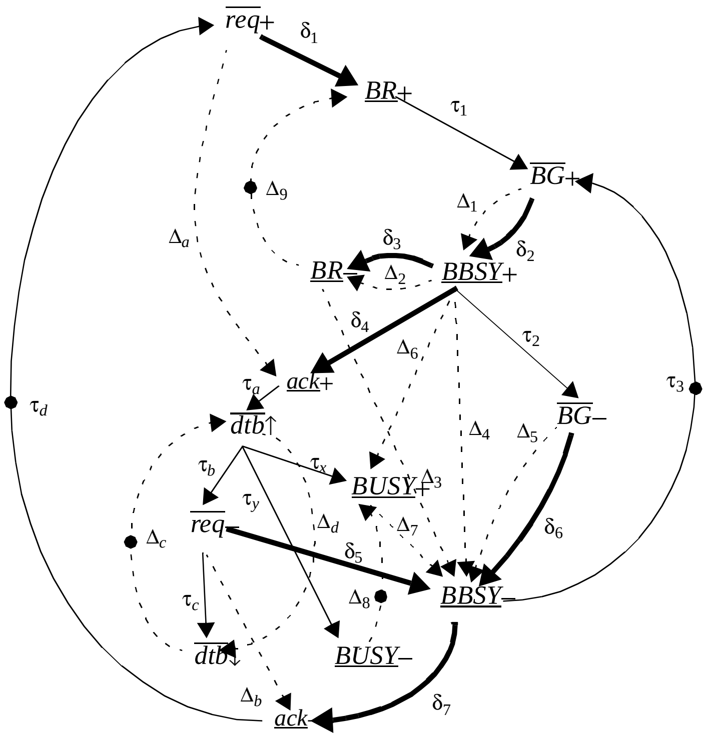


Figure 5.2.3 Bus arbitration interface design.

**Definition 5.2.1.-** Given

1. a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$, where $\Sigma_c = \langle N_c, Y_c, \lambda_c \rangle$ is a timed STG with underlying Petri net $N_c = \langle P_c, T_c, F_c, M_{c0}, \Gamma \rangle$ and $C_{Nc}$ is a set of constraint rules,

2. a partition on the set of places $P_c$ into the set of $\tau$-places $P_\tau$ and the set of $\delta$-places $P_\delta$, with associated sets of random variables $\tau = \{\tau_i \mid \tau_i = \Gamma(p_i), p_i \in P_\tau\}$ and $\delta = \{\delta_i \mid \delta_i = \Gamma(p_i), p_i \in P_\delta\}$,

3. a joint pdf characterizing the set of random variables $x = \tau \cup \delta$ that can be written as:

$$f_x(x) = f_\tau(\tau) \cdot f_\delta(\delta) \qquad \text{(Eq. 5.2.1)}$$

where $f_\tau(\tau)$ and $f_\delta(\delta)$ are joint pdf's of the random variables $\tau$ and $\delta$ respectively.

4. a projection $R_\tau \subseteq R^m$ of the joint pdf $f_\tau(\tau)$, where $m$ is the size of $P_\tau$, and

5. a region $R_a \subseteq R^n$ of allowed values for $\delta$, where $n$ is the size of $P_\delta$;

the timing analysis for synthesis (TAFS) problem is the problem of finding the largest projection $R_\delta \subseteq R_a$ of the joint pdf $f_\delta(\delta)$ such that any possible execution of net $N_c$ satisfies all the constraint rules $c_{ij} \in C_{Nc}$.

Let us compare TAFS and interface timing verification. In timing verification, the input is a complete graph, with a fully specified signal transition graph and the problem is to check that every possible execution of the net satisfies the constraints. In TAFS, the input is also a complete graph, but some of the random variables are unspecified, and thus the problem is to find a characterization of the set of unspecified random variables $\delta$ that satisfies the constraints. Such characterization of $\delta$ is not a joint pdf but its projection. The significance of this fact can be understood by realizing that the set $\delta$ represents the delays of the logic that is yet to be implemented, so that it makes sense to try to determine the projection of the pdf (which from the discussion of Section 2.5.4, specifies only the domain of the delay values by neglecting the probability measure of the joint pdf) rather the pdf itself, otherwise the solution of TAFS would be too restrictive by yielding a specific joint pdf that the interface logic must implement. In this sense, the result of TAFS is ideal to guide the interface implementation because there exist many possible implementations that can satisfy a projection. Notice however that, by dealing with a projection, the probability measure, which was key for our reliability analysis discussed in Chapter 4, is lost. This is not severe because at this phase of the design, the designer is more concerned with getting a broader picture of the properties of a design that has not been fully carried out down to silicon. Notice however that although TAFS is an interval analysis, it solves a more difficult problem than interval interface timing verification. Other comments regarding Definition 5.2.1 are discussed in the following paragraphs.

Condition 3 of the definition assumes that the joint pdf of the net is separable into two pdf's, that is the set of $\tau$ delays and the set of $\delta$ delays are independent; this assumption is not unrealistic, mainly because the interface block is implemented as a separate entity from the components, and although the components and the interface will operate under similar temperature conditions, this will amount to a weak correlation; stronger correlation results from, for example, sharing locality in silicon. (On the other hand, from the discussion of Chapter 4, correlation data improves the time separation calculation so that by neglecting correlation one would expect conservative results.) In the sequel we shall refer to the two components of the set of random variables $x$, the set of $\delta$ delays and the set of $\tau$ delays by the symbols $\delta$ and $\tau$ respectively for the sake of conciseness.

The reader may find the presence of $R_a$ in step 5 of Definition 5.2.1 superfluous, because one seeks to find a solution region $R_\delta$, but $R_a$ seems to be "constraining" $R_\delta$. Our reasoning is that in some circumstances one may know up-front a characterization of the interface delays $\delta$. For instance, if just a known set of technologies is being considered, it might be possible to put an upper bound on the maximum $\delta$ values. If no knowledge is assumed on the $\delta$ values, the allowed region $R_a$ turns out to be the non-negative hyper-octant $\{\delta_i \geq 0\}$ (notice that even in this "unconstrained" case, the values of delays $\delta$ are required to be non-negative). Thus, if partial information is known about the values of $\delta$, this can be specified in our general scheme by defining a suitable $R_a$.

Finally $R_\delta$ may turn out to be empty. This particular result signifies that there is no possible assignment for the delays that satisfy the constraint rules of the complete graph. In other words, it is possible to check if an interface design is feasible, prior to implementation, thus saving considerable design effort. Moreover, even if $R_\delta$ is not empty, by studying $R_\delta$ it may be possible to detect potential problems that would arise during implementation; for example, if the maximum value of a given interface delay is too small, then it may not be possible to implement it given a target technology.

In summary, the goal of TAFS is to best characterize the set of unknown delays $\delta$ by finding the largest projection of the joint pdf that satisfies the timing constraints of the design. The TAFS problem for the case of protocol graphs with arbitrary underlying Petri nets is an extremely difficult one. In this dissertation we solve TAFS for the same sub-class that we studied in Chapters 3 and 4, namely the class of complete graphs whose timed STG is a cycle invariant AOC STG. We also restrict the projections of pdf's $f_\tau(\tau)$ and $f_\delta(\delta)$ and the allowed regions $R_a$ to be convex polyhedra [30].

## 5.3 Solving TAFS

In this section we first present a general procedure that solves the timing analysis for synthesis problem, and then we develop algorithms that implement the procedure.

### 5.3.1 TAFS procedure

By its formulation, one can see that TAFS can be naturally cast as a constraint satisfaction problem, namely one must find the largest set of values that the unknown delays $\delta$ can take such that the timing constraints are satisfied. In this section we present a procedure that solves TAFS for complete graphs whose timed STG is a cycle-invariant AOC STG. Firstly we present some concepts that will be necessary to describe the TAFS procedure.

From Section 2.5.1, we know that a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon \rangle$ defines a time window $\Delta_{ij}$ with respect to the $k$-th occurrence of the constraining transition $t_i$ during which the $(k+\varepsilon)$-th occurrence of the constrained transition $t_j$ is allowed to occur. The constraint rule is satisfied if for all occurrence indices $k > 0$, the following constraint equation for $c_{ij}$ is true:

$$\tau_{tj(k+\varepsilon)} - \tau_{ti(k)} \in \Delta_{ij} \qquad \text{(Eq. 5.3.1)}$$

If a net is repeatable, then the time separation $\tau_{tj(k+\varepsilon)} - \tau_{ti(k)}$ is invariant for $k \geq M$. Thus it is possible to check Eq. 5.3.1 for $k \geq M$ by checking Eq. 5.3.1 for $k = M$. In Chapter 3, we identified a structural condition on the net, the presence of a *cycle-invariant fork transition* (refer to Definition 3.3.9), that implies repeatability. Furthermore, as discussed in Section 3.3.4, one can express the left-hand side of Eq. 5.3.1 with respect to a cycle-invariant fork transition $x$ as:

$$\tau_{tj(k+\varepsilon)\Diamond x} - \tau_{ti(k)\Diamond x} \in \Delta_{ij} \qquad \text{(Eq. 5.3.2)}$$

where each of $\tau_{tj(k+\varepsilon)\Diamond x}$ and $\tau_{tj(k+\varepsilon)\Diamond x}$ are expressions containing linear/min/max terms on a set of random variables that defines the timing behavior of the net.

With TAFS the probability measure of the delays has been dropped. Moreover, the characterization of the random variables of the net is incomplete because one only knows the projection of the joint pdf of the $\tau$ delays, which specifies the set of possible values that $\tau$ can take. Then instead of checking if the constraint equation (refer to Eq. 5.3.2) is satisfied, one must find the projection of the joint pdf of the $\delta$ delays that satisfy the constraint equation, for *all possible values* of the $\tau$ delays. To understand this crucial point, consider that there are some values of $\delta$, say $\delta_0$, that satisfy the constraint equation for some values of $\tau$, say $\tau_0$. However if $\delta_0$ together with another set of values of $\tau$ does not satisfy the constraint equation, then $\delta_0$ is only a particular solution which does not hold in general. Thus the set of values of $\delta$ that satisfies the constraints must do so for **all** possible values of $\tau$.

To solve TAFS we shall proceed in two steps: first we shall determine all the set of values of $\delta$ and $\tau$ that satisfy the constraint equation; and then we shall restrict such set to the values that satisfy the constraint equation for all values of $\tau$.

**Definition 5.3.1.-** Given a TAFS problem, the allowed region for $(\tau, \delta)$ is $R_{\tau+a} = \{(\tau, \delta) \in R^{m+n} \mid \tau \in R_\tau \text{ and } \delta \in R_a\}$.

The allowed region $R_{\tau+a}$ represents the set of allowed values $(\tau, \delta)$ defined by the projection of $f_\tau(\tau)$ and the allowed values of $\delta$. In this sense, $R_{\tau+a}$ is the preliminary projection of the joint pdf describing the complete graph, where $R_a$ is used as a first approximation of $f_\delta(\delta)$.

**Definition 5.3.2.-** Given a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon\rangle$ of a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc}\rangle$, the $k$-th constraint separation of $c_{ij}$ is $g_{cij}(k) = \tau_{tj(k+\varepsilon)} - \tau_{ti(k)}$.

Notice that when $g_{cij}(k)$ is calculated with respect to a fork transition, it is a function of the $\tau$ and $\delta$ random variables.

**Definition 5.3.3.-** Given a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon\rangle$ of a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc}\rangle$, the *region* of the $k$-th time separation from $t_i$ to $t_j$ is $R_{\Delta ij}(k) = \{(\tau, \delta) \in R^{m+n} \mid g_{cij}(k) \in \Delta_{ij}\}$.

Hence $R_{\Delta ij}(k)$ represents the set of values $(\tau, \delta)$ that satisfy the constraint equation $g_{cij}(k) \in \Delta_{ij}$.

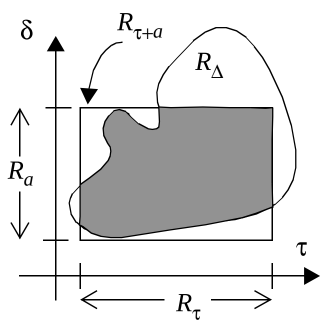


Figure 5.3.1 Set of $(\tau, \delta)$ values that satisfy a constraint $\Delta$.

Consider for the moment a single constraint equation $g_{cij}(k) \in \Delta$. Given the set of allowed values $R_a$ of $\delta$ and the set of values $R_\tau$ that $\tau$ can take, one can construct the allowed region $R_{\tau+a}$ of $(\tau, \delta)$. Thus the allowed values of $(\tau, \delta)$ that satisfy $g_{cij}(k) \in \Delta$ is given by $R_{\tau+a} \cap R_{\Delta ij}(k)$. We call this region the feasible region of the constraint. These ideas are illustrated in Figure 5.3.1. The regions $R_a$ and $R_\tau$ define the box $R_{\tau+a}$. The intersection of $R_{\Delta ij}(k)$ with $R_{\tau+a}$ (the shaded area) are the allowed values $(\tau, \delta)$ that satisfy the constraint equation.

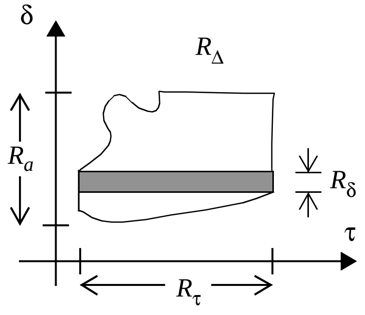


Figure 5.3.2 Set of $\delta$ values that satisfy a constraint $\Delta$ for all values of $\tau$.

However TAFS asks for the values of $\delta$ that satisfy the constraint equation for all $\tau$, which is $R_\delta$. To compute this area one can "cut" the region $R_{\tau+a} \cap R_{\Delta ij}(k)$ into *vertical* infinitesimal "slices" in Figure 5.3.1, and the *horizontal* intersection over all the slices would be $R_\delta$. We have called this procedure the reduction of the feasible region $R_f$. Figure 5.3.2 illustrates region $R_\delta$, of the values of $\delta$ that satisfy the constraint equation for all $\tau$. A formal treatment is presented in Section 5.3.4.

