Université de Montréal

Timing Verification of Interface Specifications and Controllers

par
Fen Jin

Départment d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en informatique

août, 2000

QA
76
D54
2001
v. 002

Université de Montréal
Facuté des études supérieures

Cette thèse intitulée

Timing Verification of Interface Specifications and Controllers

Présentée par
Fen Jin

a été évaluée par un jury composé des personnes suivantes:

Dr. El Mostapha Aboulhamid	président-rapporteur
Dr. Eduard Cerny	directeur de recherche
Dr. Yvon Savaria	membre du jury
Dr. Marco A. Escalante	examinateur externe

Thèse acceptée le:

text on back

Résumé

Vérification des spécifications temp des interfaces et des contrôleurs

De nos jours, les systèmes micro-électroniques se composent généralement de composants tels des (ASICS), de circuits intégrés dédiés et d'autres IPs (Intellectual properties). Ces composants sont la plupart du temps conçus par différentes équipes travaillant dans différentes organisations. Dans ces systèmes, les composants communiquent entre eux via des interfaces. Il est donc très important de s'assurer de la compatibilité de ces interfaces, et aussi de s'assurer que les implémentations de ces interfaces respectent les spécifications de ces dernières.

Ce présent travail traite de la vérification des spécifications temporelles des interface et de la vérification de implémentations.

La vérification des spécifications temporelles des interface inclut la notion de compatibilité et aussi d'autres propriétés temporelles qu'il faut garantir. Dans ce travail, nous résolvons les problèmes de vérification des spécifications temporelles en utilisant le temps maximum de séparation entres les événements dans un graphe de contraintes tire des spécifications. Les interfaces en question sont spécifiées comme des boucles exprimées avec le langage HAAD (Loop over a leaf Hierarchical Annotated Action Diagram). Nous appliquons notre solution pour la vérification de certaines propriété temporelle qu il faut garantir dans la modélisation de la modélisation de la spécification d'une opération de lecture répétée d'un microprocesseur d'une mémoire.

La vérification des contrôleurs d'interfaces, avec leur spécification consiste a que l'implémentation produira les bons événements aux bons moments, comme décrits dans la spécification, en supposant que les événements aux entrées du contrôleur sont émis temps ainsi spécifiés dans la spécification. Dans ce travail, nous présentons une méthode pour vérifier si l'implémentation d'une machine à états pseudo-synchrone (entrée échantillonnée) d'un contrôleur en temps réel satisfait les spécifications de son diagramme temporel.

On applique notre méthode a un contrôleur de bus qui provient d'un design industriel et nous vérifions deux cycles d'ECRITURE avec la modélisation de la spécifi-

Abstract

Timing Verification of Interface Specifications and Controllers

Microelectronic systems are normally composed of components such as Application Specific Integrated Circuits (ASICs), custom integrated circuits and other intellectual properties. These components are generally designed by different teams from different organizations. In the systems, components are connected and communicated with each other through interfaces. To make the systems work, it is very important to verify that the interface specifications of components are compatible with each other and the implementations of interfaces are correct with respect to their specifications.

This work deals with timing verification of interface specifications and the verification of the interface controller implementations against their specifications.

The timing verification of interface specifications includes compatibility and the verification of the safety timing property. In this work, we solve the timing verification problems using maximum time separation between events in constraint graphs transformed from the specifications. The interested interfaces are specified as loops over a leaf Hierarchical Annotated Action Diagram (HAAD) language. We apply our solution technique in verifying several safety timing properties of a specification modeling a repeated READ operation of a microprocessor from a memory.

The verification of interface controllers against their specification consists in making sure that the implementation will produce correct events on time as given in the specification, under the assumption that the inputs events fed to the controller are on time as also stated in the specification. In this work, we present a method for verifying whether a pseudo-synchronous (sampled input) finite-state machine implementation of a real-time controller satisfies its timing diagram specification.

We apply our method to a bus controller from an industrial design and check against its timing diagram specification modeling two consecutive asynchronous WRITE cycles.

All the algorithms and verification methods in this work are implemented in a Constraint Logic Programming environment based on Relational Interval Arithmetic.

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Eduard Cerny, for giving me the chance to work in the hardware verification area and for spending time on me in the last five and half years. Without his encouragement and guidance, I doubt that I can finally finish this work.

Second, I want to thank the examiners of the dissertation, especially Dr. Marco A. Escalante, for their insight and detailed comments.

I also want to thank my family members, my parents for taking care of my girl in the first two years after I came to Canada, my husband for his full support and sacrifice, and my lovely daughter Kathy who accompanied me in the university drawing her dreams on many weekends while I was working.

Last, I thank my friends. Many thanks to JiaHao Wu and his family, with whom I spent all my christmas evenings in the last six years, they made me feel home and let me see the hope of future. Many friends in Lab Lasso and CRT also deserve my thanks, they made my life in UdM more enjoyable and memorable.

Glossary

A	set of assume constraints
B	a block partition
C	set of commit constraints
E	set of events
E_{IN}	set of input events
E_{OUT}	set of output events
$G = <E, R>$	(cyclic) constraint graph
$G^f = <E^f, R^f>$	acyclic finite constraint graph
$G^\alpha = <E^\alpha, R^\alpha>$	α unfolded constraint graph of G
$G^\infty = <E^\infty, R^\infty>$	infinite unfolded constraint graph of G
$G_m = <E_m, R_m>$	correspondent graph of G containing both bounds of linear constraints and lower bounds of latest constraints
$G_M = <E_M, R_M>$	correspondent graph of G containing both bounds of linear constraints and upper bounds of latest constraints
M	controller machine
R	set of rule edges
TD	timing diagram specification
TDTA	TD transition automaton
U_α	set of events in unfolding α of a constraint graph G
W	set of waveforms
WTA	waveform transition automaton
WV	vector of waveform values of a TD

WV_{IN}	vector of values of waveforms for input ports of a TD
WV_{OUT}	vector of values of waveforms for output ports of a TD
e_v, v	an event
$(e_u, e_v, [b_{uvmin}, B_{uvmax}])$	timing constraint from e_u to e_v involved in linear combination
$(e_u, e_v, [d_{uvmin}, D_{uvmax}])$	timing constraint from e_u to e_v involved in latest combination
$(e_u, e_v, [T_{uvmin}, T_{uvmax}])$	timing constraint from e_u to e_v
m(v), sd(s,v)	shortest distance from s to v
msources(v)	set of source events of constraints to v represented in G_m.
preds(v)	set of source events of constraints to v involved in a max combination
trigs(B)	triggers of B
$\delta(s, e)$	minimum time separation from s to e
Δ, $\Delta(s, e, \beta)$	maximum time separation from s to e with index difference β in a(cyclic) constraint graph
Δ^{α}	time separation from s_{α} to $e_{\alpha+\beta}$ in an acyclic graph
$\Delta(s, e)$	time separation from s to e in an acyclic graph
$\Re$	set of real
$\tau(v)$	occurrence time of event v

Chapter 1

Introduction

This work deals with interface timing verification problems in microelectronic systems. Microelectronic systems are normally implemented as a hierarchy of functional blocks. The blocks communicate with each other through interfaces. The functional blocks can be Application Specific Integrated Circuits (ASICs) or custom Integrated Circuits (ICs). With design reuse methodology, the ASICs and custom ICs can be intellectual properties (IPs). As a result, the systems are composed of blocks which are possibly designed and verified by different design teams from different organizations. This makes the system-level verification task to assure correctness of interfaces between blocks one of the most challenging problems in microelectronic system design.

Interfaces in a microelectronic system are places where different blocks meet and communicate with each other. An example of interfaces is the place where a microprocessor interacts with a memory. When blocks are connected to form a system, each block is connected through interfaces to the environment composed of the other blocks in the system. For a block to work properly in a system, the environment must respect assumptions made on the interface of the block. This verification can be achieved by a compatibility checking of the environment specification with respect to the specification of the block. If the environment is verified as incompatible with the interface specification of that block, then an interface transducer has to be designed to bridge the differences between the block and its environment. In this work, we are concerned with timing property (compatibility and other safety timing property) verification of interface specifications and the verification of implementations of interface controllers against their specifications.

We use mathematical methods to check the complete behaviour of interface specifications and controller implementations. The work is thus in the domain of formal verification. We use a subset of Hierachical Annotated Action Diagrams (HAAD) [17] language, which was developed specifically for interface specification and verification in the laboratoire LASSO, University of Montreal, as the specification language of interfaces. We use finite state machine (FSM) derived from models in hardware description languages such as Verilog [68] or VHDL [75] as the implementation of the interface controllers.

In the first part of the work, we concentrate on algorithms for computing the maximum time separations between events in constraint graphs describing timing relationship between events in interface specifications. The algorithms are directly applicable for compatibility verification of interface specifications. They can also be applied to check other safety timing properties of interface specifications. In the second part of the work, we propose a methodology to verify the correctness of an interface controller implementation with respect to its specification. In both cases, we implemented the solution techniques in a constraint logic programming (CLP) environment based on Relational Interval Arithmetic (RIA) and domain narrowing [66].

In this introduction, we first describe in more detail the two problems of interest to us, namely timing verification of interface specifications and verification of real-time controllers against their specifications, in Sections 1.1 and 1.2, respectively. We then summarize the contributions of our work in Section 1.3, and give a plan of the thesis in Section 1.4.

1.1 Timing Verification of Interface Specifications

The work on timing verification of interface specifications started about a decade ago. Several members in the laboratoire LASSO have worked on this subject ([17][20][36][37][50] etc.). Their work is mainly on semantics and timing verification of HAAD specifications. Their timing verification method is applicable to interfaces specified by leaf HAAD specifications, a subset of the HAAD language[17]. There, the timing verification problems are solved by finding the maximum time separation of events in constraint graphs extracted from the specifications. The original motivation of the first part of our work is to extend the solution techniques of computing the maximum time separation to specifications modeling a more general cyclic interface behaviour (Loop over a leaf HAAD specification [17]). We show that the maximum time separation solution techniques developed in this thesis can be used to solve compatibility and other safety timing verification problems. We apply our solution technique to verify several safety timing properties of an interface specification modeling a repeated microprocessor READ access to a memory.

In this section, we first survey formal specification models related to system and timing property specifications. Then we give the reasons for adopting the HAAD language [17] as the formal specification language. After that, we summarize the work on the timing verification of interface specifications.

1.1.1 **Specification Languages**

An interface specification describes the communication protocol on an interface. Traditionally, interface specifications in general specifications (GS) of a device are given in the form of timing diagrams. There are normally additional textual annotations written in natural languages such as English, and the timing constraints usually appear in tables describing timing relationships between signal transitions on ports of the interfaces. The annotations are necessary for designers to understand the specification, but they make it difficult to model the specification in a formal language for analysis, because there is no formal semantics for such descriptions. In the following, we first summarize related formal specification languages, and then illustrate why these models are not powerful enough to specify interface behaviours directly.

In the literature, there are many formal specification models such as Timed Temporal Logics, Real-Time Logics, Timed Process Algebras, Timed Automata, Time Trace Structures, Timed Petri Nets, etc., which can model real-time systems and properties. These timed specification models have been used to specify and verify timing properties in real time systems, software engineering, and VLSI CAD communities. In the following, we briefly describe these timed models.