Without the existence of a cycle-invariant fork transition (refer to Section 3.3) one would have to check an infinite number of constraint equations $g_{cij}(k) \in \Delta_{ij}$. The following definitions specialize Definition 5.3.3 for the important case of cycle-invariant fork transitions.

**Definition 5.3.4.-** Given a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon \rangle$ of a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$ and a cycle-invariant fork transition of $t_i$ and $t_j$ for $k \geq M$, the *invariant* constraint separation of $c_{ij}$ is $g_{cij} = \tau_{tj(n+\varepsilon)} - \tau_{ti(n)}$ for any $n \geq M$.

**Definition 5.3.5.-** Given a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon \rangle$ of a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$ and a cycle-invariant fork transition of $t_i$ and $t_j$, the *region* of the invariant constraint separation of $c_{ij}$ is $R_{\Delta ij} = \{(\tau, \delta) \in R^{m+n} \mid g_{cij} \in \Delta_{ij}\}$.

The definitions above refer to a particular region of a constraint rule, either for a particular cycle $k$, or in the presence of a cycle-invariant fork transition for an infinite number of cycles as discussed in Chapter 3. In general the region of a constraint rule is the intersection of the regions for all cycles.

**Definition 5.3.6.-** Given a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon \rangle$ of a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$, the *total region* of constraint $c_{ij}$ is $R_T(c_{ij}) = \bigcap_{k > 0} R_{\Delta ij}(k)$ .

The following lemma establishes that for a constraint rule associated with a cycle-invariant fork transition it is possible to compute the total region by a finite number of intersections.

**Lemma 5.3.1.-** For a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon \rangle$ with associated cycle-invariant fork transition from $t_i$ to $t_j$, the total region of constraint $c_{ij}$ is $R_T(c_{ij}) = \bigcap_{k} R_{\Delta ij}(k)$ , $k = 1, \ldots, M$.

**Proof.-** Direct from Definition 3.3.9 of cycle-invariant fork transition. □

According to our previous discussion (*cf.* Figure 5.3.1), the following definitions capture the concept of the feasible region of a constraint rule.

**Definition 5.3.7.-** Given a TAFS problem and a constraint rule $c_{ij} = \langle t_i, t_j, \Delta_{ij}, \varepsilon \rangle$ of a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$, the feasible region of constraint $c_{ij}$ is $R_f(c_{ij}) = R_T(c_{ij}) \cap R_{\tau+a}$.

**Definition 5.3.8.-** Given a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$, the feasible region of $\Psi_c$ is $R_f = \bigcap_{c_{ij} \in C_{Nc}} R_f(c_{ij})$ .

Finally the important concept of the δ-reduction of a feasible region is defined.

**Definition 5.3.9.-** Given a TAFS problem and a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$, the δ-reduction of the feasible region is $R_r = \{\delta \in R^n \mid \forall \tau \in R_\tau, (\tau, \delta) \in R_f\}$.

**Theorem 5.3.2.-** The δ-reduction of the feasible region of a complete graph is the solution of TAFS.

**Proof.-** From Definition 5.3.8, the feasible region $R_f$ of a complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$ clearly satisfies all the constraint equations $g_{cij}(k) \in \Delta_{ij}$ for all constraint rules $c_{ij} \in C_{Nc}$. Then $R_r$ also satisfies the constraints. We have to show that $R_r$ is the largest sub-set of $\{\delta \in R_a\}$ that satisfies the constraints for all possible values of τ. This is guaranteed by Definition 5.3.9. Therefore $R_r$ is $R_\delta$. □

Theorem 5.3.2 links the δ-reduction operation to our original problem of finding the solution of the TAFS problem. The following procedure summarizes the major steps that one must follow to solve a TAFS problem.

**Procedure 5.3.1.-** Given a TAFS problem for a cycle-invariant complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$,

1. For each constraint rule $c_{ij} \in C_{Nc}$, obtain the total region $R_T(c_{ij})$.

2. For the complete graph $\Psi_c = \langle \Sigma_c, C_{Nc} \rangle$, obtain the feasible region $R_f$.

3. Obtain the $\delta$-reduction of $R_f$, and return it as $R_\delta$.

It is easy to prove using Theorem 5.3.2 that Procedure 5.3.1 solves TAFS. Using the fact that the complete graph is cycle-invariant step 1 computes the total region of a constraint rule by considering a finite number of regions of constraint equations $g_{cij}(k) \in \Delta_{ij}$. In the following sections we shall discuss how to implement the three steps of Procedure 5.3.1 for the case in which the projection $R_\tau$ of $f_\tau(\tau)$ is a polyhedron (refer to Section 2.5.4). It will be shown that even if $R_\tau$ is convex, in general $R_\delta$ is not.

### 5.3.2 Linearization of the constraint equations

In this section we shall discuss how to obtain the feasible region of a complete graph. The first step consists of determining the region of a constraint equation. The complete graphs that we consider have an underlying AOC STG, thus from the discussion of Section 3.3.3, the constraint equations contain only linear/min/max terms. We shall exploit the form of the constraint equations to develop a procedure that, using standard techniques, can find the feasible region of the complete graph.

As mentioned in the previous section, a constraint equation $g_{cij}(k) \in \Delta_{ij}$ defines a region of the values $(\tau, \delta)$ that satisfy the equation for a particular execution of the net. The problem we want to address in this section is how to compute such a region. The crucial point is that a constraint equation is an expression on the random variables $(\tau, \delta)$ involving only linear, min and max operators. We use a linearization technique that converts the non-linear expression of a constraint equation into a set of linear expressions [82, 20] which, as we shall show below, have some nice properties. To make evident the dependency of the constraint separation $g_{cij}$ on $(\tau, \delta)$ we shall denote $g_{cij}(k)$ by $g_{cij}(k, \tau, \delta)$ in the sequel.

**Definition 5.3.10.-** The $i$ linear sub-case of a max term $\max(a_1, \ldots , a_n)$, for $i = 1, \ldots ,n$, replaces the max term with term $a_i$ and adds the set of inequalities $\{a_i \geq a_j \mid j = 1, \ldots ,n \text{ and } j \neq i\}$.

**Definition 5.3.11.-** The $i$ linear sub-case of a min term $\min(a_1, \ldots , a_n)$, for $i = 1, \ldots ,n$, replaces the min term with term $a_i$ and adds the set of inequalities $\{a_i \leq a_j \mid j = 1, \ldots ,n \text{ and } j \neq i\}$.

**Definition 5.3.12.-** The linearization of a max, or a min, term is the set of $n$ linear sub-cases of the max, or min, term.

**Definition 5.3.13.-** A linear sub-case of a constraint equation $g_{cij}(k, \tau, \delta) \in \Delta_{ij}$ is formed by selecting a linear sub-case for each of the max and min terms that appear in $g_{cij}(k, \tau, \delta)$.

Thus to find a linear sub-case of a constraint equation one must choose a “winner” for each max and min term in $g_{cij}(k, \tau, \delta)$. The linearization of a constraint equation comprises all the linear sub-cases.

**Definition 5.3.14.-** The linearization of a constraint equation $g_{cij}(k, \tau, \delta) \in \Delta_{ij}$ is the set of all linear sub-cases of the constraint equation.

For example, the linearization of the left-hand side of the following constraint equation $g_{cij}(k, \tau, \delta) \in \Delta_{ij}$ given by

$$g_{cij}(k, \tau, \delta) = \tau_1 + \max(\min(\tau_2, \delta_3) + \tau_4, \delta_5) - \delta_3$$

consists of the following linear sub-cases:

1. (linear sub-case 1 of the max term, linear sub-case 1 of the min term):

$$\begin{aligned}&\tau_1 + \tau_2 + \tau_4 - \delta_3 \in \Delta_{ij}\\&\tau_2 \leq \delta_3\\&\tau_2 + \tau_4 \geq \delta_5\end{aligned}$$

2. (linear sub-case 1 of the max term, linear sub-case 2 of the min term):

$$\tau_1 + \tau_4 \in \Delta_{ij}$$
$$\tau_2 \geq \delta_3$$
$$\delta_3 + \tau_4 \geq \delta_5$$

3. (linear sub-case 2 of the max term, linear sub-case 1 of the min term):

$$\tau_1 + \delta_5 - \delta_3 \in \Delta_{ij}$$
$$\tau_2 \leq \delta_3$$
$$\tau_2 + \tau_4 \leq \delta_5$$

4. (linear sub-case 2 of the max term, linear sub-case 2 of the min term):

$$\tau_1 + \delta_5 - \delta_3 \in \Delta_{ij}$$
$$\tau_2 \geq \delta_3$$
$$\delta_3 + \tau_4 \leq \delta_5$$

**Lemma 5.3.3.-** The maximum number of linear sub-cases of a constraint equation does not exceed the product of the linear sub-cases of its different max and min terms.

**Proof.-** In the worst case, the selection of each different linear sub-case of a max/min term is independent of the selection of linear sub-cases of the other max/min terms thus generating a maximum number of linear sub-cases which is the product of the terms of all the max/min terms. □

In the previous example, the constraint equation generated four linear sub-cases corresponding to two independent selections of each of its two min/max terms (*i.e.*, the number of cases achieves the maximum of Lemma 5.3.3). However if there are some common terms to the min/max terms of a constraint equation, a fewer number of linear sub-cases may be generated. For example, if the left-hand side of a constraint equation is given by

$$max(a, b) + min\ (a, b)$$

one can notice that only two linear sub-cases belong to the linearization, namely for the case that $a \geq b$ and for the case that $a \leq b$ due to the sharing of common terms in the min and max terms.

**Lemma 5.3.4.-** Each linear sub-case of a constraint equation describes a polyhedron in $R^{m+n}$, where $m$ and $n$ are the number of $\tau$ and $\delta$ variables respectively.

**Proof.-** A linear sub-case of a constraint equation consists of the original constraint equation in which each max/min terms has been replaced by a particular winner term, and a set of inequalities that have been added for each winner selection. Each added inequality is linear according to Definitions 5.3.10 and 5.3.11. Also the new expression for the constraint equation is linear by construction, due to the replacement of the max/min terms (the only non-linear terms in the constraint equation) with terms $\tau_i$. Notice that the expression $g \in \Delta$, where $\Delta$ is a closed interval $[d, D]$, can be replaced by $d \leq g \leq D$. (If $\Delta$ is not a closed interval, the corresponding lower/upper bound of the interval that is not closed is described using the strict inequality symbol '<'.) Finally a set of linear inequalities on $k$ variables describes a convex polyhedron in $R^k$ [30]. □

Before stating the property that the linear sub-cases of a constraint equation describe non-overlapping regions, we need to give some basic definitions (adapted from [60]). For a definition of a hyperplane, refer to Definition 2.5.5.

**Definition 5.3.15.-** A hyperplane $H$ *cuts* a region $A \subseteq R^k$ if both open spaces into which $R^k$ is divided by $H$ contain points of $A$.

**Definition 5.3.16.-** The Euclidean distance between two points $x_1$, $x_2 \in R^k$, is $d(x_1, x_2) = (\langle x_1 - x_2, x_1 - x_2 \rangle)^{1/2}$, where $\langle x_1, x_2 \rangle$ denotes the dot product of $x_1$ and $x_2$. The Euclidean distance between regions $A, B \subset R^k$ is $d(A, B) = \inf(\{d(x_1, x_2) \mid x_1 \in A$ and $x_2 \in B\})$, where inf selects the minimum element of a set of values.

**Definition 5.3.17.-** A hyperplane $H$ *supports* a region $A \subseteq R^k$ if $H$ does not cut $A$ and $d(A, H) = 0$.

**Definition 5.3.18.-** Let $P_k$ be a polyhedron in $R^k$. A set $F \subset P_k$ is a face of $P_k$ if there exists a hyperplane $H$ that supports $P_k$ and $F = P_k \cap H$.

**Definition 5.3.19.-** Two polyhedra in $R^k$ are *non-overlapping* if their intersection is either empty or is a common face of the two polyhedra.

**Lemma 5.3.5.-** The polyhedron described by a linear sub-case of a constraint equation is non-overlapping with any of the polyhedra described by the other linear sub-cases of the constraint equation.

**Proof.-** A linear sub-case $i$ differs from the other linear sub-cases of a constraint equation by at least one selection of a winner for a given max/min term. Without loss of generality, assume that for the given *max* term, term $a_i$ was chosen as the winner for linear sub-case $i$, while $a_j$ was chosen as the winner, with $j \neq i$, for the other linear sub-case. Then an added inequality $a_i \geq a_j$ is one of the inequalities that describe the polyhedron of sub-case $i$, while $a_j \geq a_i$ is one of the inequalities that describe the polyhedron of the other sub-case. Each of these inequalities represents a closed half-space [30]. Furthermore, the two half-spaces define two regions that do not overlap, and only have the hyper-plane $a_i = a_j$ in common. A polyhedron is the *intersection* of the half-spaces described by a set of inequalities. Therefore the polyhedron corresponding to sub-case $i$ does not overlap with any of the polyhedra of the other sub-cases. $\square$

**Definition 5.3.20.-** The *region* of the linearization of a constraint equation $g_{cij}(k, \tau, \delta) \in \Delta_{ij}$ is $\bigcup_k P_k$, where $P_k$ is the polyhedron described by the $k$-th linear sub-case of the constraint equation.

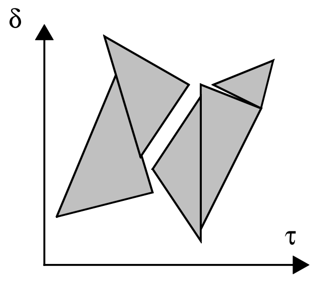


Figure 5.3.3 Region of the linearization of a constraint equation.

Thus the geometric region corresponding to the linearization of a constraint equation can be described by a set of non-overlapping convex polyhedra. However the region needs not be convex nor connected (refer to Figure 5.3.3).

**Lemma 5.3.6.-** Given two regions $A$, $B \subseteq R^k$, each one described as the union of non-overlapping polyhedra, denoted $\bigcup_i P_{Ai}$ and $\bigcup_j P_{Bj}$ respectively, their intersection $A \cap B$ is $\bigcup_{i,j} P^{AB}{}_{ij}$ where $P^{AB}{}_{ij} = P_{Ai} \cap P_{Bj}$, and $\forall\,(i, j) \neq (k, l)$, $P^{AB}{}_{ij} \cap P^{AB}{}_{kl} = \phi$.

**Proof.-** Consider one polyhedron from each region, say $P_{Ai_1}$ and $P_{Bj_1}$. Their intersection $P^{AB}{}_{i_1 j_1} = P_{Ai_1} \cap P_{Bj_1}$, is clearly in $A \cap B$. Moreover $P^{AB}{}_{i_1 j_1}$ is non-overlapping with any other $P_{Ak}$, and any other $P_{Bl}$ because of the non-overlapping property of the sets $\bigcup_i P_{Ai}$ and $\bigcup_j P_{Bj}$. Then $P^{AB}{}_{i_1 j_1}$ is non-overlapping with any other $P^{AB}{}_{kl} = P_{Ak} \cap P_{Bl}$. Finally suppose that there is an element $x \in A \cap B$ but not in

$\bigcup_{i,j} P^{AB}{}_{ij}$. But then $x$ must belong to one component $P_{Ai}$ of $A$ and one component $P_{Bj}$ of $B$, and therefore $x$ must be in one partial intersection $P_{Ai} \cap P_{Bj}$. □

Lemma 5.3.6 states that the intersection of two (possibly non-convex and disconnected) regions described as the union of non-overlapping polyhedra can also be described as the union of non-overlapping polyhedra.

**Lemma 5.3.7.-** The region of the linearization of a constraint equation can be represented as the union of non-overlapping polyhedra.

**Proof.-** It follows from Lemma 5.3.5. □

**Lemma 5.3.8.-** The region of the linearization of a constraint equation $g_{cij}(k, \tau, \delta) \in \Delta_{ij}$ is identical to the region $R_{\Delta ij}(k)$ of the $k$-th time separation (refer to Definition 5.3.3).

**Proof.-** We have to show that every point in the region of the linearization belongs to the region of the $k$-th time separation, and that there are no points belonging to the region of the $k$-th time separation that do not belong to the region of the linearization. Assume that there is a point $(\tau_0, \delta_0)$ which is in the region of the linearization. Then $(\tau_0, \delta_0)$ is in the polyhedron of a linear sub-case, which corresponds to a specialization of the constraint equation $g_{cij}(k, \tau, \delta) \in \Delta_{ij}$ in which exactly one term of each max/min term has been chosen as a winner according to Definitions 5.3.10 and 5.3.11. Therefore $g_{cij}(k, \tau_0, \delta_0) \in \Delta_{ij}$ must hold. Now suppose that there is a point $(\tau_0, \delta_0)$ that is in the region of the $k$-th time separation. Then $g_{cij}(k, \tau_0, \delta_0) \in \Delta_{ij}$ holds. Evaluate every max/min term and identify a winner. Then $(\tau_0, \delta_0)$ must belong to the polyhedron of the sub-case that corresponds to the identified winner selection. □

The total region $R_T(c_{ij})$ of a constraint rule $c_{ij}$ (refer to Definition 5.3.6) is the intersection of the regions of the constraint equations $g_{c_{ij}}(k, \tau_0, \delta_0) \in \Delta_{ij}$ for $k > 0$. The feasible region of a complete graph (refer to Definition 5.3.8) is the intersection of all the total regions of the constraint rules of the complete graph with the allowed region for $(\tau, \delta)$ $R_{\tau + a}$. From Lemma 5.3.6, the feasible region can be described as the union of non-overlapping polyhedra.

**Theorem 5.3.9.-** The feasible region of the linearization of a constraint equation can be represented as the union of non-overlapping polyhedra.

**Proof.-** It follows from Definitions 5.3.6, 5.3.7, and 5.3.8, and Lemmas 5.3.6, 5.3.7, and 5.3.8. □

Notice that from the previous discussion, the feasible region is not convex in general.

The final step to find the solution of TAFS is to perform a δ-reduction of the feasible region (refer to Definition 5.3.9). That is the topic of Section 5.3.4. Before tackling the reduction problem, we present a simple example in the following section to clarify the ideas presented so far, and to motivate the basis of our δ-reduction procedure.

## 5.3.3 An illustrative example

In this section we use a cycle-invariant constraint to walk through the steps necessary to solve a TAFS problem.

Consider for example the partial unfolding from the cycle-invariant fork transition of transitions $c$ and $d$ shown in Figure 5.3.4. The constraint rule has associated interval $\Delta = [0, 2]$. There are three places (edges) labeled with random variables $\tau_1$, $\delta_1$ and $\delta_2$. The pdf of random variable $\tau_1$ is known and its projection $R_{\tau_1}$ is the interval [0, 1]. The

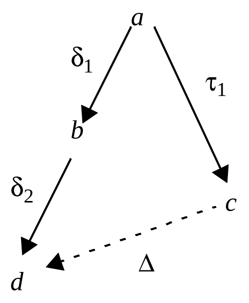


Figure 5.3.4 Computing the projection of $f_{\delta 1\delta 2}(\delta_1, \delta_2)$.

allowed region $R_a$ of the unknown random variables $\delta_1$ and $\delta_2$ is the non-negative quadrant, *i.e.* $R_a = \{(\delta_1, \delta_2) \in R^2 \mid \delta_1, \delta_2 \geq 0\}$.

Assuming that the invariance of the constraint equation applies for all $k > 0$, the region of the invariant constraint separation is also the total region of the constraint rule. The feasible region is given by the intersection of the total regions of the constraints (in our example, only one) with the allowed region $R_{\tau+a} = \{(\tau_1, \delta_1, \delta_2) \mid \tau_1 \in [0,1], \delta_1, \delta_2 \geq 0\}$. The invariant constraint equation is:

$$\delta_1 + \delta_2 - \tau_1 \in [0, 2] \qquad \text{(Eq. 5.3.3)}$$

which does not contain non-linear terms. This fact simplifies the visualization of the problem by not having to draw a cluttered region. Recall that non-linear max/min terms can be linearized so that the only difference is that we have to consider several linear sub-cases. The constraint equation defines a region between the hyperplanes $\delta_1 + \delta_2 - \tau_1 = 0$ and $\delta_1 + \delta_2 - \tau_1 = 2$. The feasible region is shown in Figure 5.3.5, which is a 3-D polyhedron with its apex at the origin (the dashed lines are hidden lines).

We now introduce some important ideas informally that are the basis of our $\delta$-reduction procedure. As discussed in Section 5.3.1, the feasible region contains some values of $\delta$ that, although they satisfy the constraints, they do only for certain values of $\tau$.

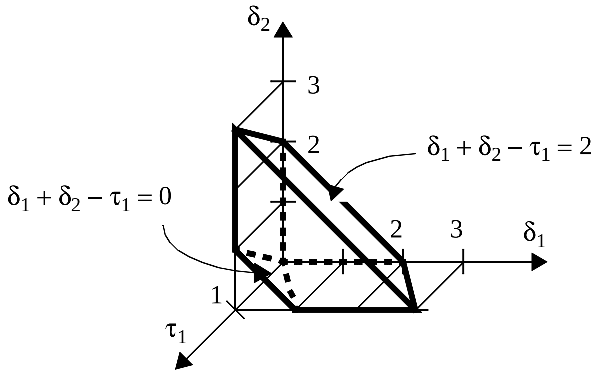

$\delta_2$
3
2
$\delta_1 + \delta_2 - \tau_1 = 2$
$\delta_1 + \delta_2 - \tau_1 = 0$
2
3
$\delta_1$
1
$\tau_1$

Figure 5.3.5 The feasible region $R_T$.

For example, $\delta_1 = 0$ and $\delta_2 = 0$ satisfy $\Delta$ only for $\tau_1 = 0$. The solution region of $\delta$ must satisfy the constraints for all possible values of $\tau$.

In Section 5.3.1 we suggested a procedure that “cuts” the feasible region in infinitesimal “slices” of $R_a$, and orthogonal to $R_\delta$. In our example, the slices would be planes parallel to plane $\delta_1$-$\delta_2$. Then the region representing the $\delta$-reduction is computed as the intersection of all such slices. Such a procedure is not practical because it involves an infinite number of slices. However if the feasible region is convex, then one needs only consider a finite number of slices of $R_a$, namely the slices that intersect one or more of the extreme points, or vertices, of the feasible region, as we shall prove in the following section. For the moment notice that to find out the $\delta$-reduction of the feasible region shown in Figure 5.3.5, one has to consider only the two slices of $R_a$ that intersect hyperplanes $\tau_1 = 0$ and $\tau_1 = 1$. The intersection of these two slices is the intersection of all the slices between $\tau_1 = 0$ and $\tau_1 = 1$. The $\delta$-reduction of the feasible region in Figure 5.3.5 is shown in Figure 5.3.6.

The ideas discussed above can be applied only to convex regions. From the discussion of the previous section, one knows that the feasible region is not necessarily convex,

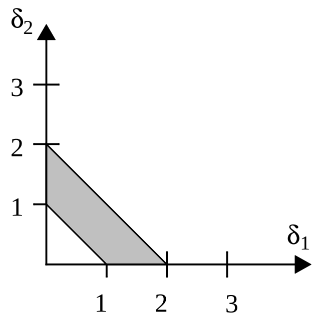


Figure 5.3.6 Projection $R_\delta$ of $f_{\delta1\delta2}(\delta_1, \delta_2)$.

but it can be represented as the union of non-overlapping convex polyhedra. And because the intersection of such non-convex regions can also be expressed as the union of non-overlapping polyhedra, we shall apply those ideas to the convex components of the feasible region (*i.e.*, the polyhedra that form the feasible region).

## 5.3.4 Reduction of the feasible region

In this section we present a procedure that obtains the δ-reduction of a feasible region. From Section 5.3.1, the δ-reduction of the feasible region of an AOC cycle-invariant complete graph is the solution of the corresponding TAFS problem. We first present some basic definitions. We start with a standard definition of an extreme point of polyhedra (*cf.* [30]).

**Definition 5.3.21.-** Given a polyhedron $P \subseteq R^k$ and two points $y$, $z \in P$, a point $x \in P$ is an extreme point of $P$ if $x = \lambda y + (1 - \lambda)z$ for all $\lambda$ such that $0 < \lambda < 1$ implies that $x = y = z$.

The set of extreme points of a polyhedron are called vertices.

To describe our reduction procedure, we need to introduce the following definitions:

**Definition 5.3.22.-** Given a polyhedron $P \subseteq R^{m+n}$, and a point $\tau_0 \in R^m$, the *slice* of $P$ at $\tau_0$ is denoted by $P(\tau_0) = \{\delta \in R^n \mid (\tau_0, \delta) \in P\}$.

**Definition 5.3.23.-** Given a point $x = (\tau, \delta)$ of a polyhedron $P \subseteq R^{m+n}$, where $\tau \in R^m$ and $\delta \in R^n$ are $x$'s components, the $\tau$-*projection* of $x$ into $R^m$ is $x$'s $\tau$ component.

**Definition 5.3.24.-** Given a polyhedron $P \subseteq R^{m+n}$, its $\tau$-region is the region $P_\tau = \{\tau \in R^m \mid \delta \in R^n \text{ and } (\tau, \delta) \in P\}$.

Notice that the region described by $P_\tau$ is the projection of region $P$ in ($m$+$n$)-dimensional space onto the $m$-dimensional space of $\tau$. (In our application, $\tau$ represents the set of known delays, and $P_\tau$ corresponds to the given projection of the joint pdf of $f_\tau(\tau)$.) The following definition corresponds to Definition 5.3.9 for the case that the feasible region is a convex region (*i.e.*, for the case that the feasible region can be described by a single convex polyhedron):

**Definition 5.3.25.-** The $\delta$-reduction of a polyhedron $P \subseteq R^{m+n}$ is the region $P_\delta = \{\delta \in R^n \mid \forall\tau \in P_\tau, (\tau, \delta) \in P\}$, where $P_\tau$ is the $\tau$-region of $P$.

Please refer to Figure 5.3.2. One can see that the $\delta$-reduction is the set of $\delta$ values that satisfy a given constraint $\Delta$ for all values of $\tau$. The following lemma gives a general, although not practical, procedure to obtain the $\delta$-reduction of a polyhedron.

**Lemma 5.3.10.-** Let $P \subseteq R^{m+n}$ be a polyhedron, then the $\delta$-reduction of $P$ is given by:

$$P_\delta = \bigcap_i P(\tau_i) \qquad \text{(Eq. 5.3.4)}$$

where $x_i \in P$, $\tau_i$ is the $\tau$-projections of $x_i$, $P(\tau_i)$ is the slice of $P$ at $\tau_i$, and the intersection is carried out over all $x_i \in P$.

**Proof.-** Let $\delta_1 \in P_\delta$, then according to Definition 5.3.25 $\delta_1$ must belong to $P_\delta$ because it must appear in every slice of $P$. Let $\delta_2 \in P_\delta$. According to Definition 5.3.25 $(\tau, \delta_2)$ belongs to $P$ for every $\tau$-projection $\tau$ of $P$. Then $\delta_2$ must be a point of the intersection. $\square$

We will need the following lemma later on:

**Lemma 5.3.11.-** Given a polyhedron $P \subseteq R^{m+n}$ and a point $x_0 \in P$, the slice $P(\tau_0)$, where $\tau_0$ is the $\tau$-projection of $x_0$, is a polyhedron in $R^n$.

**Proof.-** $P$ can be represented by a set of inequalities on variables $(\tau, \delta)$, where $\tau \in R^m$ and $\delta \in R^n$. $P(\tau_0)$ is represented by the same set of inequalities on variables $\delta$ by substituting $\tau$ by $\tau_0$, which describes a polyhedron in $R^n$. $\square$

The following Lemma states that the polyhedron reduction can be computed by considering only the extreme points of the polyhedron. Fortunately the number of extreme points of a polyhedron described by a finite set of linear inequalities is finite [109].