Temporal Logics are languages based on propositional or first order logic with special operators for reasoning about qualitative time (e.g., $\Diamond$ for "eventually" and $\Box$ for "always"). Different mechanisms have been used to include quantitative time information in temporal logics. The popular ones include the use of bounded-time temporal operators (e.g., $\Diamond_{[2,4]}$ meaning "eventually within 2 to 4 time units."), introduction of an explicit clock variable, and temporal quantification of time variables. Examples that use bounded temporal operators are Real-Time Computational Tree Logic (RTCTL) [24], Timed Computational Tree Logic (TCTL) [2], Metric Temporal Logic (MTL) [54]. An example of using explicit clock variables is Real-Time Temporal Logic (RTTL) [67]. An example of using temporal quantification of time variables is Timed Propositional Temporal Logic (TPTL) [3].

Real Time Logics (RTL) was developed by Jahanian and Mok to specify, verify, and synthesize real time systems [46]. RTL is a restricted form of First Order Logic. A special occurrence function is used to keep track of the n-th occurrence of an event, which leads to a fairly expressive logic. However, theories in RTL are undecidable. In [45], decision algorithms were developed to verify safety timing properties for systems expressed in a subset of RTL. Later in [63], Millet extended the work in [45] for verifying safety timing properties expressed in an extended subset of RTL. To specify more complex real time systems, a hierarchical graphical specification language Modechart was developed [46] with RTL as its semantics. Modes and transitions are used as the basic structure representing the control state and the control flow of a specification. Since the underlying semantics of Modechart is RTL, general theories in Modechart are undecidable, and whether a particular Modechart satisfies a particular RTL formula is also undecidable. A Modechart toolset [46] was developed for specifying and verifying safety timing properties for real time systems. The tool verifies whether a timing property expressed in RTL satisfies a real time system described in Modechart. Several subsets of RTL in which the verification problem is decidable were described in [76]. In the Modechart toolset, a model checking technique is used to check the truth of a safety timing property expressed in RTL with respect to a computation graph which is a finite representation of all the behaviours of the Modechart specification. Like all data structures used in model checking, the computation graph suffers from the state explosion problem.

Process algebra such as CCS, CSP and ACP were developed for describing and analysing communicating concurrently executing systems. A process algebra consists of a language with precisely defined semantics and a notion of equivalence. To verify a system using a process algebra, one writes a specification as an abstract process and an implementation as a detailed process. To prove correctness, the two processes are shown to be equivalent or a preorder between the two processes is shown. Recently, several timed process algebra were developed by adding the notion of time and including a set of timed operators to the untimed process algebras. Extensions of CCS include the Algebra of Communicating Shared Resources, ACSR [12][56]; Temporal CCS [64]; Timed CCS [82]; Algebra of Timed Processes ATP [65]; and Algebra of Communication Timing Charts (ACTC) [8] which was proposed especially for the specification and verification of interface behaviours. Extensions of CSP include Timed CSP [73] and Timed LOTOS [10]. The latter is based on CSP and was applied to several industrial projects. ACP was extended to Timed ACP in [6], [34].

Alur and Dill [1] proposed an extension of Büchi and Müller ω-automata to include metric dense time. Transitions are labelled by time constraints on 'clock variables', and while executing a transition, a clock can be reset to zero. A trace is accepted by a timed automaton if its transitions are performed at times that satisfy all clock constraints. A tool, KRONOS [60], was developed to assist the user with validating complex real-time systems. The tool checks whether a real-time system modelled by a timed automaton satisfies a timing property specified by a TCTL formula. KRONOS was used to verify a variety of time-dependent protocols and MOS circuits [60].

Dill [27] used trace theory to specify and verify speed-independent circuits. In trace theory, circuit behaviour is described by sets of traces, where a trace is a sequence of transitions. Dill's approach does not include metric integrate time in the model. Burch [14] extended Dill's work by incorporating discrete time and various delay models into the trace theory analysis, aiming at hardware design verification. Automata-based techniques were used to do the verification. The run time is exponential in the number of components and heavily dependent on the size of the delay values in the model.

Petri nets are a graph model that was also used extensively to model concurrent systems. They are bi-partite graphs composed of two types of nodes called places and transitions. Many different models of time were defined for Petri nets, mostly by imposing timing constraints on the enabling and firing rules of untimed Petri nets. These constraints can be represented by constants or functions. The former includes Timed Petri nets which treat a timing constraint as a single delay[40], [70], [71], Time Petri nets which treat a timing constraint as a time interval defined by a lower and an upper bound [59], [61], [72], and Timing Constraint Petri nets which associate each place with a time interval and each transition with a duration [77]. The latter includes Stochastic Petri nets [58] which treat a timing constraint as a probability function of the transition firing rate, and ER nets [32] which treat a timing constraint as a function of coloured tokens in input places. Time Petri nets were used for analysing the recoverability of communication protocols and the safety of real time systems. Timed and Stochastic Petri nets have also been used for performance evaluation. Timing Constraint Petri nets which are more like the graphical representation of the subset of RTL described in [45] were used to check schedulability of a specification, where a specification is defined as schedulable if every event in the system can be assigned an occurrence time satisfying all the imposed timing constraints [77].

In [29][30][31], Escalante et al developed an "interface specification" to model behavior of a component in a system. An interface specification consists of two parts: a timed signal transition graph (a probablistic timed petri net), and a restriction on the component's environment (a set of constraint rules). Then the system behaviour is described by "merging" interface specifications of all the components in the system. They proposed a verification technique to check time consistency for the so-called "closed systems", a type of systems that are self-contained and do not interact with any environment. In their approach, all the delays are represented by random variables.

Unfortunately, it is far from trivial to specify interface timing directly by the above formal timed specification models. The reasons are twofold. On the one hand, it is difficult to extract a formal interface specification from the data sheet in the GS. On the other hand, timing constraints describing the timing relationship on interfaces in the GS such as latest, earliest and linear constraint combinations, and delay correlation between constraints which are necessary to reason about interface timing behaviour are too complex when expressed by the existing specification models.

Efforts have been made in two directions to address these problems. One is to formalize timing diagrams, and the other one is to extend formal specification models to handle complicated timing constraints. The combination of the efforts in these two directions lead to a formal language called Hierachical Annotated Action Diagram (HAAD) [17] which was inspired by Timing Diagrams and Process Algebras.

The HAAD language was developed to specify systems with complex timing and functional behaviour on interfaces. In the HAAD language, timing information on interfaces is shown through timing constraints over actions on interface ports and internal signals, while the functional behaviour on interfaces is described by annotating actions using variables, predicates and procedures. In this way, the timing information is presented orthogonally to the functional behaviour. Formal semantics has been defined for the HAAD language without annotation. This is sufficient as a specification language for the purpose of formal timing verification. In this work, we are only interested in the timing behaviour on interfaces, we thus adopt the HAAD specification without annotation as the interface specification language (from now on, we will use "HAAD" instead of "HAAD without annotation" for notational simplification).

1.1.2 **Timing Properties on Interfaces**

To the best of our knowledge, Brozozowski et al [13] are the first ones to address the problems of checking timing properties on interface specifications. In [13] and [35], they pointed out that constraints that require the device environment to provide the proper inputs (denoted as assume constraints) and those that specify what the environment can expect from the device if the input conditions are met (denoted as commit constraints) should be distinguished. They proposed two timing properties that should be verified to guarantee the correct interaction between connected blocks in a system. The two properties are consistency and satisfiablility. A timing diagram specification is consistent if occurrence times can be assigned to all actions without violating any constraints. Interface satisfiablility of two timing diagram specifications is defined to hold if the time separation between each pair of actions implied by a commit constraint of one timing diagram is tighter than the separation between the same pair of actions implied by the assume constraints of the other timing diagram.

It was demonstrated later by Cerny et al [17][20][50] that the notions of consistency and satisfiablility of timing diagrams in [13][35] are not sufficient for verifying that two or more blocks will interact correctly when built according to their local specifications. They define formal operational semantics of the leaf TDs based on a block machine which is derived from interface specifications in the form of leaf TDs. They give sufficient conditions in [50] for a specification to be realizable (causal) and further prove that a block machine derived from a realizable specification has the same timed traces as those from the specification itself. They define the compatibility of an interface specification as follows:

All combinations of the block machines derived from realizable specification are free of dead lock.

They guarantee that a causal implementation of compatible specifications can interact correctly in a system. They further proposed compatibility conditions for leaf timing diagrams of a HAAD specification with linear timing constraints [17][50]. It is illustrated there that the conditions can be verified by finding the maximum time separation between events in constraint graphs transformed from the interface specifications. The work was extended by Girodias et al. [36][37] to verify the compatibility of leaf timing diagrams of HAAD specifications containing linear, max (latest) and min (earliest) constraints. The solution technique can also deal with delay correlation between the timing constraints.

In a HAAD specification, leaf timing diagrams model elementary operations on the interface (e.g., READ, WRITE of a processor to a memory). The complete interface behaviour is modeled by a hierarchy of leaf timing diagrams using composition operators. The solution techniques in the previous work ([17][20][50]) can only verify behaviour modeling elementary operations on interfaces.

We are interested in verifying timing properties of interface specifications which can describe more general interface behaviours than those described in [36][37] (e.g., repeated READ or WRITE of a microprocessor to a memory). More specifically, we are interested in verifying interface specifications modeled by the HAAD language containing hierarchical operators. In the first part of this work, we consider “Loop” operator over a leaf action diagram (a timing diagram), a type of specification which can describe an infinite repetitive behaviour without a choice on an interface.

1.2 Verification of Interface Controllers Against Their Specifications

In this section, we introduce the problem of verifying an implementation of an interface controller against its specification and then discuss related literature.

In the design of Systems on a Chip (SoC) from predesigned building blocks, it is important to assure that these blocks can communicate correctly. This means that much effort is spent on designing and verifying bus controllers and other communication control logic. Even if the processor bus protocol is asynchronous, the controllers of such a bus in the connecting devices are often designed as synchronous finite state machine (FSM), operating on synchronized input signals from the bus. Timing simulation is the usual method for verifying that the controller can operate in the full range of the bus protocol. This is generally far from satisfactory due to the large number of different timing situations that could exist on the input signals. To perform this verification exhaustively, yet without the full explicit enumeration of all situations, we propose a method based on Constraint Logic Programming for verifying whether a pseudo-synchronous (sampled input/output) finite-state machine (FSM) implementation of a real-time controller satisfies its leaf TD specification.

The problem can be defined as follows: given an implementation of an interface controller in the form of an FSM, a clock frequency and a specification in the form of a leaf TD, we check whether the FSM implementation always produces the correct outputs and within the timing constraints stated in the specification, provided that the inputs meet the assumptions as also stated in the specification.