**Lemma 5.3.12.-** Let $P \subseteq R^{m+n}$ be a bounded polyhedron and let $\Psi$ be the set $\{\tau_i \in R^m | \, x_i=(\tau_i,\delta_j)$ is an extreme point of $P\}$ with $N$ elements, then the $\delta$-reduction of $P$ is given by:

$$P_\delta = P_\delta^e \equiv \bigcap_i^N P(\tau_i) \qquad \text{(Eq. 5.3.5)}$$

where $P(\tau_i)$ is the slice of $P$ at $\tau_i$.

**Proof.-** Notice that $\Psi$ is the set of $\tau$-projections of the extreme points of $P$. We want to show that the slice of $P$ at the $\tau$-projection of any point $x_o \in P$ contains $P_\delta^e$. This is clearly true if $P_\delta^e = \phi$ or if $x_o$ is an extreme point of $P$. We need to prove that this is also the case when $P_\delta^e$ is not empty and $x_o$ is not an extreme point of $P$. For a $\tau_i \in \Psi$, let us define the set of points $ext(\tau_i) = \{(\tau_i, \delta_j) \mid \delta_j$ is an extreme point of $P^e_\delta\}$. A point $x \in ext(\tau_i)$ is in $P$ because its $\tau$ component is an extreme point of $P$ and its $\delta$ component belongs to $P^e_\delta$ which is included in $P(\tau_i)$. Let us construct $P'$ as the convex hull of the set of points $\chi' = \bigcup_i^N ext(\tau_i)$ over all $\tau_i \in \Psi$. Notice that the $\tau$-projection set $\{\tau \mid (\tau,\delta) \in P'\}$ is the convex hull of $\Psi$ because the $\tau$-projection set of $\chi'$ is $\Psi$. But $\{\tau \mid (\tau,\delta) \in P\}$ is also the convex hull of $\Psi$. Thus for any $x_{o} = (\tau_0,\delta_0) \in P$ there exists at least one point $x'_{o} = (\tau_0,\delta'_0) \in P'$, therefore $P'(\tau_0) \subseteq P(\tau_0)$. Since by contruction $P'(\tau_0) = P^e_\delta$, then $P^e_\delta \subseteq P(\tau_o)$. □

To illustrate Lemma 5.3.12, consider the 2-dimensional polyhedron shown in Figure 5.3.6. The polyhedron has five vertices $v_1$, $v_2$, $v_3$, $v_4$ and $v_5$. The procedure stated in the lemma finds the $\delta$-reduction as the intersection of the slices at each of the vertices, which are shown in the figure as vertical dashed lines. For this example the $\delta$-reduction consists of point $\delta_0$.

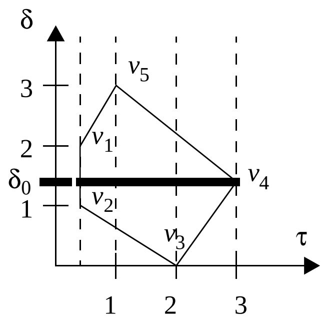


Figure 5.3.7 Projection $R_\delta$ of $f_{\delta 1 \delta 2}(\delta_1, \delta_2)$.

Thus far we have considered the case of a convex feasible region $R_f$. However $R_f$ is not convex in general, but by Theorem 5.3.9 it can be described as the union of non-overlapping convex polyhedra. Since we know how to obtain the $\delta$-reduction of a polyhedron, we now shall proceed to extend the $\delta$-reduction procedure to handle the more general feasible region. In the sequel we consider the feasible region $R_f$ of a TAFS problem for a cycle-invariant AOC complete graph.

**Definition 5.3.26.-** Given the feasible region $R_f \subseteq R^{m+n}$ of a TAFS problem, the $\tau$-projection of $R_f$ is $R_\tau = \{\tau \in R^m \mid \delta \in R^n, (\tau, \delta) \in R_f\}$.

**Definition 5.3.27.-** Given the feasible region $R_f \subseteq R^{m+n}$ of a TAFS problem, the $\delta$-reduction of $R_f$ is $R_\delta = \{\delta \in R^n \mid \forall\tau \in R_\tau, (\tau, \delta) \in R_f\}$.

**Definition 5.3.28.-** The slice of a feasible region $R_f \subseteq R^{m+n}$ at $\tau_0$, where $\tau_i \in R_\tau$, is $R_f(\tau_0) = \{\delta \in R^n \mid (\tau_0, \delta) \in R_f\}$.

**Lemma 5.3.13.-** The slice of $R_f$ at $\tau_0$ is $R_f(\tau_0) = \bigcup_i P_{fi}(\tau_0)$ .

**Proof.-** From Lemma 5.3.9, $R_f$ can be represented as the union of non-overlapping polyhedra $P_{fi}$. □

**Lemma 5.3.14.-** The $\delta$-reduction of a feasible region $R_f$ of a TAFS problem is

$$R_\delta = \bigcap_{\forall\tau_0 \in R_\tau} R_f(\tau_0) .$$

**Proof.-** If point $\delta_0$ belongs to the intersection, then it belongs to every slice of $R_f$. Thus $\delta_0$ satisfies Definition 5.3.27. Conversely if $\delta_0$ belongs to the reduction of $R_f$, then it must belong to every slice of $R_f$. □

**Definition 5.3.29.-** Given a feasible region $R_f = \bigcup_i P_{fi}$ of a TAFS problem, the set $\{x_j\}$ of extreme points of $R_f$ is the union of all extreme points of the non-overlapping polyhedra $P_{fi}$ that form $R_f$.

**Definition 5.3.30.-** Given a feasible region $R_f = \bigcup_i P_{fi}$ of a TAFS problem, and the $N$ τ-projections $\tau_j$ of the set of extreme points of $R_f$, a selection S($i_1$, … , $i_j$, … , $i_N$) of $R_f$ is a set of slices $\{P_{fi_1}(\tau_1)\ldots, P_{fi_j}(\tau_j), \ldots, P_{fi_n}(\tau_N)\}$ in which an arbitrary polyhedron $P_{fi_j}$ is selected for each $\tau_j$.

Thus in a selection of $R_f$, there is exactly one slice of $R_f$ at each τ-projection $\tau_j$ of an extreme point. Notice that the same polyhedron $P_{fi_j}$ can be selected for different $\tau_j$.

**Lemma 5.3.15.-** Given a feasible region of a TAFS problem represented by $M$ non-overlapping polyhedra $P_{fi}$ which has $N$ τ-projections $\tau_j$ of its set of extreme points, there are $M^N$ different possible selections.

**Proof.-** For each τ-projection $\tau_j$ any non-overlapping polyhedra $P_{fi}$ can be chosen (*i.e.*, the same $P_{fi}$ can be chosen for different $\tau_j$'s). □

In the following definition, we use the shorthand $S_l$ to denote a selection.

**Definition 5.3.31.-** The region of a selection $S_l = S(i_1, \ldots, i_N)$ of a feasible region is

$$RS_l = \bigcap_{j=1}^{N} P_{fi_j}.$$

**Lemma 5.3.16.-** Given the intersections of two different selections $S_{l1}$ and $S_{l2}$, $RS_{l1} \cap RS_{l2} = \varnothing$.

**Proof.-** Selections $S_{l1}$ and $S_{l2}$ contain at least one different $P_{fi}$ for a given $\tau_j$, say $P_{fi_{j_k}}(\tau_{j_k})$ and $P_{fi_{j_l}}(\tau_{j_l})$, $k \neq l$, which are non-overlapping. □

**Theorem 5.3.17.-** Let $R_f$ be the feasible region of a TAFS problem. The δ-reduction of $R_f$ is $R_\delta = \bigcup_{l_i} S_{l_i}$, where the union is over the $M^N$ selections of $R_f$.

**Proof.-** Each $S_{l_i}$ corresponds to the δ-reduction of the portion of $R_f$ corresponding to the selection of non-overlapping polyhedra $P_{fi}$ of $R_f$, therefore each $S_{l_i}$ belongs to the δ-reduction of $R_f$. Now we show that all the points of $R_\delta$ also belong to the union. Suppose there is such a point $\delta_0$, that is in $R_\delta$ but not in the union. Then for all $\tau \in R_\tau$, $(\tau, \delta_0) \in R_f$. In particular, if $\tau_j$ is the τ-projection of an extreme point of $R_f$, $(\tau_j, \delta_0) \in R_f$. But $\delta_0 \in P_{fi_i}(\tau_j)$. Because $\delta_0$ is not in the union (*i.e.*, there is no region of a selection involving $P_{fi_i}(\tau_j)$ which contains $\delta_0$), then there must exist a $\tau_0 \in R_\tau$ such that $(\tau_0, \delta_0) \notin R_f$, which is a contradiction. □

Theorem 5.3.17 suggests a procedure to obtain the reduction of the feasible region of a TAFS problem, thus yielding the solution of TAFS.

**Procedure 5.3.2.-** Given the feasible region $R_f$ of a TAFS problem for a cycle-invariant AOC complete graph, to find the δ-reduction of $R_f$:

1. Find the set $\{x_j\}$ of extreme points of $R_f$.
2. Find the slices $R_f(\tau_j)$ of $R_f$ at the τ-projections $\{\tau_j\}$ of the extreme points.
3. Obtain all the selections of $R_f$.
4. The δ-reduction $R_\delta$ of $R_f$ is the union of the non-overlapping regions of the selections of $R_f$.

Consider for example the feasible region shown in Figure 5.3.8, which consists of two convex non-overlapping polygons $P_{f1}$ and $P_{f2}$. Polygon $P_{f1}$ has four extreme points

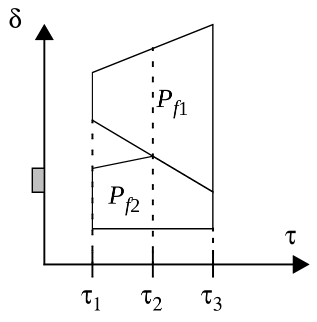


Figure 5.3.8 A feasible region.

and $P_{f2}$ has five, while the feasible region has eight (one is common to both $P_{f1}$ and $P_{f2}$). However there are only three τ-projections of the extreme points because some extreme points have the same τ-component.

Because there are three τ-projections of the extreme points and two polygons forming $R_f$, the number of possible selection is $2^3 = 8$. One selection is $\{P_{f2}(\tau_1), P_{f2}(\tau_2), P_{f1}(\tau_3)\}$ whose region is the shaded interval of δ values shown in Figure 5.3.8 (an interval or line segment is a 1-dimensional polyhedron). The other selections whose regions are non-empty are $\{P_{f1}(\tau_1), P_{f1}(\tau_2), P_{f1}(\tau_3)\}$ and $\{P_{f2}(\tau_1), P_{f2}(\tau_2), P_{f2}(\tau_3)\}$.

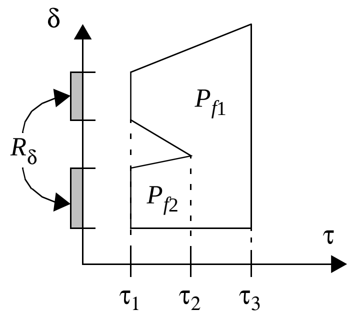


Figure 5.3.9 The δ-reduction of the feasible region shown in Figure 5.3.8.

The δ-reduction of the feasible region, shown in Figure 5.3.9, is the union of the regions of the three non-empty selections. It consists of two disconnected intervals of δ values.

Procedure 5.3.2 is correct based upon the results of this section, particularly Theorem 5.3.17. However the running complexity of Procedure 5.3.2 is proportional to the number of selections of $R_f$, which can be worst-case exponential in the number of extreme points of $R_f$, although in the examples that we have worked out most of the regions of selections are empty. The number of extreme points of a polyhedron can also be exponential on the dimension of the polyhedron [89]. However the τ-projection reduces the number of slices that need be considered. At the moment we do not have empirical results to be able to estimate the running time for the average case. Thus it is important for future work to conduct analysis of the running complexity of our Procedure for average cases and also to develop a more efficient δ-reduction algorithms.

Although different algorithms to find the extreme points of a polyhedron exist in the literature [7, 27, 109, 40, 8, 5], we have used Prof. Fukuda's cdd algorithm, which is discussed in [55]. We are very grateful to Prof. Fukuda for making his code available to us and for providing us with important pointers on this topic.

## 5.4 Bus arbitration interface example

In this section we apply the concepts developed in the previous sections of this chapter to analyze the bus arbitration interface design described in Chapter 3. We show that TAFS can be used to determine the feasibility of an interface design before the interface is even implemented.

Let us consider again the bus arbitration interface design shown in Figure 5.2.3. The complete graph contains some interface delays, labeled as $\delta_i$, which are unknown prior to the interface implementation. The purpose of TAFS is to determine in the first place if there is a possible assignment of non-negative values for the interface delays, and if so, to determine all possible assignments. All but two constraints ($\Delta_3$ and $\Delta_4$) are labeled with the interval $[0, \infty)$. Although for TAFS one would have to consider all the constraints, to be able to visualize the solution we shall concentrate on studying the effect of constraints labeled with intervals $\Delta_3$ and $\Delta_4$, which are the two critical timing constraints given in the VMEbus specification.

First we write the constraint equations (refer to Section 3.3.4) corresponding to $\Delta_3$ and $\Delta_4$. The cycle-invariant fork transition, for $k > 0$, associated with both constraints is signal transition $\underline{GA}$+. The constraint equations are given by the following expressions:

$$\{\max(\delta_4 + \delta_5 + \tau_a + \tau_b, \delta_6 + \tau_2)\} - \{\delta_3\} \subseteq \Delta_3$$
$$\{\max(\delta_4 + \delta_5 + \tau_a + \tau_b, \delta_6 + \tau_2)\} \subseteq \Delta_4$$

We proceed to apply the TAFS procedure discussed in the previous section. We linearize the max term (common to both equations) by considering two cases:

1. $\delta_4 + \delta_5 + \tau_a + \tau_b \geq \delta_6 + \tau_2$

$$\{\delta_4 + \delta_5 + \tau_a + \tau_b\} - \{\delta_3\} \subseteq \Delta_3$$
$$\{\delta_4 + \delta_5 + \tau_a + \tau_b\} \subseteq \Delta_4$$

2. $\delta_4 + \delta_5 + \tau_a + \tau_b \leq \delta_6 + \tau_2$

$$\{\delta_6 + \tau_2\} - \{\delta_3\} \subseteq \Delta_3$$
$$\{\delta_6 + \tau_2\} \subseteq \Delta_3$$

Using the following values for the projections of the known delays: $\tau_2 \in [15, 30]$, $\tau_a \in [20, 80]$, $\tau_b \in [40, 100]$, and the constraint windows $\Delta_3 = [30, \infty)$, and $\Delta_4 = [90, \infty)$, one can write the following two sets of linear inequalities:

Case 1:

$\tau_2 - \tau_a - \tau_b - \delta_4 - \delta_5 + \delta_6 \leq 0$
$- \tau_a - \tau_b + \delta_3 - \delta_4 - \delta_5 \leq -30$
$- \tau_a - \tau_b - \delta_4 - \delta_5 \leq -90$
$15 \leq \tau_2 \leq 30$
$20 \leq \tau_a \leq 80$
$40 \leq \tau_b \leq 100$
$\delta_i \geq 0$, for i = 3 to 6

Case 2:

$- \tau_2 + \tau_a + \tau_b + \delta_4 + \delta_5 - \delta_6 \leq 0$
$- \tau_2 + \delta_3 - \delta_6 \leq -30$
$- \tau_2 - \delta_6 \leq -90$
$15 \leq \tau_2 \leq 30$
$20 \leq \tau_a \leq 80$
$40 \leq \tau_b \leq 100$
$\delta_i \geq 0$, for i = 3 to 6

Because the delays $\tau_a$ and $\tau_b$ are independent, it is possible to combine them into a single delay $\tau_{ab}$, such that $60 \leq \tau_{ab} \leq 180$, to simplify our analysis. One can obtain the extreme points of the polyhedra described by the two set of inequalities using *cdd* [56]. The $\tau$-projections ($\tau_2$, $\tau_{ab}$) of the extreme points are: (15, 60), (15, 90), (15, 180), (30, 60), (30, 90), and (30, 180). Hence there are $2^6$ selections, but only 7 are non-empty:

$\{P_1(15, 60), P_1(15, 90), P_1(15, 180), P_1(30, 60), P_1(30, 90), P_1(30, 180)\}$
$\{P_1(15, 60), P_1(15, 90), P_1(15, 180), P_2(30, 60), P_1(30, 90), P_1(30, 180)\}$
$\{P_2(15, 60), P_1(15, 90), P_1(15, 180), P_2(30, 60), P_1(30, 90), P_1(30, 180)\}$
$\{P_2(15, 60), P_1(15, 90), P_1(15, 180), P_2(30, 60), P_2(30, 90), P_1(30, 180)\}$
$\{P_2(15, 60), P_2(15, 90), P_1(15, 180), P_2(30, 60), P_2(30, 90), P_1(30, 180)\}$
$\{P_2(15, 60), P_2(15, 90), P_1(15, 180), P_2(30, 60), P_2(30, 90), P_2(30, 180)\}$
$\{P_2(15, 60), P_2(15, 90), P_2(15, 180), P_2(30, 60), P_2(30, 90), P_2(30, 180)\}$

where $P_1$ and $P_2$ are the polyhedra described by cases 1 and 2 above respectively.

Figure 5.4.1 shows the union of the regions of the non-empty selections, which is a (non-convex) region in the $\{\delta_3, \delta_4, \delta_5, \delta_6\}$ space (one of the axis is labeled $\delta_4 + \delta_5$ to make possible to display the solution in three dimensions), which is the solution of TAFS. The δ-reduction is the volume bounded by planes that extend to infinity in the directions shown by the five pointers, reflecting the fact that arbitrary large delays are accommodated by the handshakes in the VMEbus and DMA bus arbitration protocols. Small values for the interface delays however can cause violations of the timing constraints. The relation between the δ's is shown in the polytope. For instance, the plane below the pointer starting at $(\delta_3, \delta_4+\delta_5, \delta_6) = (30, 60, 0)$ is the region where the path through $\delta_4$ and $\delta_5$ is too fast with respect to the $\delta_3$ path, which causes a violation of $\Delta_3$.

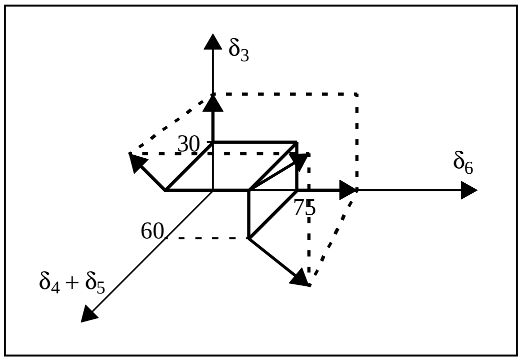


Figure 5.4.1 TAFS solution for $\Delta_3 = [30, \infty)$ and $\Delta_4 = [90, \infty)$.

Informally a delay-insensitive circuit is defined as a circuit whose correct operation is independent of circuit delays. The bus arbitration interface is clearly not delay-insensitive, otherwise its solution would consist of the whole positive octant.

If constraints $\Delta_3$ and $\Delta_4$ are narrowed to intervals [30, 100] and [90, 200], the solution is shown in Figure 5.4.2. In this case the δ-reduction is a convex region. One can see that the solution region in Figure 5.4.2 is inside the solution region shown in Figure 5.4.1 as expected.

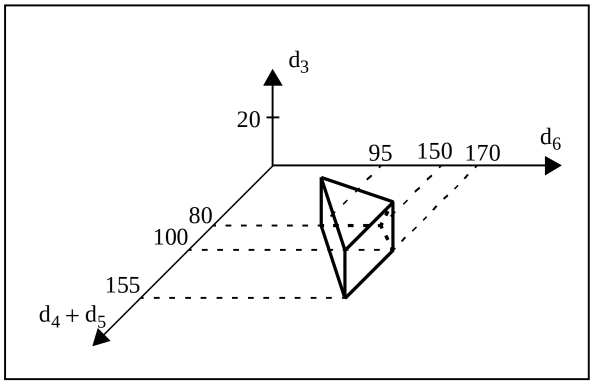


Figure 5.4.2 TAFS solution for $\Delta_3 = [30, 100]$ and $\Delta_4 = [90, 200]$.

## 5.5 Summary

In this chapter we formulated the timing analysis for synthesis problem of an interface design described by a complete graph. The underlying net of the complete graph is partially specified: only the projection of the joint pdf of a sub-set of the delays is known, and the task is to determine the maximal (*i.e.* largest) projection of a joint pdf that can characterize the rest of the delays so that all the constraint rules of the complete graph are satisfied.

We have shown that TAFS problem is complex because it involves non-convex programming due to the presence of non-linearities in the constraint equations that describe the conditions that must hold for the constraints to be satisfied. Our main contribution is to have developed a strategy that describes the non-convex regions by the union of non-overlapping convex regions, so that standard techniques from convex programming can be applied.

We have also shown that for such a representation, the solution of TAFS, called $\delta$-reduction, for cycle-invariant AOC complete graphs amounts to finding the union of a finite number of intersections of convex regions. Although our proposed procedure that obtains the $\delta$-reduction is not computationally efficient, we believe that the development of more efficient algorithms is a promising area of future research.

# Chapter 6

# Conclusions

## 6.1 Overview of the main contributions

In this dissertation we have presented some results on the topic of timing specification, timing verification and timing analysis for synthesis of hardware interface circuits.

Firstly, we have proposed an interface specification graph suitable to specify interface specifications. Our interface specification graphs include two types of timing relationships with different semantics: timing delays and timing constraints. The operational aspect of an interface specification, timing delays, is captured using a timed signal transition graph (STG). Other timed STG's have been proposed previously in the literature, however our STG model, which has an underlying probabilistic timed Petri net, is more general in the sense that a delay is represented by a random variable while in previous efforts it is characterized by an interval $[d_{min}, d_{max}]$. One important feature of our timed STG's is that they allow us to describe delay correlation. Moreover the probabilistic view of delays provides us with a statistical view of the timed behavior that proves essential for a reliability analysis.

Secondly we have developed a formal timing verification technique that can check if a system satisfies a set of given timing constraints. The technique is applicable to closed systems, that is systems that are self-contained and do not require to interact with an environment, represented by complete graphs. The type of closed systems that so far we are able to verify are the sub-class of AOC complete graphs, in which only AND and OR causality are allowed, such that each pair of transitions associated with a timing constraint has a cycle-invariant fork transition. This is not a very serious restriction. On the one hand the presence of a cycle-invariant fork transition guarantees that the timed behavior of the circuit is repeatable and thus predictable which is a desirable property of a circuit (although as was mentioned in this dissertation, the converse is not in general true; that is, the absence of a cycle-invariant fork transition does not imply that the timed behavior is not repeatable). On the other hand more complex behaviors, for instance behaviors with free choice, can be transformed into a collection of simple behaviors with only AND and OR causality by using the idea of processes (although there is potentially the problem of obtaining a large number of such simple behaviors). This restriction has been met by the components that we have modeled so far. Extending our timing verification technique is of course a valuable direction to pursue in future work. It is important to note two major features:

1. First, in contrast with previous interface timing verification techniques, for the case that some of the timing constraints are not satisfied, our technique can determine the probability that such constraints would be violated. This information can be used to support a design for manufacturability methodology.

2. Second the probabilistic treatment admits taking timing correlation into consideration, thus delivering a more accurate analysis as was illustrated in this dissertation.

Finally, we developed a technique that we have called timing analysis for synthesis (TAFS). With TAFS it is possible to investigate the feasibility of an interface design prior to interface synthesis by finding the tightest bounds of the interface delays such that all the timing constraints are satisfied. The solution (interface delay) space is not convex in general which makes this problem very difficult. A key result of our work is that we express the solution space as the union of non-overlapping convex regions thus allowing us to use convex techniques. TAFS facilitates the exploration of interface designs without having to go through the expensive process of producing a corresponding implementation to be able to ascertain that the interface circuit will satisfy the given timing constraints.

## 6.2 Future work

In this section we present some directions that we consider worth pursuing as future research. Some of our contributions, due to their novelty, are in an early stage of development and therefore more work is needed to achieve maturity. Predicting the future is always a risky business, and thus we do not claim to hold the truth and we encourage the reader to explore the directions he or she considers important.

A first direction, as hinted in the previous section, is to extend our techniques so that they can handle richer timed behaviors. A clear candidate is the class of AND/OR causality with free choice. Another interesting problem is to extend the techniques so they can handle constrained transitions for which there is no cycle-invariant fork transition; although we think that maintaining repeatibility of the behavior is highly desirable.

Improving the efficiency of our techniques is another important direction otherwise our techniques cannot directly be applied to large-scale problems. The general problem is NP-complete. One can attack this difficulty using two different strategies:

1. By developing heuristics that although they may not guarantee an exact solution they should not find a wrong solution; in the probabilistic timing verification, a wrong solution is to underestimate the probability that a timing constraint would be violated; in the timing analysis for synthesis a wrong solution is to determine that there is a solution space for an infeasible interface design.

2. By identifying special cases for which the time complexity of the techniques is significantly reduced. As a simple example, if only single causality is allowed

TAFS can be solved by a linear program. Another example is to restrict the type of probability density functions that characterize the circuit delays (e.g. Gaussian pdf's).

We also believe that our probabilistic timed Petri nets can be used in related areas such as real-time systems, distributed systems, computer networks, *etc*. As an example, we show in Appendix A the modelling and analysis of a Seitz arbiter using our probabilistic timed STG's.

A final direction we suggest is to investigate if our techniques can be used in a hierarchical methodology. In such a methodology one can use the fact that a set of modules has been verified to work correctly to prove that a system composed of such modules also works correctly. This is a powerful mechanism to contain the ever-increasing complexity of hardware systems. An example is a delay-insensitive methodology that uses circuits which behave correctly in the presence of arbitrary delay variations. (Notice that TAFS can determine if an interface design is delay-insensitive, namely when the solution space does not restrict any of the interface delays.)

# Bibliography

[1] M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chila, G. Conte and A. Cumani, “On Petri nets with stochastic timing”, in *Proceedings of the International Workshop on Timed Petri nets*, pp. 80–87, 1985.

[2] T. Amon and G. Borriello, “An approach to symbolic timing verification,” in *Proceedings of the 29th ACM/IEEE Design Automation Conference*, pp. 410–413, 1992.

[3] T. Amon, H. Hulgaard, S. M. Burns, and G. Borriello, “An algorithm for exact bounds on the time separation of events in concurrent systems,” in *Proceedings of the International Conference on Computer Design*, pp. 166–173, 1993.

[4] Analog Devices, *ADSP-21060/62 SHARC data sheet*, Norwood, MA, November 1994.

[5] D. Avis, “A C implementation of the reverse search vertex enumeration algorithm”, Technical report, School of Computer Science, McGill University, Montreal, Canada, 1993.

[6] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity*. Wiley series in Probability and Mathematical Statistics. New York:Wiley and Sons, 1992.

[7] M. L.Balinski, “An algorithm for finding all vertices of convex polyhedral sets”, *Journal of the Society of Industrial Applied Mathematics*, Vol. 9, No. 1, pp. 72–88, March 1961.

[8] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, “The quickhull algorithm for convex hulls”, Technical report GCC53, The Geometry Center, Minnesota, U.S.A., 1993.

[9] M. Berkelaar, “Statistical delay calculation”, in Notes of the International Workshop on Logic Synthesis”, Lake Tahoe, May 1997.

[10] M. Berkelaar, “Statistical delay calculation, a linear time method”, in Proc. of the International Workshop on Timing Analysis TAU’97, pp 15–24, November 1997.