Although there are many published results regarding static timing verification of synchronous sequential circuits, most of them are concerned with set-up and hold time checks on sequential elements (flip-flops, latches, memories etc.). In our work, we are interested in the timing information as determined by the functionality of the circuit, i.e., a mixture of timing and behavioural verification over a number of clock cycles. This kind of verification could potentially be carried out using models based on Timed Automata (TA) [2] or timed Petri Nets [40], however, the TA models and the accompanying verification techniques based on reachability analysis of the derived region graphs are unnecessarily difficult and complex in this practical context of verifying realistic RTL designs of hardware interface controllers.

Clocksin [26] at University of Cambridge used logic programming to carry out simulation of synchronous sequential circuits. He used rule based unit clauses which resemble standard truth tables to model relations in the modules of the circuit to be simulated. As we shall show, such an approach leads to exponential time explosion, because most of the possible FSM executions are enumerated. Instead, we use constraints rather than rule based clauses to model the execution of sequential circuits in real time. This consists of unrolling the FSM over a sufficient number of clock cycles to cover the range of time implied by the TD specification, expressing the unrolled instances of the FSM as a series of constraints. We then link these constraints to the timing constraints from the TD using a set of automata that convert event occurrences on signals in the TD to signal levels required/produced by the controller FSM. In addition, since we inherently deal with uncertainty intervals as to the time of occurrence of events, we can carry out this verification under variations in the clock frequency and delay correlation, similarly as in [17][36].

The representation of the sets of states, the output traces of the TD automata, and the implementation FSM is compact (linear with time) in terms of the number of constraints, while the number of traces thus characterized may grow exponentially with time. Although we verify only finite behaviours as described by the leaf timing diagram, most realistic bus protocols usually return to an initial state after executing a particular operation cycle. We can thus verify that at the end of the finite trace the final state of the automaton and the maximum separations of the last events on each port correspond to the initial state. In a more general approach, we plan in future work to estimate an upper bound on the number of unfoldings required of the combined TD plus FSM system before all the maximum event separations start to repeat. The approach can be compared to model checking on finite computations derived from a Kripke structure using a satisfiablility procedure [7].

1.3 Contributions of the Work

The contributions of this dissertation can be summarised in three main points.

a) We developed a method for performing compatibility and other safety timing property verification of interfaces specified by a HAAD language in the form of Loop over leaf TDs. We solve the verification problems by providing algorithms for computing the maximum time separation of events in infinite constraint graphs derived from the specifications.

b) We propose a method for verifying the correctness of an interface implementation in the form of an FSM against its leaf HAAD specification.

c) Applications: We verified causality and several safety timing properties on a HAAD specification modeling a repeated microprocessor READ operation (inspired by MC68360) from a memory; we also verified the FSM implementation of a bus controller with respect to its leaf HAAD specification.

Regarding the algorithms of finding the maximum timing separation between events in constraint graphs transformed from HAAD specifications in the form of Loop over leaf TDs, we show the way to transform the HAAD specification with linear-plus-latest timing constraints to constraint graphs. We derive well-formedness conditions of these constraint graphs, and define the maximum time separation problem in the constraint graphs. We propose algorithms for computing the maximum time separations in constraint graphs with linear-only, linear-plus-latest, and restricted linear-plus-latest constraints which relates to realizable (causal) specifications.

In the case of linear-only constraint systems, the proposed algorithm is polynomial in the number of events in the constraint graph which is also the number of events in the leaf timing diagram of the HAAD specification.

In the case of linear-plus-latest constraints, we show that the time separations will become periodic functions of event indices. The number of unfoldings needed for the time separations to become periodic depends on the delay values in the specification. We give a sufficient condition to determine when this happens and propose an algorithm for computing the maximum time separation. The algorithm consists of a step by step unfolding of the constraint graph and computing the time separation in the unfolded constraint graphs. The computation continues until the time separation becomes a periodic function and no further unfolding is need. Since the number of unfoldings required for the time separation to become periodic depends on the delay values in the specification, the algorithm is not practical for real complex problems. We then restrict the linear-plus-latest constraint graphs to causal ones which correspond to realizable specifications. In such causal constraint graphs, we are able to give an upper bound on the number of unfoldings that one has to consider in order to compute the maximum time separation. Unlike the case of linear-only systems where the upper bound is dependent only on the number of events in the constraint graph, the upper bound here is also dependent on the delay values in the specification. But the value of the upper bound can be determined after a time separation calculation in a finite process graph containing events and constraints of a number of unfoldings. For most practical application, this number is fairly small. We then obtain an exact and efficient algorithm for computing the maximum time separation in causal constraint graphs with linear and latest constraints.

Regarding the problem of verifying the correctness of real time interface controllers against their specifications, the contributions can be stated in more detail as follows:

- Modeling a timing diagram specification as communicating "TD" automata that accept event traces respecting the timing constraints;
- The representation of the finite unfolding of the implementation FSM, the finite execution of the TD automata and timing requirements in the form of constraints;
- Formulating the FSM versus TD timing verification problem as a consistency check of a series of constraint systems;
- Implementation of the method using Constraint Logic Programming within a Relational Interval Arithmetic environment;
- Acceleration of the convergence of the implemented algorithm by adding redundant constraints.

We applied the solution techniques to real designs. We verified the causality and two safety timing properties of an interface specification modeling repeated microprocessor READ operations (inspired by MC68360) from a memory. It is based on computing the maximum time separation of events in restricted constraint graphs containing linear-plus-latest constraints. We also verified that the FSM model of a real-time controller extracted from its RTL Verilog description satisfies its timing diagram specification.

Part of this work has been published in [18] [19] [49].

1.4 Organization of the Thesis

The thesis consists of two major parts. The first part, Chapter 2 through Chapter 6, describes algorithms for solving the maximum time separation problem in constraint graphs transformed from HAAD specifications, and their application in safety timing property verification of interface specifications. The second part, Chapter

7 and Chapter 8 describes a solution technique for verifying pseudo-synchronous interface controllers against their leaf HAAD specifications, and its application to a real industrial design.

Chapter 2 is an introduction to the first part of the work. We first describe HAAD specifications, and then demonstrate that for a block in a system to interact correctly, the interface specifications involved must satisfy the causality and compatibility conditions. We then show that the causality and compatibility conditions can be verified by the maximum time separations between events in constraint graphs transformed from HAAD specifications.

In Chapter 3, we define the constraint graphs extracted from HAAD specifications in the form of Loop over leaf timing diagrams. We then give well-formedness conditions of such graphs and formally define the problem of maximum time separation of events in such constraint graphs. We show through a real example that the maximum time separation can be used to verify other interesting safety timing properties on interfaces.

In Chapter 4, we develop and prove the correctness of an algorithm for computing the time separation of events in finite unfolded constraint graphs. The algorithm is a variation of the one proposed by McMillan and Dill [62]. The difference between the two algorithms is that ours finds the maximum time separation from one start event to all the events in the graph while the one in [62] finds the time separations between all pairs of events in the graph. We use extensively the variation algorithm to solve the maximum time separation problem in the infinite constraint graphs of interest in Chapter 5 and Chapter 6.

In Chapter 5, we discuss algorithms for finding the maximum time separation of events in constraint graphs extracted from HAAD specification in the form of Loop over leaf timing diagrams. In the case of linear-only systems, we propose and prove the correctness of an algorithm which is polynomial with the number of events in the leaf timing diagrams. In the case of linear-plus-latest systems, we show that the maximum time separation will become periodic functions of event occurrence index. We give a sufficient condition to check when this happens. The speed of convergence of the iteration algorithm depends on delay values in the specification, which makes the algorithm impractical for real complex problems.

In Chapter 6, we restrict the constraint graph in such a way that events in unfolded constraint graphs can be partitioned into a set of topologically ordered blocks that respect the causality conditions. The restricted graphs reflect realizable designs and can be used to model most realistic systems. We propose and prove a practical and exact algorithm of computing the maximum time separations between events in the restricted constraint graphs. We then simplify the algorithm of finding a time separation in a restricted finite graph with linear-plus-latest constraints. After that, we derive an algorithm of computing the maximum time separation in causal cyclic constraint graphs. Since the causality conditions of constraint graphs need the information of the maximum time separation between events, there is an interleaving between computing the maximum time separations and the causal conditions of the cyclic constraint graph. We give sufficient causal conditions based on the maximum time separation between events under the assumption that the system is causal. Finally, we apply the solution technique to verify causality and two safety timing properties of interface specifications modeling continuous READ of the microprocessor from the memory.

In Chapter 7, we describe the methodology for verifying finite state machine (FSM) implementation of a real-time controller against its timing diagram specification. We use constraints to model the execution of the implementation in real time. We solve the problem by formulating the verification problem as a consistency checking of a constraint system. The consistency check is implemented in a Constraint Logic Programming environment (CLP) based on relational interval arithmetic (RIA) [66].

In Chapter 8, we apply the method developed in Chapter 7 to an industrial design. We verify that the implementation of a bus controller meets the specification under all timing situations.

In Chapter 9, we conclude the work and give some future directions of research.

Chapter 2

Interface Specifications and Verification

This chapter serves as an introduction to the first part of the work (Chapter 2 through Chapter 6). It consists of 4 sections. In Section 2.1, we describe specifications of interfaces using Hierarchical Annotated Action Diagram (HAAD). In Section 2.2, we summarize the causality and compatibility conditions of interface specifications. In Section 2.3, we show that these conditions can be verified by finding the maximum time separations between pairs of events in constraint graphs derived from interface specifications. In Section 2.4, we give a survey of the work related to computing the maximum time separations.

2.1 HAAD Specifications of Interfaces

Timing Diagrams as given in the data sheets of GS have been used for decades by designers to describe timing relations between signal transitions on interface ports. They are easy to understand, but they are an informal language with no formal semantics to interpret them unambiguously.

In [11], Borriello proposed a formalized timing diagram. A formalized timing diagram is a hierarchy of segments which consist of a collection of events and timing constraints between the events. It supports the hierarchical operators: Parallel, Choice, Sequential Composition and Loop. It has been used for the synthesis of interface transducers.

However, since formalized timing diagrams do not distinguish the constraints that characterize the assumptions on the environment and those produced or guaranteed by the device, the formalized timing diagrams cannot specify the assumption and reaction relationship between signal transitions on interface ports.

In [53], a structured language, Hierarchical Annotated Action Diagrams (HAAD) were developed for describing the behaviour of digital systems as seen from their interfaces. In the HAAD language, the interface behaviour is captured as a hierarchy of timing diagrams. In the hierarchy, leaf timing diagrams model elementary behaviours (operations such as READ, WRITE), complex behaviours are formed hierarchically using composition operators such as Concatenation, delayed Choice, Concurrency, Loop, and Exception. In the following, we first introduce the Loop operator and then give a definition of leaf timing diagrams. We do not include the definitions of the other hierachical operators here, since they are not used later on in this thesis. The interested reader can refer to [17] for the formal definition of all the hierarchical operators and formal semantics of the HAAD language.

A Loop over a leaf timing diagram as shown in Figure 2.1 models the following interface behavior: the same behaviour described in the leaf TD repeats an infinite number of times.