[11] B. Berthomieu and M. Diaz, "Modeling and verification of time dependent systems using time petri nets," *IEEE Transactions on Software Engineering*, vol. 17, pp. 259–273, March 1991.

[12] E. Best and J. Desel, "Partial order behavior and structure of Petri nets", *Formal Aspects of Computing*, vol. 2, pp. 123–138, 1990.

[13] W. P. Birmingham and D. P. Siewiorek, "Single board computer synthesis," in *Expert Systems for Engineering Design* (M. D. Rychener, ed.), chapter 5, pp. 113–139, Boston: Academic Press, 1988.

[14] W. P. Birmingham, A. P. Gupta, and D. P. Siewiorek, "The MICON system for computer design," *IEEE Micro*, vol. 9, pp. 61–67, October 1989.

[15] G. V. Bochmann, "Hardware specification with temporal logic: An example," *IEEE Transactions on Computers*, vol. C-31, pp. 223–231, Mar. 1982.

[16] G. Borriello and R. H. Katz, "Synthesis and optimization of interface transducer logic," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 274–277, 1987.

[17] J. A. Brzozowski, T. Gahlinger, and F. Mavaddat, "Consistency and satisfiability of waveform timing specifications," *Networks*, vol. 21, pp. 91–107, Jan. 1991.

[18] B. Büeler, A. Enge, K. Fukuda and H.-J. Lüthi, "Exact Volume Computation for Polytopes: A Practical Study," Technical report, Institute for Operations Research, Swiss Federal Institute of Technology, Zurich, Switzerland.

[19] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang, "Symbolic model checking: $10^{20}$ states and beyond," in *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, June 1990.

[20] T. M. Burks and K. A. Sakallah, "Min-max linear programming and the timing analysis of digital circuits", in *Proceedings of the International Conference on Computer Design*, pp. 152–155, 1993.

[21] S. M. Burns and A. J. Martin, "Syntax-directed translation of concurrent programs into self-timed circuits," in *Proceedings of the Fifth MIT Conference on Advanced Research in VLSI* (J. Allen and F. Leighton, eds.), pp. 35–50, Cambridge, Massachussetts: MIT Press, 1988.

[22] S. M. Burns, "Performance analysis and optimization of asynchronous circuits," PhD dissertation, Tech. Rep. Caltech-CS-TR-91-01, December 1990.

[23] S. M. Burns, Personal communication, Intel Corp., January 1997.

[24] R. Camposano, L. F. Saunders, and R. M. Tabet, “VHDL as input for High-level synthesis,” *IEEE Design & Test of Computers*, pp. 43–49, Mar. 1991.

[25] E. Cerny and K. Khordoc, “Interface Specifications with conjunctive timing constraints: realizability and compatibility”, in *Proceedings of the 2nd AMAST Workshop on Real-Time Systems*, 1995.

[26] B. Chandrasekaran and S. Mittal, “Deep versus compiled knowledge approaches to diagnostic problem-solving,” in *Development in Expert Systems*, London: Academic Press, 1984.

[27] N. V. Chernikova, “An algorithm for finding a general formula for non-negative solutions of systems of linear inequalities”, *U.S.S.R. Computational Mathematics and Mathematical Physics*, No. 5, pp. 228–233, 1965.

[28] Chronology. *Timing Designer user’s manual*. Beaverton, Oregon, 1993.

[29] T.-A. Chu, “On the models for designing VLSI asynchronous digital systems,” *INTEGRATION, the VLSI journal*, no. 4, pp. 99–113, 1986.

[30] V. Chvátal. Linear Programming. New York:Freeman, 1983.

[31] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness, “Verification of the Futurebus+ cache coherence protocol,” In L. Claesen, editor, *Proceedings of the Eleventh Symposium on Computer Hardware Description Languages and their Applications*. North-Holland, April 1993.

[32] J. Cohen and T. Hickey, “Two algorithms for determining volumes of convex polyhedra.” *Journal of the ACM*, vol. 26, No. 3, pp. 401–414, July 1979.

[33] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*. Cambridge,MA: The MIT Press & McGraw-Hill, 1990.

[34] D. del Corso, H. Kirmann, and J. D. Nicoud, *Microcomputer buses and links*. London: Academic Press, 1986.

[35] D. L. Dill and E. M. Clarke, “Automatic verification of asynchronous circuits using temporal logic,” in *Proceedings of the Chapell Hill Conference on VLSI* (H. Fuchs, ed.), pp. 127–143, Computer Science Press, 1985.

[36] D. L. Dill, S. M. Novick, and R. F. Sproull, “Automatic verification of speed-independent circuits with petri net specifications,” in *Proceedings of the International Conference on Computer Design*, pp. 212–216, 1989.

[37] N. J. Dimopoulos, “On the structure of the Homogeneous Multiprocessor,” *IEEE Transactions on Computers*, vol. C-34, pp. 141–150, February 1985.

[38] N. J. Dimopoulos, K. F. Li, and E. G. Manning, “DAME: A rule based designer of microprocessor based systems,” in *Proceedings of the 2nd International Conference on Industrial & Computer Engineering Applications of Artificial Intelligence and Expert Systems*, pp. 486–492, June 1989.

[39] N. J. Dimopoulos, B. Huber, K. F. Li, D. Caughey, M. Escalante, D. Li, R. Burnett, and E. G. Manning, “Modelling components in DAME,” in *Proceedings of the 3rd International Conference on Industrial & Computer Engineering Applications of Artificial Intelligence and Expert Systems*, pp. 716–725, July 1990.

[40] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, 1987.

[41] M. Escalante, N. J. Dimopoulos, B. Huber, K. F. Li, D. Li, and E. G. Manning, “Generic design rules for the design of microprocessor based systems in DAME: Bus arbitration subsystem,” in *Proceedings of the International Symposium on Circuits and Systems*, pp. 3166–3169, June 1991.

[42] M. A. Escalante, “Bus arbitration modelling and design in DAME: An expert microprocessor-based-systems designer,” M. A. Sc. thesis, University of Victoria, Department of Electrical and Computer Engineering, 1991.

[43] M. A. Escalante, B. Huber, N. J. Dimopoulos, K. F. Li, D. Li, and E. G. Manning, “Bus arbitration modelling and design in DAME: An expert microprocessor-based-systems-designer,” in *Proceedings of the International Symposium on Artificial Intelligence*, pp. 238–244, Nov. 1991.

[44] M. A. Escalante, N. J. Dimopoulos, D. M. Miller, K. F. Li, and E. G. Manning, “The implementor subsystem in DAME: Using OASIS to complete the design automation of microprocessor-based systems,” in *Proceedings of the Canadian Conference on VLSI*, pp. 139–146, October 1992.

[45] M. A. Escalante and M. H. M. Cheng, “Decomposing signal transition graphs,” in *Proceedings of the Canadian Conference on VLSI*, pp. 3B-19–3B-24, Nov. 1993.

[46] M. A. Escalante and N. J. Dimopoulos, “Timed asynchronous interface design in microprocessor-based systems,” in *Proceedings of the Canadian Conference on VLSI*, pp. 7-1–7-6, Nov. 1993.

[47] M. A. Escalante and N. J. Dimopoulos, “Timing analysis for synthesis in microprocessor interface design,” in *Proceedings of the Seventh High-Level Synthesis Symposium*, pp. 23–28, May. 1994.

[48] M. A. Escalante, “A probabilistic approach to timing analysis for synthesis and its application to microprocessor interface design”, Technical Report ECE 94-6, Dept. of Electrical and Computer Engineering, University of Victoria, June 1994.

[49] M. A. Escalante, “A probabilistic timing analysis in microprocessor interface design”, in *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 277–280, 1995.

[50] M. A. Escalante and N. J. Dimopoulos, “Assessing the Feasibility of Hardware Interface Designs before their Implementation”, *In Proceedings of the Asian South-Pacific Design Automation Conference (ASP-DAC’95)*, Chiba, Japan, August 1995.

[51] M. A. Escalante, N. Dimopoulos, D. Gyuroff, and H. Müller, “Timing analysis for synthesis of hardware interface controllers using timed signal transition graphs”, in *Proceedings of the 8th International Workshop on Petri nets and Performance Models*, pp. 232–240, 1995.

[52] M. A. Escalante and N. J. Dimopoulos, “Modeling Timing Correlation and the Accurate Timing Verification of Digital Interface Circuits”, *In Proceedings of the Mid-West Symposium on Circuits and Systems*, Des Moines, Iowa, August 1996.

[53] M. A. Escalante, L. Lavagno, and N. J. Dimopoulos, “Performance Analysis of an Arbiter using Probabilistic Timed Petri Nets”, *In Proceedings of the International Workshop on Logic Synthesis*, Lake Tahoe, May 1997.

[54] J. Esparza, “A partial order approach to model checking”, Manuscript, Institut für Informatik, Universität Hildesheim, Germany, February 1993.

[55] K. Fukuda, “cdd.c: A C-implementation of the double description method for computing all vertices and extremal rays of a convex polyhedron given by a system of linear inequalities”, Technical report, Department of Mathematics, Swiss Federal Institute of Technology, Zurich, Switzerland, 1993.

[56] K. Fukuda, “Cdd user’s manual,” Institute for Operations Research, Swiss Federal Institute of Technology, Zurich, Switzerland, 1996.

[57] M. R. Garey and D. S. Johnshon. Computers and Intractability: A guide to the theory of NP-Completeness. W. H. Freeman and Company. 1979.

[58] A. Gibbons. *Algorithmic Graph Theory.* Cambridge:Cambridge University Press, 1989.

[59] P. Gritzmann and V. Klee, “On the complexity of some basic problems in computational convexity: Volume and mixed volumes,” Technical Report 94-07, Universität Trier, 1994.

[60] B. Grünbaum. *Convex Polytopes*. London:John Wiley and Sons, 1967.

[61] J. Gunawardena, “Causal automata,” *Theoretical Computer Science*, vol. 101, no. 2, pp. 265–288, 1992.

[62] B. T. Hailpern, “Verifying concurrent processes using temporal logic,” Tech. Rep. 195, Computer Systems Laboratory, Stanford Univ., Stanford, CA, 1980.

[63] C. A. R. Hoare, “Communicating sequential processes,” *Communications of the ACM*, vol. 21, pp. 666–677, Aug. 1978.

[64] L.A. Hollaar, “Direct implementation of asynchronous control units,” *IEEE Transactions on Computers*, vol. C-31, pp. 1133–1141, December 1982.

[65] B. Huber, M. Escalante, D. Caughey, N. J. Dimopoulos, K. F. Li, D. Li, and E. G. Manning, “Microprocessor components and signal behavior modelling in DAME,” in *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, vol. 1, pp. 19.4.1–19.4.4, September 1990.

[66] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello, “Practical applications of an efficient time separation of events algorithm,” in *Proceedings of the International Conference on Computer-Aided Design*, pp. 146–151, IEEE, 1993.

[67] H. Hulgaard, *Timing analysis and verification of timed asynchronous circuits*. PhD dissertation, University of Washington, 1995.

[68] E. Hyvönen, “Constraint reasoning based on interval arithmetic: the tolerance propagation approach,” *Artificial Intelligence*, vol. 58, pp. 71–112, 1992.

[69] IEEE. *IEEE Standard for a Versatile Backplane Bus: VMEBus*. New York: IEEE Press, 1988.

[70] H. Jyu, S. Malik, S. Devadas, and K. Keutzer, “Statistical timing analysis of combinational circuits”, *IEEE Trans. on VLSI Systems*, vol. 1, No. 2, pp. 126–137, Jun. 1993.

[71] G. Juanole and Y. Atamna, “Dealing with arbitrary time distributions with the stochastic timed Petri net model - Applications to queueing systems”, in *Proceedings of the Fourth International Workshop on Petri Nets and Performance Models*, pp. 166–173, December 1991.

[72] K. Khordoc and E. Cerny, “Modeling Cell-processing hardware with action diagrams”, in *Proceedings of the International Symposium on Circuits and Systems*, June 1994.

[73] L. Kucera, *Combinatorial Algorithms*, Bristol:Adam Hilger, 1990.

[74] Y.-H. Kuo, L. Kung, C.-C. Tzeng, G.-H. Jeng, and W.-K. Chia, “KMDS: An expert system for integrated hardware/software design of microprocessor-based digital systems,” *IEEE Micro*, vol. 11, pp. 32–92, August 1991.

[75] J.B. Lasserre, “An analytical expression and an algorithm for the volume of a convex polyhedron in $R^n$,” *J. of Optimization Theory and Applications*, vol. 39, No. 3, pp. 363–377, 1983.

[76] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli, “Algorithms for synthesis of hazard-free asynchronous circuits,” in *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pp. 302–308, June 1991.

[77] L. Lavagno, C. W. Moon, R. K. Brayton, and A. Sangiovannin-Vincentelli, “Solving the state assignment problem for signal transition graphs,” in *Proceedings of the 29th ACM/IEEE Design Automation Conference*, pp. 568–572, 1992.

[78] L. Lavagno and A. Sangiovanni-Vincentelli, “Linear programming for optimum hazard elimination in asynchronous circuits,” in *Proceedings of the International Conference on Computer Design*, pp. 275–278, 1992.

[79] L. Lavagno, “Synthesis and testing of bounded wire delay asynchronous circuits from signal transition graphs,” Tech. Rep. UCB/ERL M92/140, U.C. Berkeley, November 1992.

[80] J. Lawrence, “Polytope volume computation,” *Mathematics of Computation*, vol. 57, No. 195, pp. 259–271, 1991.

[81] Y. Liu, “Reasoning about asynchronous designs in CCS,” Tech. Rep. No. 92-492-30, University of Calgary, Department of Electrical and Computer Engineering, Calgary, Alberta, 1992.

[82] K. L. McMillan and D. L. Dill, “Algorithms for interface timing verification,” in *Proceedings of the International Conference on Computer Design*, pp. 48–51, 1992.

[83] K. L. McMillan and J. Schwalbe, “Formal verification of the Encore Gigamax cache consistency protocols,” in the Proceedings of the International Symposium on Shared Memory Multiprocessors, April 1991.

[84] M. A. Marsan, G. Balbo, and G. Conte, “A class of Generalized Stochastic Petri nets for the performance evaluation of multiprocessor systems”, *ACM Transactions on Computer Systems*, Vol. 2, 1984.

[85] A. J. Martin, S. M. Burns, T. K. Lee, D. Borkovic, and P. J. Hazenwindus, “The design of an asynchronous microprocessor,” in *Proceedings of the Decennial Caltech Conference on VLSI* (C. L. Seitz, ed.), pp. 351–373, Cambridge, Massachussetts: MIT Press, 1989.

[86] A. J. Martin, “Programming in VLSI: From communicating processes to delay-insensitive circuits,” in *UT Year of Programming Institute on Concurrent Programming* (C. A. H. Hoare, ed.), pp. 1–64, Reading, Massachussetts: Addison-Wesley, 1990.

[87] A. J. Martin, “Synthesis of asynchronous VLSI circuits,” in *Formal methods for VLSI design* (J. Staunstrup, ed.), ch. 5, pp. 237–283, North-Holland, 1990.

[88] F. Maruyama and M. Fujita, “Hardware verification,” *IEEE Computer*, pp. 22–32, Feb. 1985.

[89] T. H. Mattheis and D. S. Rubin, “A survey and comparison of methods for finding all vertices of convex polyhedral sets”, *Journal of Mathematics of Operation Research*, Vol. 5, No. 2, pp. 167–185, May 1980.

[90] Philip M. Merlin and David J. Farber, “Recoverability of communication protocols - Implications of a theoretical study”, *IEEE Transactions on Communications*, pp. 1036–1043, September 1976.

[91] Philip M. Merlin, “Specification and validation of protocols”, *IEEE Transactions on Communications*, vol. COM-27, No. 11, pp. 1671–1680, November 1979.

[92] R. Milner, *Communication and Concurrency*, Series in Computer Science, Hertfordshire: Prentice Hall, 1989.

[93] C. E. Molnar, T.-P. Fang, and F. U. Rosenberger, “Synthesis of delay-insensitive modules,” in *Proceedings of the Chapell Hill Conference on VLSI* (H. Fuchs, ed.), pp. 67–86, Computer Science Press, 1985.

[94] B. Moszkowski, “A temporal logic for multilevel reasoning about hardware,” *IEEE Computer*, pp. 10–21, Feb. 1985.

[95] Motorola, *MC68010 microprocessor user’s manual*, Austin, Texas, August 1983.

[96] T. Murata, “Petri nets: Properties, analysis and applications,” *Proceedings of the IEEE*, vol. 77, pp. 541–580, Apr. 1989.

[97] C. Myers and T. H.-Y. Meng, “Synthesis of timed asynchronous circuits,” in *Proceedings of the International Conference on Computer Design*, pp. 279–284, 1992.

[98] C. J. Myers and T. H.-Y. Meng, “Synthesis of timed asynchronous circuits,” *IEEE Transactions on VLSI Systems*, vol. 1, no. 2, pp. 106–119, June 1993.

[99] F. N. Najm, R. Burch, P. Yang, and I. N. Hajj, “Probabilistic simulation for reliability of CMOS VLSI circuits”, *IEEE Transactions on CAD*, Vol. 9, No. 4, pp. 439–450, April 1990.

[100] F. N. Najm, “Feedback, correlation, and delay concerns in the power estimation of VLSI circuits,” in *Proceedings of the 32th ACM/IEEE Design Automation Conference*, pp. 612–617, June 1995.

[101] J. A. Nestor and D. E. Thomas, “Behavioral synthesis with interfaces,” in *Proceedings of the International Conference on Computer-Aided Design*, pp. 112–115, 1986.

[102] K. Okumura, “A formal protocol conversion method,” in *Proceedings ACM SIGCOMM*, pp. 30–37, 1986.

[103] E.-R. Olderog, “Nets, terms and formulas: Three views of concurrent processes and their relationship,” Tech. Report, Universität Oldenburg, FB Informatik, 1989.

[104] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

[105] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, 2nd. edition. New York: McGraw-Hill, 1984.

[106] P. Z. Peebles. *Probability, random variables, and random signal principles*, 3rd ed. New York:McGraw-Hill, 1993.

[107] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Englewoods Cliffs, NJ: Prentice Hall, 1981.

[108] C. A. Petri, “Non-sequential Processes”, Internal Report GMD-ISF-77-5, Gesellschaft für Informatik and Datenver-arbeitung, Bonn, Germany, 1977.

[109] F. P. Preparata and M. I. Shamos, *Computational Geometry: An introduction*. New York: Springer-Verlag, 1985.

[110] C. V. Ramamoorthy and Gary S. Ho, “Performance Evaluation of Asynchronous concurrent systems using Petri nets”, *IEEE Transactions on Software Engineering*, vol. SE-6, No. 5, pp. 440–449, September 1980.

[111] C. Ramchandani, “Analysis of asynchronous concurrent systems by Petri nets”, Project MAC, TR-120, MIT, Cambridge, MA, 1974.

[112] H. Ratschek and J. Rokne, *Computer Methods for the Range of Functions*. Chichester, England: Ellis Horwood, 1984.

[113] W. Reisig, *Petri nets: An Introduction*, Springer-Verlag, Berlin, 1985.

[114] T. G. Rokicki, *Representing and Modeling Digital Circuits*. PhD dissertation, Stanford University, Dec. 1993.

[115] L. Y. Rosenblum and A. V. Yakovlev, “Signal graphs: From self-timed to timed ones,” in *Proceedings of the International Workshop on Timed Petri Nets*, pp. 199–207, IEEE Computer Society Press, July 1985.

[116] C. L. Seitz, *System Timing*, chapter 7, pp. 218–262, Reading, Massachusetts: Addison-Wesley, 1980.

[117] J. Sifakis, “Performance evaluation of systems using nets”, in *Net Theory and Applications*, Lecture Notes in Computer Science, Springer-Verlag, pp. 307–319, 1980.

[118] J. L. A. van de Snepscheut, *Trace theory and VLSI design*. No. 200 in Lecture Notes in Computer Science, Berlin: Springer-Verlag, 1985.

[119] H. S. Stone, *Microcomputer Interfacing*. Reading, Massachussetts: Addison-Wesley, 1982.

[120] Texas Instruments, *MOS memory data book*, Dallas, Texas, 1988.

[121] J. T. Udding, “A formal model for defining and classifying delay-insensitive circuits and systems,” *Distributed Computing*, vol. 1, pp. 197–204, 1986.

[122] P. Van Beek, “Reasoning about qualitative temporal information,” *Artificial Intelligence*, vol. 58, pp. 297–326, 1992.

[123] P. Vanbekbergen, F. Catthoor, G. Goosens, and H. de Man, “Optimized synthesis of asynchronous control circuits from graph-theoretic specifications,” in *Proceedings of the International Conference on Computer-Aided Design*, pp. 184–187, 1990.

[124] P. Vanbekbergen, *Synthesis of Asynchronous Controllers from Graph-theoretic Specifications*. PhD dissertation, Katholieke Universiteit Leuven, Sept. 1993.

[125] P. Vanbekbergen, G. Goosens, and B. Lin, “Modeling and synthesis of timed asynchronous circuits”, in *Proceedings of the European Design Automation Conference*, pp. 460–465, 1994.

[126] W. M. P. Van Der Aalst, “Interval timed coloured Petri nets and their analysis”, in 14th. International Conferenco on the Application and Theory of Petri Nets, No. 691 in Lecture Notes in Computer Science, Berlin: Springer-Verlag, 1993.

[127] P. Van Hentenryck, H. Simonis, and M. Dincbas, “Constraint satisfaction using constraint logic programming,” *Artificial Intelligence*, vol. 58, pp. 113–159, 1992.

[128] E. Walkup and G. Borriello, “Interface timing verification with application to synthesis,” in *Proceedings of the Design Automation Conference*, pp. 106–112, 1994.

[129] P. H. Winston. *Artificial Intelligence*, second edition. Reading, MA:Addisson-Wesley, 1984.

[130] W. Wolf, *Modern VLSI design: A systems approach*. Englewoods Cliffs, NJ: Prentice Hall, 1994.

[131] A. V. Yakovlev, “On limitations and extensions of STG model for designing asynchronous control circuits,” in *Proceedings of the International Conference on Computer Design*, pp. 396–400, 1992.

[132] T.-Y. Yen, A. Ishii, A. Casavant, and W. Wolf, “Efficient algorithms for interface timing verification”, in *Proceedings of the European Design Automation Conference*, pp. 34–39, 1994.

[133] J. Zejda and E. Cerny, “Gate-level timing verification using window narrowing”, in *Proceedings of the European Design Automation Conference*, pp. 374–379, 1994.

[134] J. J. Zhu and R. T. Denton, “Timed Petri nets and their application,” in *Proceedings of the Military Communications Conference MILCOM*, pp. 195–199, Oct. 1988.

# Appendix A

# Performance analysis of an arbiter

## A.1 Introduction

Interval timing analysis has been used to determine the worst-case scenario of the performance of asynchronous circuits modeled by timed signal transition graphs [3]. There are some situations in which a worst-case analysis is not very appropriate. For instance if some of the involved delays are unbounded.

In this appendix we consider the problem of modeling the performance of an arbiter. An ideal 2-way arbiter controls access to a shared resource that can only service one client at a time. Such an arbiter can accept up to two requests at any time, but it will produce at most one grant even if the requests arrive simultaneously.

A typical circuit that implements an arbiter is the Seitz arbiter shown in Figure A.1.1. If only one of the requests is generated, the corresponding grant is produced after some delay. However if both requests arrive at (about) the same time, the NAND latch may enter a metastable state, and the resolution time $\tau_m$, after which only one of the grants is generated, can be arbitrarily long. The probability density function that describes the time $\tau_m$ when a circuit that has entered a metastable behavior leaves such state is given by (cf. [116]):

$$f_{\tau m}(\tau_m) = C\, e^{-K\tau m} \qquad \text{(Eq. A.1.1)}$$

where $C$ and $K$ are constants that depend on properties of the circuit elements. Notice that in the Seitz' arbiter, the differential detection circuit after the NAND latch always exhibits a well defined binary output, not being affected by the metastable behavior that may take place in the SR latch.

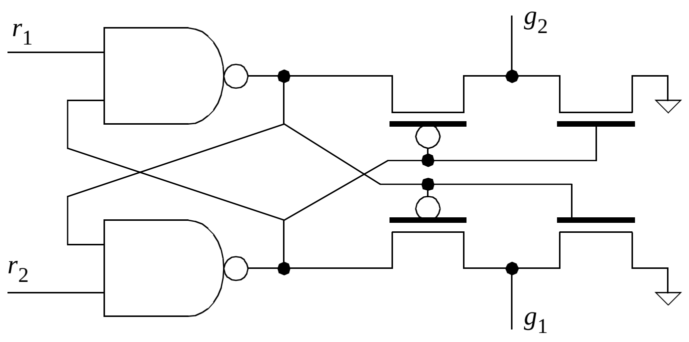


Figure A.1.1 Seitz' arbiter.

If one wants to determine the worst delay of a grant from a request for the Seitz' arbiter, the answer would be "arbitrarily long", which lacks a quantitative notion. Instead of using a timed Petri net with intervals associated with its places which has the limitation that can only characterize a worst-case scenario, we propose to model the metastable behavior using a probabilistic Petri net in which random variables are associated with its places [48, 49] because it allows us to quantify a possibly unbounded delay by obtaining its probability density function (pdf).

Figure A.1.2 shows a partial timed Petri net [117, 51] that represents the *timed* behavior of an ideal arbiter. In interval *timed* Petri nets, a compact non-empty time interval is associated with each place [51]. A transition fires immediately when all its input places have a *visible* token. When a transition fires, it consumes the tokens on its input places, and sends tokens to its output places. A place labeled with interval $\Delta_i$ that receives a token at time $\tau$, will make the token visible at time $\tau + \tau_i$, where $\tau_i \in \Delta_i$.

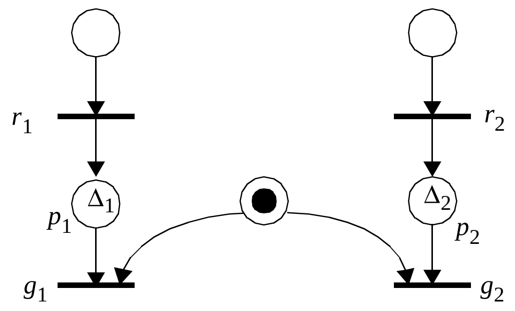


Figure A.1.2 Arbiter.

To understand the net shown in Figure A.1.2, let us assume that the token shown in the common input place to transitions $g_1$ and $g_2$ is already visible. Suppose that a token is made visible at the input place of $r_1$. Then transition $r_1$ fires and sends a token to place $p_1$ labeled with interval $\Delta_1$. When the token *matures* (i.e., becomes visible) in place $p_1$, and if there is no visible token at place $p_2$ (labeled with interval $\Delta_2$), then transition $g_1$ fires. Thus the grant enabled by the the first visible token (at place $p_1$ or place $p_2$) is the only one that fires. If tokens at both places $p_1$ and $p_2$ mature exactly at the same time, one of the grant transitions $g_1$ or $g_2$ is chosen to fire non-deterministically.

Clearly this Petri net cannot model the richer behavior of the Seitz' arbiter, since it does not distinguish between meta-stability and normal (digital) behavior. In the next section we will propose a more accurate model that takes meta-stability into account. In order to do so, we have to consider probabilistic Petri nets [49]. In such nets, each place is associated with a random variable which is characterized by a probability density function. This variable represents the random maturing time of the token, relative to the time when the token arrives in the place.

## A.2 Model of the Seitz' arbiter

In this section we introduce our probabilistic approach to the timing analysis of asynchronous circuits by working out a case example: the Seitz' arbiter.