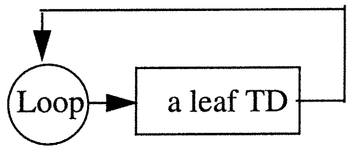


Figure 2.1 Hierarchical Loop over a leaf timing diagram

A leaf timing diagram TD is composed of a set of waveforms and a set of timing constraints. A waveform is a sequence of signal transitions (events)[1] between steady state signal values. Timing constraints relate waveform events. The constraints are of two possible intents: assume and commit [20]. Assume constraints express assumptions on the occurrence times of input events, and commit constraints define the limits on occurrence times of the output events to be satisfied by

[1]A signal transition on a waveform corresponds to an action in HAAD specification and an event in a leaf TD specification

the implementation under the input assumptions. More formally, a leaf timing diagram TD is defined as follows [17].

Definition 2.1: A leaf timing diagram TD is a 4-tuple, TD = (W, E, A, C), where

- W= $\{w_1, \ldots, w_m\}$ is the set of waveforms;
- E = $\{e_1, \ldots, e_n\}$ is the set of events on the waveforms;
- A = $\{A_1, \ldots, A_p\}$ is the set of assume constraints;
- C = $\{C_1, \ldots, C_q\}$ is the set of commit constraints.

A timing constraint $(e_u, e_v, [T_{uvmin}, T_{uvmax}]) \in A \cup C$, $T_{uvmin} \leq T_{uvmax}$, T_{uvmin}, $T_{uvmax} \in \Re$ (the set of real numbers), from e_u (the source event of the constraint) to e_v (the sink event) represents the following inequality between the occurrence times t_u and t_v of e_u and e_v, respectively:

$$T_{uvmin} \leq t_v - t_u \leq T_{uvmax}. \tag{2.1}$$

A timing constraint can be one of two types, that is, precedence or concurrence constraint. A precedence constraint ($T_{uvmin} \geq 0$) from e_u to e_v denotes that e_u must precede e_v in occurrence time, while a concurrence constraint ($T_{uvmin} \leq 0$, $T_{uvmax} \geq 0$) from e_u to e_v means that there is no definite order in the occurrence times of the two events. In a TD, it is possible that two or more constraints sink at the same event e_v, all these constraints are combined together to determine the occurrence time of e_v. For an event e_v, let U be the set of all u such that e_u is an event and $(e_u, e_v, [T_{uvmin}, T_{uvmax}])$ is a constraint to e_v. A HAAD specification allows three types of constraint combinations defined as follows.

Definition 2.2: If the constraints to e_v are combined using a linear operator, then the occurrence time t_v must lie within the intersection of the intervals $[t_u + T_{uvmin}, t_u + T_{uvmax}]$, for all $u \in U$. In other words, all the constraints to e_v are to be satisfied simultaneously as given in (2.2).

$$\max_{u \in U}\{t_u + T_{uvmin}\} \le t_v \le \min_{u \in U}\{t_u + T_{uvmax}\} \tag{2.2}$$

Definition 2.3: If the constraints to e_v are combined using the max operator, then the earliest (latest) occurrence time t_v is determined by the source event e_u which makes $t_u + T_{uvmin}$ ($t_u + T_{uvmax}$) be the maximum among all the u's as given in (2.3). All the constraints to e_v must be precedence.

$$\max_{u \in U}\{t_u + T_{uvmin}\} \le t_v \le \max_{u \in U}\{t_u + T_{uvmax}\}. \tag{2.3}$$

This composition can model, for instance, the timing behaviour of a rising transition on the output of an AND gate caused by the rising transitions on the inputs of the gate.

Definition 2.4: If the constraints to e_v are combined using the min operator, then the earliest (latest) occurrence time t_v is determined by the source event e_u which makes $t_u + T_{uvmin}$ ($t_u + T_{uvmax}$) be the minimum among all the u's as given in (2.4). All the constraints from e_u to e_v must be precedence.

$$\min_{u \in U}\{t_u + T_{uvmin}\} \le t_v \le \min_{u \in U}\{t_u + T_{uvmax}\}. \tag{2.4}$$

This composition can model, for instance, the timing behaviour of a falling transition on the output of an AND gate caused by falling transitions on the inputs of the gate.

Figure 2.2 and Figure 2.3 show leaf timing diagrams of a memory device and of a memory controller. The operations on the interfaces are an end of a read cycle, followed by a write cycle and then beginning of another read cycle.

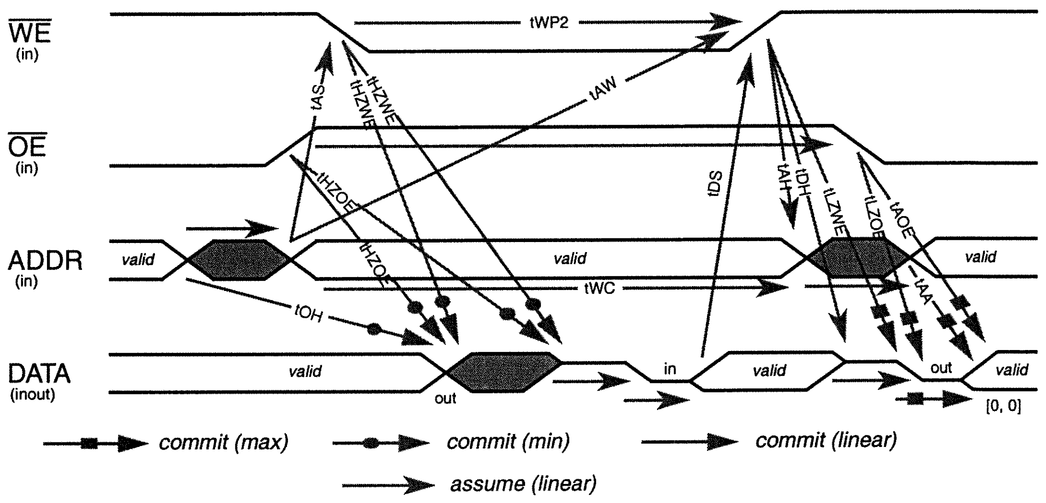


Figure 2.2 A timing diagram specification of a RAM interface (end of a read cycle - a write cycle - beginning of another read cycle)

Table 2.1: Constraints for the timing diagram in Figure 2.2

Name	Delay Value (ns)	Name	Delay Value (ns)
tAA	(0, 10]	tHZOE	(0, ∞)
tAH	(0, ∞)	tHZWE	(0, 7]
tAOE	(0, 7]	tLZOE	(0, 7]
tAS	(0, ∞)	tLZWE	(9, ∞)
tAW	(10, ∞)	tOH	(9, ∞)
tDH	(0, ∞)	tWC	(15, ∞)
tDS	(8, ∞)	tWP2	(12, ∞)

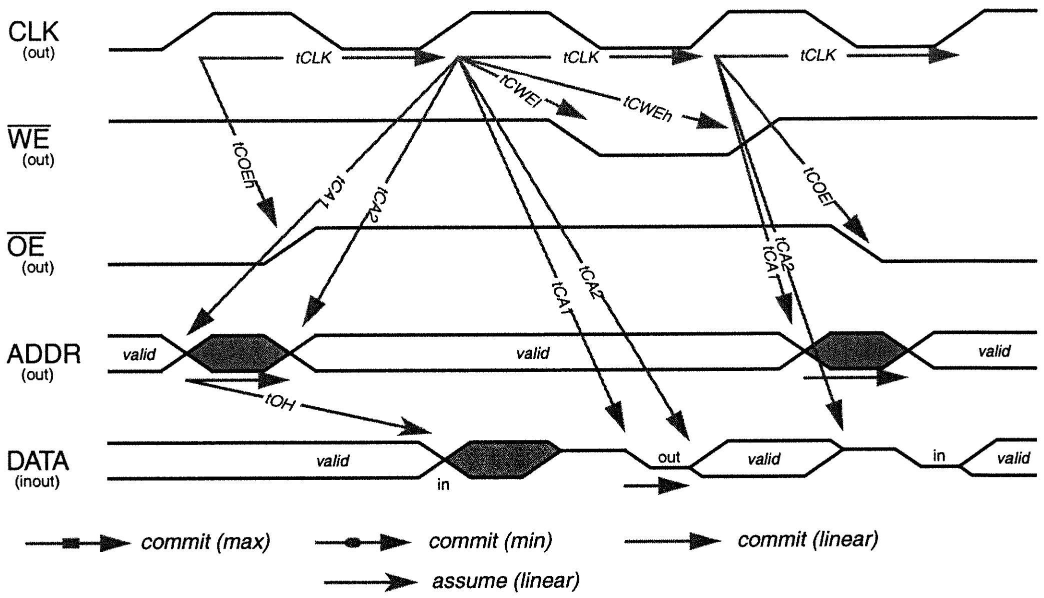


Figure 2.3 A timing diagram specification of a controller interface (End of a read - write cycle - beginning of another read cycle)

Table 2.2: Constraints for the timing diagram interface in Figure 2.3

Name	Delay Value (ns)	Name	Delay Value (ns)
tCA1	[18, 19.8]	tCOEh	[31.8, 33]
tCA2	[18, 20]	tCWEl	[22, 28]
tCLK	[25.73, 25.73]	tCWEh	[38.8, 45]
tCOEl	[16, 18]	tOH	(3, ∞)

2.2 Verification of Interface Specifications

In this section, we summarize the formal semantics and the verification of causality and compatibility of interface specifications as discussed in [36], [50]. The semantics of causality and compatibility guarantees that causal HAAD specifications are realizable (can be simulated by a causal system [50]) and components of compatible specifications can interoperate correctly when forming a system. We show that causality and compatibility conditions can be verified by computing the maximum time separations between events in constraint graphs extracted from the specifications.

In [50], the realizability of a leaf HAAD specification is stated in terms of the existence of a causal block machine derived from the specification, the derivation is based on a partition of the action set of the specification. The authors in [50] proved that all causal block machines derived from an action diagram have the same timed trace set as that of the TD specification. The compatibility of an interface specification is then defined as the absence of dead lock when any combinations of causal block machines are executed. It is thus guaranteed that the components included in a compatible specification can interoperate correctly in a system. In the case of systems with linear-only constraints, the authors in [50] give the compatibility conditions. The compatibility conditions consists of realizability of each interface specification, consistency and satisfiablility of the composition of the interface specifications. The work is extended later in [36] to systems specified by leaf TDs containing linear, latest, earliest constraints and also delay correlation.

In the following, we summarize the consistency, causality and compatibility conditions from [13][36][50] which are applicable to leaf HAAD specifications.

The consistency condition of a timing diagram is as given in [13] and [35]: A TD is consistent if all events in the TD can be assigned an occurrence time without violating any timing constraint.

To state causality and compatibility conditions, we first give a definition of the maximum and minimum time separation between events satisfying all constraints in a TD.

Definition 2.5: Given a TD = (W, E, A, C), a start event s and an end event e, s, e $\in$ E, the maximum ($\Delta(s, e)$) and the minimum ($\delta(s, e)$) time separations from s to e are defined as:

$$\Delta(s, e) = \max(\tau(e) - \tau(s)) \qquad (2.5)$$

and

$$\delta(s, e) = \min(\tau(e) - \tau(s)) \qquad (2.6)$$

subject to the constraints in $A \cup C$

The causality conditions of a TD are developed in [50]. We need the following definitions to state the causality conditions.

Definition 2.6: A block B of a TD = (W, E, A, C) is a nonempty subset of events in E, i.e., $B \subseteq E$. A block partition P of a TD is a set of disjoint blocks of all the events in E, i.e., a collection of blocks satisfying the following two conditions.

a) $\forall\, B_1, B_2 \in P$, either $B_1 = B_2$ or $B_1 \cap B_2 = \varnothing$ and

b) $\cup_{B \in P} B = E$.

For every block B of a partition P, there are **triggers** and **local constraints** of the block. A **trigger** of a block is the source event in another block whose sink event is in the block, i.e.,

$$\text{trigs}(B) = \{e_u \mid (e_u, e_v, [T_{uvmin}, T_{uvmax}]) \in A \cup C,\ e_u \notin B \text{ and } e_v \in B\}. \qquad (2.7)$$

The **local constraints** of a block B are the constraints that either relate events in the block or relate triggers of the block to the events in the block, i.e., the local constraints are $\{(e_u, e_v, [T_{uvmin}, T_{uvmax}]) \in A \cup C \mid e_u \in \text{trigs}(B) \cup B \text{ and } e_v \in B\}$. Let B(v) denote the block containing event e_v.

The causality conditions of a TD can now be stated as follows ([20][50]): A leaf TD is causal if a partition P of the TD satisfying the following three conditions can be found.

Causality Condition 1: *In* and *out* actions do not share a block B of P, where *In* (*out*) actions correspond to signal transitions on input (output) ports,

Causality Condition 2: For all events $e_v \in B$ and for all triggers $e_u \in \text{trigs}(B(v))$,

$$\delta(e_v, e_u) \leq 0, \tag{2.8}$$

where $\delta(e_v, e_u)$ is the minimum time separation from e_v to e_u satisfying the local constraints of B. In other words, the triggers must be in the past of all $e_v \in B$ (well-defined triggers [50]).

Causality Condition 3: for all $B \in P$, the maximum time separations between pairs of triggers of B computed using all the constraints in the TD are strictly tighter than those between the same pairs of triggers computed using the local constraints of B.

Interface specifications $TD_1, \ldots, TD_n$ are compatible if the following three conditions are satisfied [36][50]:

Compatibility Condition 1: Each TD_i, $i = 1, \ldots, n$, is causal.

Compatibility Condition 2: The composition TD of TD_i, i=1, …, n, is consistent, where the composition TD is defined as a timing diagram that includes all the actions and all the commit constraints from all TD_i, $i = 1, \ldots, n$.

Compatibility Condition 3 (satisfiablility in [13] [35]): The maximum time separations between all pairs of events computed in the composition TD must be less than those between the same pairs of events computed in the TD including all the actions and all the assume constraints from all TD_i, $i = 1, \ldots, n$.

In the next section, we shall show that the consistency, causality and thus the compatibility conditions can be verified by computing the maximum time separation of events.

2.3 From Verification of Timing Diagrams to Maximum Time Separation

To verify the consistency of a TD, we can create a directed graph $G = \langle E, R \rangle$, based on the TD specification, where E, the set of events, represents actions in the TD; R, the set of rule edges, represents the timing constraints in the TD.

A TD is consistent if all actions in the TD can be assigned occurrence times without violating any timing constraint. That is, for any two events in G, it is possible to compute the maximum time separation between them. In the case of linear-only systems, a TD is consistent if there is no negative cycle in G [35], [20].

The second causality condition can be verified by computing the maximum time separation of events in the constraint graph derived from the composition TD. The condition can be verified by checking whether the minimum time separations from the events in the block to the related triggers are smaller than 0, i.e., the maximum time separations from the triggers to their related sink events in the block are greater than 0.

The third causality condition is explicitly stated as comparing the maximum time separation between the same pairs of events in two different constraint graphs.

Since the compatibility of interface specification can be verified by consistency and causality of individual or composition of timing diagrams involved in the specifications, we can conclude that compatibility verification can be solved by computing the maximum time separation of events in constraint graphs extracted from the specification.

2.4 Related Work

As illustrated in Section 2.3, the causality and compatibility verification of interface specifications can be reduced to computing the maximum time separations between pairs of events in constraint graphs. In Chapter 3, we shall show that the maximum time separation of events can also be applied to solve other safety timing properties of interface specifications.

The maximum time separation problem has been explored by many researchers from different fields. Algorithms have been proposed under various restrictions on the constraint systems ([21][22][37][50][62][78][80][83]). All the restrictions can be put into two categories: a) on the type of constraints, and b) on whether the events in the specification are allowed to occur repeatedly. When the events in the specification can occur only once, the problem is well solved in any combination of timing constraints. The complexity of the algorithms ranges from polynomial for the linear-only systems to NP-complete for max-plus-min-plus-linear systems. This can be summarized as follows:

For systems with linear-only constraints, a shortest path algorithm, e.g., Floyd-Warshall algorithm [33][35] can be used, the complexity of the algorithm is $O(n^3)$ where n is the number of events in the constraint graph. Vanbekberge [78], and McMillan and Dill [62] proposed algorithms for max-only and min-only systems with complexity $O(n^3)$. Yen et al. [83] gave an algorithm to calculate the maximum separations in systems combining either max or min constraints with linear constraint in the complexity of $O(n^3 \log n)$. Walkup and Borriello [80][81] presented an algorithm for solving the same problem with the complexity of $O(n^6)$ which is unproven. The complexity of the algorithm by McMillan and Dill [62] is pseudo polynomial in $O(n^3\Sigma_{ij}s_{ij})$, where $\Sigma_{ij}s_{ij}$ is the sum of all the initial timing separations (the timing bounds in Relations (2.2) and (2.3)). They showed that whenever both max and min constraints are present, the maximum time separation problem becomes NP-complete. A branch-and-bound algorithm based on an algorithm for max-plus-linear or min-plus-linear constraints is appropriate in such cases. Burks et al [15] proposed a branch-and-bound algorithm and a mixed integer linear programming method to compute the maximum time separation in systems containing min and max constraints. Girodias et al. [37] use Constraint Logic Programming (CLP) environment based on Relational Interval Arithmetical (RIA) to solve the maximum time separation problem for linear-only, max(min)-only, max(min)-plus-linear, and max-plus-min-plus-linear systems with similar complexity as the other algorithms. Their approach can also solve the problem when delay correlation exists in the system, a problem which is difficult to handle by the other approaches [36]. Chakraborty et al. [22] presented a polynomial-time approximate algorithm for computing the maximum time separations in systems containing min-plus-max constraints.

All the above algorithms are only applicable to acyclic finite specifications. Cycles are necessary to model repetitive interface behaviours (e.g., Loop over a leaf TD in

HAAD specifications). Cyclic behaviours make the problem more complicated because the effective number of events and constraints becomes infinite.

There are fewer concrete results for cyclic systems. Hulgaard et al. ([5] [41] and [42]) give an algorithm for computing the maximum time separations of events in systems specified as cyclic process graphs with latest-only constraints. The algorithm is based on a functional decomposition technique that permits an implicit evaluation of an infinitely unfolded process graph. The algorithm has also been applied for timing analysis of a class of safe Petri nets with no conditional behaviour, where interval time delays are specified on the places of the nets and the latest firing semantics are adopted [41]. Chakraborty et al. [21] worked on cyclic systems with min-plus-max constraints. They restricted the systems to be tightly-coupled. They can compute the maximum time separations in such cyclic systems by using the approximate algorithm [22] developed for acyclic constraint graphs. They applied the method to the timing verification of an asynchronous differential equation solver chip.

In the following four chapters, we concentrate on developing new algorithms for computing the maximum time separations in constraint graphs transformed from HAAD specifications containing a Loop operator over a TD, a type of cyclic systems containing linear-plus-latest constraints. We define the constraint graphs transformed from the HAAD specifications and the maximum time separation problem in the constraint graphs in Chapter 3. The maximum time separation in such a constraint graph is the maximum over an infinite number of time separations in a graph containing infinite number of events and constraints. We solve the problem by starting from an algorithm for computing the max time separations in graphs with a finite number of events and constraints in Chapter 4. We then extend the algorithm to solve our problem in Chapter 5. Since the complexity of the algorithm developed in Chapter 5 depends on the delay values in the specification and is impractical for real complex problems, we restrict the HAAD specification to causal ones and present a more practical solution to this restricted problem in Chapter 6.

Chapter 3

The Maximum Time Separation Problem

In Chapter 2, we have shown that the timing verification problems of interface specifications can be reduced to finding the maximum time separations in constraint graphs transformed from the specifications. As summarized in Section 2.4, the maximum time separation problem has been solved in various restricted systems. Here, we are interested in interface timing verification problems in more general systems where the events in the specifications can occur an infinite number of times. This corresponds to HAAD specifications consisting of a Loop over a leaf TD. As an example, consider the repetitive READ of a microprocessor (inspired by MC68360) from a memory specified by a Loop over a leaf TD which describes the interface behaviour of one READ cycle. The Loop and the leaf TD specifications are illustrated in Figure 3.1 and Figure 3.2, respectively.

In this chapter, we first define in Section 3. 1 a constraint graph from such HAAD specifications. Note, our constraint graphs are more general than the so-called process graphs in Hulgaard et al [42] which allow only the latest timing constraints. We allow linear, latest and mixed linear and latest timing constraints. We present the well formedness conditions and the execution model of such constraint graphs in Section 3. 2. Then we define a finite unfolded constraint graph in Section 3. 3. After that, we give the definition of the maximum time separation problem in such constraint graphs, and show on an example in Section 3. 4 how to formulate some useful safety timing properties as the maximum time separation problems.