Throughout this appendix we will make the following assumptions: (i) the circuit responds with a fixed delay if the separation between the requests is greater than $T_w$; (ii) if the requests arrive within $T_w$ of each other, the probability that a grant is generated after delay $\tau_m$ is given by Eq. A.1.1; (iii) strictly speaking $K$ depends on the time of arrival of the requests but in this appendix we assume that $K$ is invariant.

We propose to model the Seitz' arbiter with the Petri net shown in Figure A.2.1. This Petri net models only the grant phase of the arbiter in which only one grant is generated to a given request or requests. To understand the behavior of the Petri net shown in Figure A.2.1, consider first the case in which a request arrives and the other request is not issued during the window $T_w$.

Due to symmetry, it suffices to consider only request $r_1$. When transition $r_1$ fires after a token matures at its input place, it puts tokens into places $p_1$ and $p_2$. Unlabeled places such as $p_1$ make tokens visible immediately (i.e., the pdf $f_x(x)$ of the corresponding associated random variable $x$ is the Dirac's impulse function $\delta(x)$). Place $p_2$ is labeled with random variable $\tau_w$ with pdf shown in Figure A.3.1. Thus transition *proceed* will fire after $T_w$, and transition $g_1$ will fire after $D_1$. The total delay from the occurrence of $r_1$ to the issuance of the respective grant is $T_w + D_1$.

However if request $r_2$ fires within window $T_w$ after $r_1$ has fired, then transition *meta* will fire and either $g_1$ or $g_2$ (as selected by the free choice place $p_4$) will be generated after a delay $\tau_m$. Random variable $\tau_m$ obeys an exponential pdf as given by Eq. A.1.1.

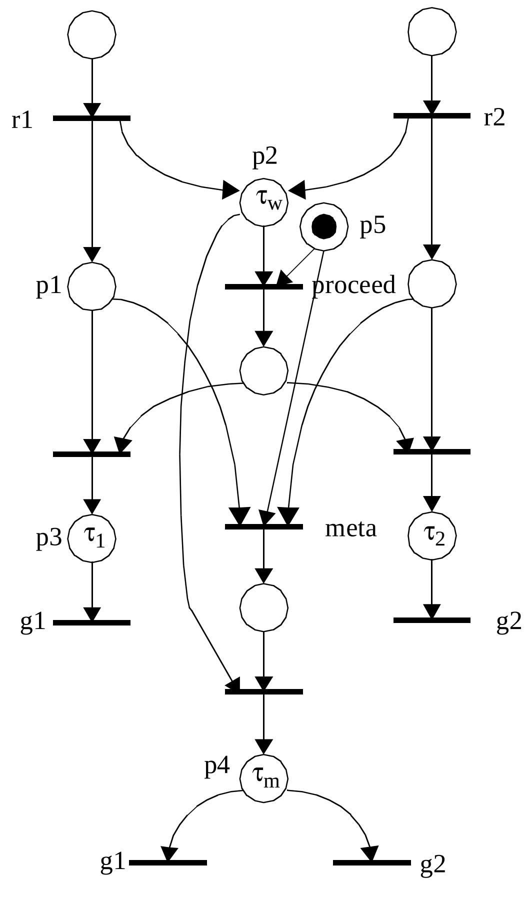


Figure A.2.1 Modeling metastability.

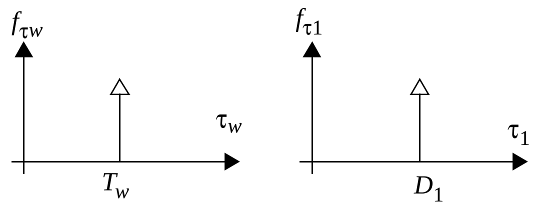


Figure A.2.2 Probability distributions of the random variables associated with labeled places of the Petri net shown in Figure A.2.1.

## A.3 Analysis

In this section we discuss how to analyze the Petri net shown in Figure A.2.1. We assume that the pdf's of the time of occurrence for requests $r_1$ and $r_2$, $\tau_{r1}$ and $\tau_{r2}$, are known and given by $f_{\tau r1}(\tau_{r1})$ and $f_{\tau r2}(\tau_{r2})$ (refer to Figure A.3.2). (In [49] we show how to find the pdf of a given transition for a sub-class of probabilistic timed Petri nets.) Our goal is to determine the probabilistic profile (i.e., pdf) of the grant transitions $g_1$ and $g_2$.

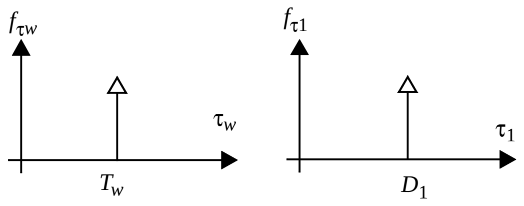


Figure A.3.1 Probability distributions of the random variables associated with labeled places of the Petri net shown in Figure A.2.1.

In this section we discuss how to analyze the Petri net shown in Figure A.2.1. We assume that the pdf's of the time of occurrence for requests $r_1$ and $r_2$ are known (refer to Figure A.3.2). (In [49] we show how to find the pdf of a given transition for a sub-class of probabilistic timed Petri nets.) Our goal is to determine the probabilistic profile (i.e., pdf) of the grant transitions $g_1$ and $g_2$.

From the previous analysis it is clear that the firing of transitions *meta* and *proceed* are mutually exclusive (i.e. there is a single token in place $p_5$).

Let us find the time of occurrence of transition $g_1$. First we introduce some basic concepts from [105]. Let $x$ be a random variable with probability density function (pdf) $f_x(x)$. The probability that variable $x$ takes a value in range $[x_1,x_2)$ is given by:

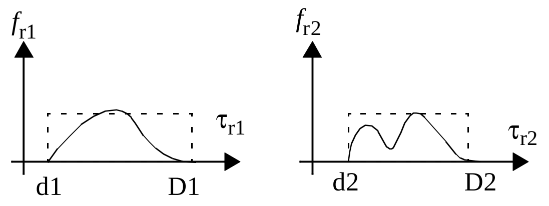


Figure A.3.2 Probability of the time occurrence of requests $r_1$ and $r_2$.

$$\text{Prob}\{x_1 \le x < x_2\} = F_x(x_2) - F_x(x_1) \qquad \text{(Eq. A.3.1)}$$

where $F_x(x)$ is the accumulative distribution function of random variable x, related to $f_x(x)$ by the following equation:

$$F_x(x) = \int_{-\infty}^{x} f_x(t)\, dt \qquad \text{(Eq. A.3.2)}$$

Using Eqs. A.3.1 and A.3.2, it can be shown that:

$$\text{Prob}\{x_0 \le x < x_0 + dx\} = f_x(x)\, dx \qquad \text{(Eq. A.3.3)}$$

The random variable $\tau_m$ associated with place $p_4$ represents a metastable state and thus it is described by the exponential pdf $f_{\tau m}(\tau_m)$ given by Eq. A.1.1. The probability density functions $f_{\tau ri}(\tau_{ri})$ describe the firing of transitions $r_i$ at time $\tau_{ri}$, for $i = 1, 2$.

From the discussion in Section A.2, the probability that transition *meta* will fire at time $\alpha$ (blocking the firing of *proceed*) is:

$$\begin{gathered}\text{Prob}\{\alpha \le \tau_{meta} < \alpha + d\alpha\} = \\ \text{Prob}\{\alpha \le \tau_{r1} < \alpha + d\alpha\} \cdot \text{Prob}\{\alpha - T_w \le \tau_{r2} < \alpha\} + \\ \text{Prob}\{\alpha - T_w \le \tau_{r1} < \alpha\} \cdot \text{Prob}\{\alpha \le \tau_{r2} < \alpha + d\alpha\} + \\ \text{Prob}\{\alpha \le \tau_{r1} < \alpha + d\alpha\} \cdot \text{Prob}\{\alpha \le \tau_{r2} < \alpha + d\alpha\} \qquad \text{(Eq. A.3.4)}\end{gathered}$$

Similarly the probability that transition *proceed* fires at time $\alpha$ given that transition $r_1$ has occurred is given by:

$$\mathrm{Prob}\{\alpha \le \tau_{proceed\,|\,r1} < \alpha + d\alpha\} =$$
$$\mathrm{Prob}\{\alpha - T_w \le \tau_{r1} < \alpha - T_w + d\alpha\} \cdot [1 - \mathrm{Prob}\{\tau_{r2} \le \alpha\} \qquad \text{(Eq. A.3.5)}$$

Thus the pdf's of the occurrence time for transitions meta and proceed are given by:

$$f_{meta}(\alpha) =$$
$$[F_{\tau r2}(\alpha) - F_{\tau r2}(\alpha - T_w)] \cdot f_{\tau r1}(\alpha) + [F_{\tau r1}(\alpha) - F_{\tau r1}(\alpha - T_w)] \cdot f_{\tau r2}(\alpha) \qquad \text{(Eq. A.3.6)}$$

$$f_{proceed\,|\,r1}(\alpha) = [1 - F_{\tau r2}(\alpha)] \cdot f_{\tau r1}(\alpha - T_w) \qquad \text{(Eq. A.3.7)}$$

Let us assume for the sake of illustration that both $f_{\tau r1}$ and $f_{\tau r2}$ are uniform in the interval $[0,D]$ and that $D = 20\,T_w$. Substituting the parameters of the pdf's into Eqs. A.3.6 and A.3.7, one can obtain the following expressions:

$$f_{proceed\,|\,r1}(\alpha) = \begin{cases} \dfrac{D-\alpha}{D^2}, & \text{if } T_w \le \alpha < D \\ 0, & \text{otherwise} \end{cases}$$

$$f_{meta}(\alpha) = \begin{cases} \dfrac{2\alpha}{D^2}, & \text{if } 0 \le \alpha < T_w \\ \dfrac{2T_w}{D^2}, & \text{if } T_w < \alpha \le D \\ 0, & \text{otherwise} \end{cases}$$

If *proceed* has occurred due to $r_1$, the grant $g_1$ will be issued at time $\tau_{proceed} + \tau_1$, where $\tau_1$ is the random variable associated with place $p_3$. To compute the firing time of

$g_1$ we shall use the fact that the pdf of random variable $x = y + z$ is $f_x = f_y * f_z$ if $y$ and $z$ are independent random variables, where the $*$ operator denotes convolution [105].

If meta has occurred, place $p_4$ selects either $g_1$ or $g_2$, with a 50% chance. (Note: in a first approximation, a non-deterministic choice event can be considered a randomly selected event; an extension of the model could assign a probability to each of the choices of a free choice place). The pdf of random variable $\tau_m$ associated with $p_4$ is $f_{\tau m} = C\, e^{-K\tau m}$, for $\tau_m \geq 0$, and $g_1$, if selected, will fire at time $\tau_{meta} + \tau_m$.

Thus the probability that $g_1$ will be issued at time $\alpha$ is given by:

$$f_{g1}(\alpha) = f_{proceed\,|\,r1}(\alpha) * f_{\tau 1}(\alpha) + 0.5\, f_{meta}(\alpha) * f_{\tau m}(\alpha) \qquad \text{(Eq. A.3.8)}$$

The first term corresponds to the generation of $g_1$ via *proceed* (which is $f_{proceed\,|\,r1}(\alpha - D_1)$, a transport delay) and the second term corresponds to the generation of $g_1$ via *meta*. Figure A.3.3 shows the pdf of the occurrence of grant $g_1$ for the uniform case. One can observe a “triangular” shape that corresponds to $g_1$ generated via *proceed*, and a tail that corresponds to $g_1$ generated via *meta*. The area under the curve is 0.5 which represents the 50% probability of occurrence of $g_1$ ($g_1$ and $g_2$ being equally likely to occur). The probability that $g_1$ is generated after a delay $> 15$ diminishes exponentially. For example the probability that $g_1$ will be generated after 15time units is approximately 1.9%. Moreover, the probability that $g_1$ will be generated after 40 time units is under 0.15%.

## A.4 Summary

We have introduced a probabilistic model capable of representing with more accuracy the complex behavior of the Seitz’ arbiter, including metastability. The advantage of our approach is twofold: first our analysis procedure relies upon a formal model for

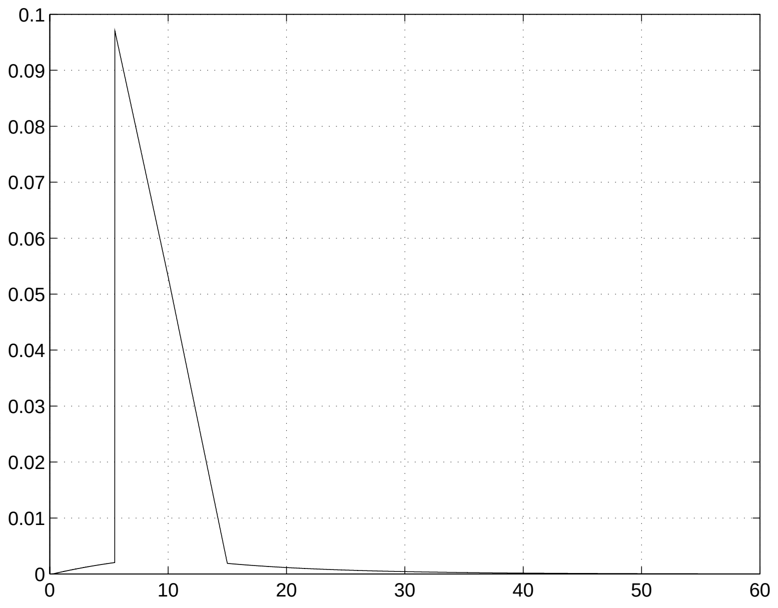


Figure A.3.3   Probability density function of the occurrence time of $g_1$ for $D$=10, $D_1$=5, and $K$=0.1.

circuit specification (a probabilistic timed Petri net), and secondly our model is an extension of signal transition graphs (STG's) [29, 79] which are widely used to describe asynchronous circuits.

We believe that a probabilistic analysis is essential in the qualitative study of the impact of metastable behavior in the timing performance of asynchronous circuits which can exhibit this phenomenon.