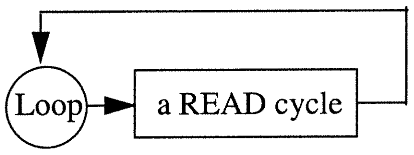


Figure 3.1 HAAD hierarchy for a repetitive microprocessor READ from a memory

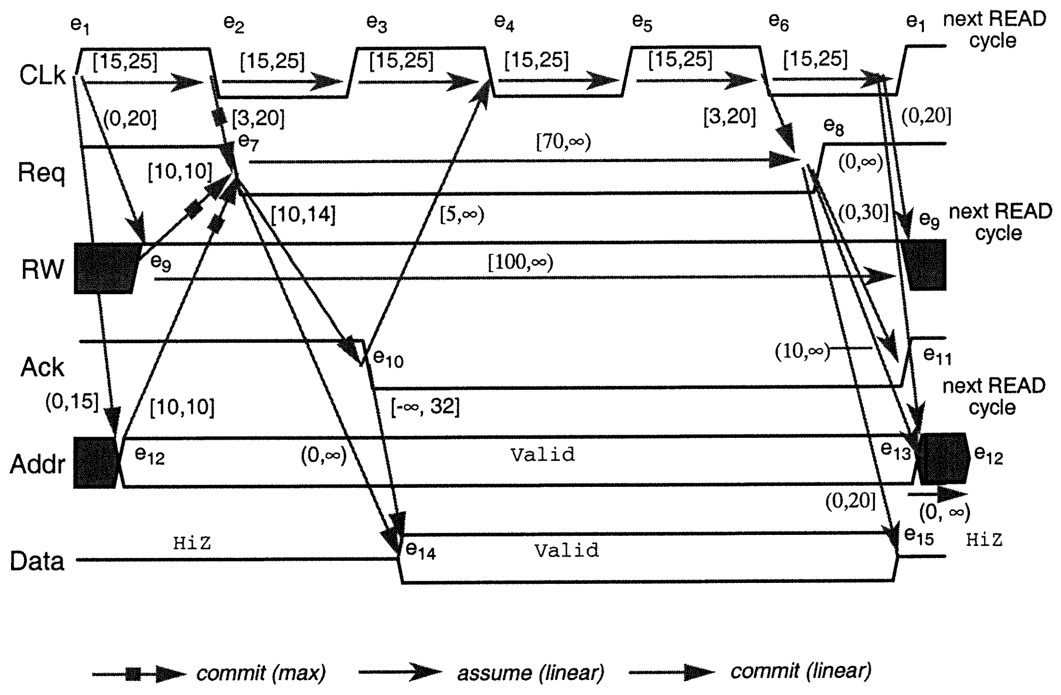


Figure 3.2 A leaf timing diagram specifying one READ cycle of a microprocessor.

3. 1 Constraint Graphs

In this section, we define a constraint graph from a HAAD specification.

Definition 3.1: Given a HAAD specification in the form of Loop over a leaf TD, where TD = (W_{td}, E_{td}, A_{td}, C_{td}), we define a constraint graph, G = < E, R> as follows:

The events in E are in one to one correspondence to the events in E_{td}, i.e., E = E_{td};

The rule edges in R are transformed from the constraints in the HAAD specification. The transformation rules are as follows:

For each timing constraint (u_{td}, v_{td}, [T_{uvmin}, T_{uvmax}]) $\in A_{td} \cup C_{td}$, where u_{td}, v_{td} are events in the same operation cycle (e.g., events in the same READ cycle), there is an edge $u \xrightarrow{[T_{uvmin}, T_{uvmax}], 0} v \in R$.

For each timing constraint in the HAAD specification, $(u_{td}, v_{td}, [T_{uvmin}, T_{uvmax}]) \in A_{td} \cup C_{td}$, where u_{td}, v_{td} are events in two adjacent operation cycles, there is an edge $u \xrightarrow{[T_{uvmin}, T_{uvmax}], \varepsilon} v \in R$ in G, where ε is either -1 or 1 depending on the relative position of u_{td} and v_{td} in the specification: if u_{td} is in the current operation cycle and v_{td} is in the next cycle, then $\varepsilon = 1$; if u_{td} is in the next operation cycle and v_{td} is in the current cycle, then $\varepsilon = -1$.

In the example of the HAAD specification modeling the repeated READ of the microprocessor from the memory, there are 15 events (e_1 to e_{15}) in the leaf TD as shown in Figure 3.2. All the rule edges transformed from the constraints between events in the same READ cycle have $\varepsilon = 0$; the rule edge corresponding to the constraint from e_6 of the current READ cycle to e_1 of the next READ cycle in Figure 3.2 has $\varepsilon = 1$; and the rule edge corresponding to the constraint from e_1 of the next READ cycle to e_{13} of the current READ cycle in Figure 3.2 has $\varepsilon = -1$. We thus get the constraint graph as shown in Figure 3.3.

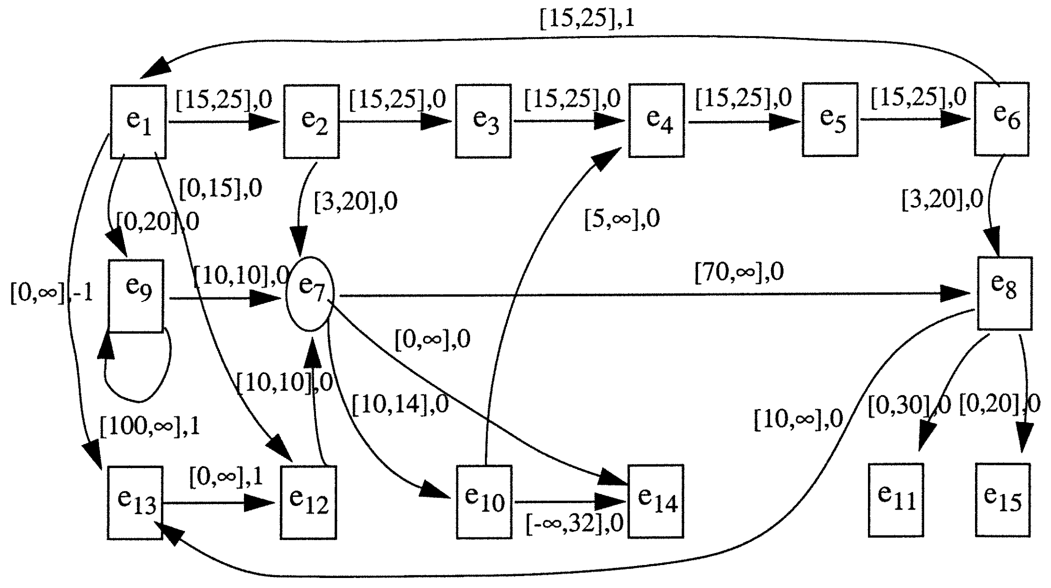


Figure 3.3 A constraint graph specifying a repeated microprocessor READ from a memory derived from the HAAD specification in Figure 3.1 and Figure 3.2.

In a constraint graph, the events in E and the rule edges in R are repeatable (will appear an infinite number of times). We denote the k-th occurrence of event v as v_k, and the occurrence time of v_k as $\tau(v_k)$. Each rule edge in a constraint graph corresponds to an infinite number of constraints ($\forall k \geq \max(0, \varepsilon)$, $T_{uvmin} \leq \tau(v_k) - \tau(u_{k-\varepsilon}) \leq T_{uvmax}$), all the occurrences of a constraint are of the same type (precedence or concurrency as defined in Section 2.2.1). When more than two edges in G have a common sink event v, the occurrence time of v_k, $k \geq 0$, is affected by a combination of the constraints. We consider here three types of combinations: latest (max), conjunctive (linear), and mixed latest and linear combinations, defined as follows.

Definition 3.2: If all the constraints to event $v \in E$ are combined with the linear (conjunctive) operator, then the occurrence time of v_k, $k \geq 0$, is determined by Relation (3.1).

$$max_{u \in sources(v_k)}\{\tau(u) + b_{uv}\} \leq \tau(v_k) \leq min_{u \in sources(v_k)}\{\tau(u) + B_{uv}\} \text{ ,(3.1)}$$

where u ∈ sources(v_k) if there is an edge $u \xrightarrow{[b_{uv}, B_{uv}], \varepsilon} v$ in G and k ≥ max(0, ε). The edge can be either a precedence or a concurrency constraint.

Definition 3.3: If all the constraints to v ∈ E are combined using the max (latest) operator, then the occurrence time of v_k, k ≥ 0, is determined by Relation (3.2).

$$max_{u \in preds(v_k)}\{\tau(u) + d_{uv}\} \leq \tau(v_k) \leq max_{u \in preds(v_k)}\{\tau(u) + D_{uv}\} \quad (3.2)$$

where u ∈ preds(v_k) if there is an edge $u \xrightarrow{[d_{uv}, D_{uv}], \varepsilon} v$ in G, k ≥ max(0, ε), and the edge must be a precedence constraint.

Definition 3.4[1]**:** If some constraints to event v, v ∈ E, are combined by a linear operator while the others are combined by a max operator, then the occurrence time of v_k, k ≥ 0, is determined in the following manner: first, the linear and latest operators play the role independently, then their results affect the occurrence time of v_k conjunctively, that is, ∀ k ≥ 0, Relation (3.1) is satisfied where u ∈ sources(v_k) if the edge $u \xrightarrow{[b_{uv}, B_{uv}], \varepsilon} v$ in G is involved in a linear combination and k ≥ max(0, ε), meanwhile, Relation (3.2) is also satisfied where u ∈ preds(v_k) if the edge $u \xrightarrow{[d_{uv}, D_{uv}], \varepsilon} v$ in G is involved in a max combination, k ≥ max(0, ε), and the edge must be a precedence constraint.

Instead of [T_{uvmin}, T_{uvmax}], we use the notation [d_{uv}, D_{UV}] and [b_{uv}, B_{UV}] to identify the bounds of timing constraints in max and linear combinations, respectively, and

1. A constraint combination like this can always be decomposed into a latest combination consisting of all the constraints originally involved in a latest combination, a linear combination consisting of all the constraints originally involved in a linear combination, and another linear combination consisting of two timing constraints with the delay value [0, 0], one from the result of the latest combination, the other one from the result of the linear combination. Then the constraint will have a correspondent in HAAD specification as defined in Definition 2.3 and Definition 2.4.

we use preds(v_k) and sources(v_k) as the source events of constraints involved in max and linear combinations, respectively. We do these to distinguish the type of the constraints in max combinations, which are restricted to precedence, from those in linear combinations, which can be precedence or concurrency constraints.

The constraint graph includes all the information contained in the original HAAD specification. We define an infinite unfolded constraint graph G^{∞} to explicitly describe the infinite interface behaviour.

Definition 3.5: $G^{\infty} = <E^{\infty}, R^{\infty}>$ is a graph with an infinite number of events and rule edges, where $E^{\infty} = \{v_k \mid v \in E, k \geq 0\} \cup \{root\}$. Node *root* is included in E^{∞} to model the initial start up behaviour of the system, R^{∞} = {constraints from Relations (3.1) and (3.2) for v_k, $k \geq 0\} \cup R_0$, where R_0 contains user specified start up rules in the form of $root \xrightarrow{[T_{vmin}, T_{vmax}]} v_i$, where $v \in E$, $i \geq 0, T_{vmax} \geq T_{vmin} \geq 0$. They may be combined with other constraints using linear or latest operators.

As shown in the example of the repeated READ of the microprocessor to the memory, the timing constrains are all in the form of intervals, there are many possibilities for the occurrence times of events. We define an execution of the constraint graph to model one of these possibilities.

Definition 3.6: An execution of the behaviour defined by G is an assignment of occurrence times to events in E^{∞} which is consistent with the constraint system represented by R^{∞}.

We restrict our analysis to well-formed constraint graphs such that in any execution of the constraint graph, the occurrence times of all events in G cannot go backwards[1] as the event indices advance. The precise definition of a well-formed

[1]We allow the occurrence times of some events stuck at some value with the advance of the event indices to comply with the specifications in [42]

constraint graph is given in Section 3. 2

3. 2 Well-formed Constraint Graphs

As stated in Definition 3.6, in an execution of a constraint graph, all the constraints represented by G^{∞} must be respected. This requires that Relation (3.1) (related to all the linear constraints), the left hand side of Relation (3.2) (related to lower bounds of all the max constraints), as well as part of the right hand side of Relation (3.2) (relate to the upper bounds of the max constraints which determine the right hand side max function of Relation (3.2)) must be satisfied simultaneously. The upper bounds of the max constraints which fail in determining the max function of the right hand side of Relation (3.2) do not play any role in determining the occurrence times of events in an execution of a constraint graph.

To derive well-formedness conditions which guarantee that the occurrence times of all the events in G will not retreat as the event indices advance, we first analyse all possible timing behaviour of every event in G with the advance of its occurrence index while respecting both bounds of all the linear constraints and the lower bounds of all the max constraints in G. From there, we get sufficient conditions that prevent the occurrence of any event from going backward with the advance of the event index. Since the occurrence times of events satisfying all the constraints in G^{∞} are constrained more than those satisfying all the linear constraints and the lower bounds of all the max constraints (by the upperbounds of max constraints which win in determining the max function of the right hand side of Relation (3.2)), the solution space of the occurrence times in the executions of a constraint graph are subsets of those satisfying the lower bounds of the max constraints and both bounds of all the linear constraints. Therefore, the sufficient conditions that prevent the occurrence of any event from going backward with the advance of the event index while satisfying all the linear constraints and the lower bounds of all the max constraints are also sufficient to prevent the occurrence of any event from going backward with the advance of the event index while satisfying all the constraints in G.

To analyse all the possible changes in occurrence times of an event, we construct a directed graph $G_m = \langle E, R_m \rangle$[1] as follows.

Definition 3.7: Given a constraint graph $G = \langle E, R \rangle$, the graph $G_m = \langle E, R_m \rangle$ is constructed as follows: for each edge $u \xrightarrow{[d_{uv}, D_{uv}], \varepsilon} v$ in R involved in a max combination, there is an edge $v \xrightarrow{-d_{uv}, -\varepsilon} u$ in R_m, for each edge $u \xrightarrow{[b_{uv}, B_{uv}], \varepsilon} v$ in R involved in a linear combination, there are two edges $u \xrightarrow{B_{uv}, \varepsilon} v$ (if B_{uv} is not ∞) and $v \xrightarrow{-b_{uv}, -\varepsilon} u$ (if b_{uv} is not $-\infty$) in R_m.

To prevent the occurrence times of all events in G_m from retreating as the event indices advance, every event in G_m has to lie on a cycle in G_m, otherwise, the event may be isolated with no timing constraint to it, and thus the event could be assigned an arbitrarily occurrence time. This would violates the requirement that the occurrence times of all the events cannot retreat as the event indices advance. We also require that there is a time separation between any two events in G_m, i.e., for all pairs of nodes i and j in G_m, there is a path from i to j and from j to i, in other words

Condition 1: G_m is strongly connected [38], that is, $\forall u, v \in E$, there is a path from u to v and a path from v to u in G_m.

To satisfy the constraints represented by G_m, $\forall k \geq \max(0, \varepsilon)$, $\tau(v_k) - \tau(u_{k-\varepsilon}) \leq w$ must hold for every edge $u \xrightarrow{w, \varepsilon} v$ in G_m, and $\forall k \geq \max(0, \varepsilon(c))$,

$$\tau(v_k) - \tau(v_{k - \varepsilon(c)}) \leq w(c) \tag{3.3}$$

[1]G_m is the dual of the compulsory graph used in [83] for computing time separations in finite graphs.

has to be satisfied for every event v along a cycle, where w(c) is the sum of the w values of the edges along the cycle.

We distinguish three types of cycles in G_m: (1) max cycles where all the edges along the cycles correspond to the rule edges in G involved in max combinations; (2) linear cycles where all the edges along the cycles correspond to the constraints involved in linear combinations; and (3) mixed cycles where some edges along the cycle correspond to the rule edges involved in linear combinations and others correspond to those involved in max combinations.

If *Condition 1* is satisfied, then every event in G_m lies on at least one of the above three types of cycles.

In the following, we first analyse the kinds of changes in occurrence times of the events along each type of cycle as the event indices change, and derive the sufficient conditions to prevent the occurrence times of the events from retreating as the event indices advance. The well-formedness conditions of G are then introduced.

3.2.1 Events Along a Max Cycle

As all constraints involved in max combinations are precedence constraints, we have $w(c) \leq 0$. From Relation (3.3), $\forall\, k \geq \max(0, \varepsilon(c)), \tau(v_k) - \tau(v_{k-\varepsilon(c)}) \leq w(c)$ must hold for every event v along the cycle, i.e., v_k has to occur no later than $v_{k-\varepsilon(c)}$. There are three cases to consider depending on the sign of $\varepsilon(c)$.

a) $\varepsilon(c) < 0$, then $k < k - \varepsilon(c)$, the events along the cycle can not go backward in their occurrence times as the event indices advance from k to $k - \varepsilon(c)$.

b) $\varepsilon(c) = 0$, then w(c) has to be 0 (or else the system of constraints is inconsistent). Therefore, all the events along the cycle have to occur at the same time. We require that all the events along such a max cycle be represented as one event in the constraint graph.

c) $\varepsilon(c) > 0$, then $k > k - \varepsilon(c)$, and the occurrence time of events along the cycle could retreat as the event indices advance from $k - \varepsilon(c)$ to k.

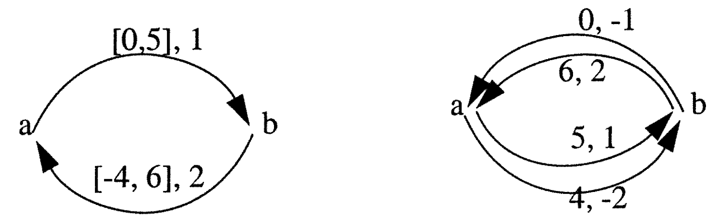
Generally, if we can prove 'if not A then not B', then A is a necessary condition of B, and if we can prove 'if A then B', then A is a sufficient condition of B. Therefore, $\varepsilon(c) < 0$ is a necessary and also a sufficient condition to prevent the occurrence time of events along a max cycle from retreating as the event indices advance.

3.2.2 Events Along a Linear or a Mixed Cycle

For events along a linear or a mixed cycle, it follows from Relation (3.3) that for every $k \geq \max(0, \varepsilon(c))$, $\tau(v_k) - \tau(v_{k-\varepsilon(c)}) \leq w(c)$ must hold for every event v along the cycle. There are also three cases to consider depending on the sign of $\varepsilon(c)$.

a) $\varepsilon(c) < 0$:

If $w(c) \geq 0$ then v_k may still be assigned an occurrence time smaller than that of $v_{k-\varepsilon(c)}$, i.e., the occurrence time of v will not retreat when the event indices advance from k to $k - \varepsilon(c)$. For example, in the constraint graph shown in Figure 3.4a, all constraints are involved in linear combinations.



a) A constraint graph G.   b) G_m corresponding to G.

Figure 3.4 A well formed constraint graph

In corresponding G_m (Figure 3.4b), there is a cycle with $\varepsilon(c) = -2 - 1 = -3 < 0$ and $w(c) = 4 + 0 = 4 > 0$. We can still assign occurrence times of events in G as follows: $\tau(a_0) = 1$, $\tau(b_0) = 0$, $\tau(a_1) = 3$, $\tau(b_1) = 2$, …, and $\tau(a_i) = 2i + 1$, $\tau(b_i) = 2i$, for all $i > 1$. All timing constraints are satisfied because $\tau(b_{i+1}) - \tau(a_i) = 2i + 1 - 2i = 1 \in [0, 5]$, $\tau(a_i) - \tau(b_{i+2}) = 2i + 1 - 2(i + 2) = -3 \in [-4, 6]$. Hence, in the constraint graph in Figure 3.4a, all event occurrence times advance as the event indices advance. We will further discuss the reasons for the well-formedness of this graph at the end of this subsection.

If $w(c) < 0$, then v_k has to be assigned an occurrence time greater than that of $v_{k-\varepsilon(c)}$, and the occurrence times of events along the cycle must advance as the event indices advance from $k - \varepsilon(c)$ to k.

The condition $\varepsilon(c) < 0 \Rightarrow w(c) < 0$ is thus only a sufficient condition which guarantees that the occurrence times of events along such a cycle will not retreat as the event indices advance.

b) $\varepsilon(c) = 0$:

If $w(c) < 0$, then $\tau(v_k) - \tau(v_k) \leq w(c)$ can not be satisfied, i.e., the system of constraints is inconsistent.

If $w(c) \geq 0$, $\tau(v_k) - \tau(v_k) \leq w(c)$ always holds, it does not add any constraints on the occurrence time of $\tau(v_k)$. Consequently, $\varepsilon(c) = 0 \Rightarrow w(c) \geq 0$ is a necessary condition.

c) $\varepsilon(c) > 0$:

If $w(c) \leq 0$, then v_k has to be assigned an occurrence time no later than $v_{k-\varepsilon(c)}$ and $k > k - \varepsilon(c)$. Hence the occurrence times of events along the cycle can retreat when the event indices advance.

If $w(c) > 0$, then v_k can be assigned an occurrence time later than that of $v_{k-\varepsilon(c)}$, the occurrence times of events will not go backwards when the event indices go forwards.

Therefore, $\varepsilon(c) > 0 \Rightarrow w(c) > 0$ is a necessary and also a sufficient well-formedness condition.

The above discussion yields the following 3 well-formedness conditions.

Condition 2: For all max cycles c in G_m, $\varepsilon(c) < 0$ must hold[1].

Condition 3: For all linear or mixed cycles c in G_m.

a. if $\varepsilon(c) = 0$ then $w(c) \geq 0$,

b. if $\varepsilon(c) > 0$ then $w(c) > 0$.

Condition 4: If neither Condition 2 nor Condition 3 apply, then the event must be on a mixed cycle such that $w(c) < 0$ and $\varepsilon(c) < 0$.

Definition 3.8: A constraint graph G is defined as *well-formed* if the above Conditions 1 through 4 are satisfied.

In the example shown in Figure 3.4, *Condition 1* is obviously satisfied; *Condition 2* doesn't apply; *Condition* 3 holds because there are four linear cycles in G_m: c1 with $\varepsilon(c1) = 0$ and $w(c1) = 4$, c2 with $\varepsilon(c2) = 3$ and $w(c2) = 11$, c3 with $\varepsilon(c3) = 0$ and $w(c3) = 10$, and c4 with $\varepsilon(c4) = -3$ and $w(c4) = 4$. Since all events satisfy Con-

[1]The $\varepsilon(c)$ in G_m equals to $-\varepsilon(c)$ in G, hence, this condition is the same as in [42] for latest only systems.

ditions 1 to 3 with $\varepsilon(c) \geq 0$, the constraint graph in Figure 3.4 is well-formed.

3. 3 Finite Unfolded Constraint Graphs

To compute the maximum time separation of events in G, we need the information about the occurrence times $\tau(v_k)$ for all $k \geq 0$. We examine the timing behaviour of the events by unfolding G step by step. We thus need to define a finite unfolded constraint graph here.

$$\text{Let } U_\alpha = \{v_\alpha \mid v \in E\}, \alpha \geq 0 \tag{3.4}$$

From Equation (3.4), for each $v_\alpha \in U_\alpha$, $v_{\alpha+k}$ is in $U_{\alpha+k}$ for all $k > 0$. Therefore, there is the same number of events in U_α for all $\alpha \geq 0$. Let $|U_\alpha| = n$, n is also the number of events in G.

Define iteratively the α-unfolded constraint graph $G^\alpha = <E^\alpha, R^\alpha>$ of G as follows:

$E^0 = root$, $R^0 = \emptyset$;

$E^1 = E^0 \cup U_0$, $R^1 = R^1{}_0 \cup \{$constraints between events in $U_0\}$, where $R^1{}_0$ is a non-empty finite number of constraints from *root* to events in U_0 which is composed of two parts: the start up rule edges to the events in U_0 given by the user, $R^1{}_{01}$, and the start up rules $R^1{}_{02}$ added to assure that all events occur after *root*. $R^1{}_{02}$ is constructed as follows:

For all $v_0 \in U_0$, if v_0 is a sink event of constraints involved a linear combination, add to $R^1{}_{02}$ a precedence constraint $root \xrightarrow{[0, \infty]} v_0$ to the linear combination, else if it is a sink event of constraints with a max combination, add a precedence constraint

$root \xrightarrow{[0,0]} v_0$ to the max combination, else it is a sink event of constraints with mixed combinations, then add a constraint either as in a linear combination or as in a max combination. The constraints in R^1_{02} add paths from all events to *root* in U_0. When we say that there is a path from all events to *root* in $U_{\alpha-1}$, $\alpha > 0$, we mean that there is a path from the event to *root* in G_m^{α}, which can be constructed from G^{α} in the same way as G_m from G in Definition 3.7. We can thus guarantee that all events in U_0 occur after the *root*.

For all $\alpha > 1$, $E^{\alpha} = E^{\alpha-1} \cup U_{\alpha-1}$, $R^{\alpha} = R^{\alpha}_0 \cup R^{\alpha-1} \cup$ {constraints between events in $U_{\alpha-1}$} $\cup$ {constraints between events in $U_{\alpha-2}$ and events in $U_{\alpha-1}$}, where $R^{\alpha}_0 = R^{\alpha}_{01} \cup R^{\alpha}_{02}$. R^{α}_{01} includes all user defined start up rules. Since there is a finite number of user given start up rules, there exists a K_s such that there is no user given start up rule to the events in U_{α} for all $\alpha > K_s$. R^{α}_{02} is constructed in the same way as R^1_{02} in G^1. Since G_m is strongly connected, after adding constraints to R^{α}_{02}, for $\alpha = 0, \ldots, K_c$, where $K_c = |\max_c(\varepsilon(c))|$, c is a cycle in G_m, there is a path from all events in $U_{\alpha-1}$ to *root*, for all $\alpha > K_c$.

We call the addition of U_{α}, $\alpha \geq 0$, and the related constraints to G^{α} as the addition of one unfolding to G^{α}. For all $\alpha > K$, where

$$K = \max(K_s, K_c), \tag{3.5}$$

adding one unfolding means adding a fixed number of events (n) and rule edges.

3. 4 Maximum Time Separation of Events - Problem Definition

The problem we address is as follows [42]: Given two events, s and e in E (s for start, e for end), and a separation β in the occurrence indices, determine the largest

δ and the smallest Δ such that

$$\forall\alpha \geq \max(0, \beta),\ \delta \leq \tau(e_{\alpha+\beta}) - \tau(s_{\alpha}) \leq \Delta \tag{3.6}$$

subject to all constraints in G

Finding an algorithm for the maximum time separation Δ is sufficient, because δ can be obtained by formulating it as the maximum separation problem $\forall\alpha \geq \max(0, -\beta)$, $-\Delta \leq \tau(s_{\alpha}) - \tau(e_{\alpha-(-\beta)}) \leq -\delta$.

Let Δ^{α} be the least value such that $\tau(e_{\alpha+\beta}) - \tau(s_{\alpha}) \leq \Delta^{\alpha}$ (we also use the notation $\Delta(s_{\alpha}, e_{\alpha+\beta})$ instead of Δ^{α} when the start and the end events need to be emphasized). We call Δ the maximum time separation from s to e, and Δ^{α} the time separation from s_{α} to $e_{\alpha+\beta}$, Δ^{α} is the maximum time separation from s_{α} to $e_{\alpha+\beta}$ satisfying all the constraints in G^{∞}. Without loss of generality we suppose that $\beta \geq 0$ (when $\beta < 0$, the situation is similar). Δ can then be expressed as

$$\Delta = \max(\Delta^{0}, \Delta^{1}, \Delta^{2}, \ldots) \tag{3.7}$$

The maximum time separation in constraint graphs as defined in Equation (3.6) can be applied directly to compatibility verification of HAAD specifications in the form of Loop over a leaf TD. For instance, Causality Condition 2 (Equation (2.8)) for all finite unfolded constraint graph G^{α}, $\alpha \geq 0$, can be expressed as follows:

$$\forall\alpha \geq 0,\ \tau(e_{v,\alpha}) - \tau(e_{u,\alpha}) \leq 0, \tag{3.8}$$

where $e_{u,\alpha}$ and $e_{v,\alpha}$ are the α-th occurrence of e_u and e_v respectively.

The maximum time separation problems can also be applied to check other safety timing properties on interfaces. In the example of the MC68360 processor specified in Figure 3.1 and Figure 3.2, there are several important properties that should be verified. For instance, there should not be any overlap on the activation and deactivation of ACK and DATA signals. I.e., the ACK and DATA signals have been deactivated by one device before it is activated by possibly a different device in the following read cycle. This corresponds to the requirement that:

$$\forall\, \alpha \geq 0,\ \tau(e_{10,\,\alpha}) \leq \tau(e_{11,\,\alpha+1}), \text{ and} \tag{3.9}$$
$$\forall\, \alpha \geq 0,\ \tau(e_{14,\,\alpha}) \leq \tau(e_{15,\,\alpha+1}) \tag{3.10}$$

where $e_{i,j}$ is the j-th occurrence of event e_i. We can check these requirements by performing a maximum separation analysis, determining the smallest $\Delta 1$ and $\Delta 2$ such that

$$\forall\, \alpha \geq 0,\ \tau(e_{11,\,\alpha}) - \tau(e_{10,\,\alpha+1}) \leq \Delta 1 \text{ and} \tag{3.11}$$
$$\forall\, \alpha \geq 0,\ \tau(e_{15,\,\alpha}) - \tau(e_{14,\,\alpha+1}) \leq \Delta 2. \tag{3.12}$$

If $\Delta 1 > 0$ or $\Delta 2 > 0$, then there may be a timing requirement violation.

From the definition in Section 3. 4, Δ^{α} is the time separation from s_{α} to $e_{\alpha+\beta}$ satisfying all the constraints in G^{∞}, which contains an infinite number of events and constraints. It is impossible to find the time separation in this infinite graph directly. We have to find a way to perform the calculation in a finite graph. Even though both the start event s_{α} and the end event $e_{\alpha+\beta}$ are in $G^{\alpha+\beta+1}$, the time separation Δ^{α} may depend on the occurrence times of the events in later unfoldings ($U_{\alpha+\beta+1}$, $U_{\alpha+\beta+2}$, ...) because of the linear combinations of constraints in G^{∞}. It may be impossible to calculate Δ^{α} in $G^{\alpha+\beta+1}$, but we shall show that Δ^{α} can be calculated in a finitely unfolded constraint graph $G^{\alpha+\beta+r}$ where r is an integer such that the time separation from s_{α} to $e_{\alpha+\beta}$ computed in $G^{\alpha+\beta+r}$ is the same as that computed in G^{∞}. The existence of such r will be proven and its value will be determined in Chapter 5.

We can thus calculate Δ in Equation (3.7) in two steps: In the first step, we calculate Δ^{α} for all $\alpha \leq K$, where K is as defined in Equation (3.5). This can be achieved by using an algorithm for computing the maximum separation in the finite unfolded constraint graphs $G^{\alpha+\beta+r}$. In the second step, we add unfoldings to $G^{K+\beta+r}$ to construct $G^{K+i+\beta+r}$, $i > 0$, and find the time separations Δ^{K+i}. As the newly added constraints repeat themselves in each unfolding, we can analyse the series Δ^{K+i}, $i > 0$, and deduce the maximum time separation Δ.

In the next Chapter, we develop and prove an algorithm for computing time separations in graphs with a finite number of events and constraints.

Chapter 4

Maximum Time Separation in Finite Linear-Plus-Latest Systems

As defined in Chapter 3, the maximum time separation in constraint graphs needs the information of time separation Δ^{α} from s_{α} to $e_{\alpha+\beta}$ for all $\alpha \geq 0$. We start first with computing one Δ^{α}. We consider in this chapter the time separations for finite graphs containing linear-plus-latest constraints. The algorithm developed here is a variation of that proposed by McMillan and Dill [62]. It finds time separations from one start event to all events in a directed graph while the original algorithm finds time separations among all events.

In Section 4.1 and 4.2, we develop and prove the correctness of the variation algorithm. We then give several examples in Section 4.3. Finally, we analyse the complexity of the algorithm in Section 4.4.

4.1 The Variation Algorithm

Given a directed connected graph $G^f = <E^f, R^f>$, which corresponds to a finite unfolded constraint graph G^{α} in Section 3. 3, with a finite number of vertices (events) E^f and a finite number of edges R^f. Each edge represents a timing constraint which can be one of the two types: precedence or concurrency as defined in Section 2.1. Events here cannot repeat and thus are not indexed. When more than one edge sink at the same event v in the graph, the occurrence time of v is determined in one of the three ways as defined in Section 3. 3, i.e., linear, max and mixed combinations. Suppose that *root* has been included in G^f and constraints from *root* to events in G^f have been added as in Section 3. 3 such that there is a path from every event to *root* in G^f_m, where G^f_m is constructed from G^f in the same way as G_m from G in Section 3. 3. Now, given a start event s and an end event e in G^f, we wish to calculate the smallest $\Delta(s, e)$ respecting all the constraints in G^f.

It is well known that, in linear-only finite graphs, $\Delta(s, e)$ is the shortest distance from s to e in G^f_m where G^f_m is derived from G^f as defined in Definition 3.7.

In max-only constraint graphs, $\Delta(s, e)$ can be calculated using the following two steps [42]:

Step 1: Construct G^f_m from G^f. For all events $v \in E^f$, calculate m(v) as:

$$m(v) = \begin{cases} sd(s, v) & \text{if } s \leadsto v \\ \infty & \text{if } s \not\leadsto v \end{cases} \tag{4.1}$$

where $s \leadsto v$ ($s \not\leadsto v$) denotes that there is (is not) a path from s to v in G^f_m, and sd(s, v) is the shortest distance from s to v in G^f_m.

Step 2: set $\Delta(s, \mathit{root}) = m(\mathit{root})$, and for all events $v \in E^f$, let $\Delta(s, v)$ be

$$\Delta(s, v) = \min(m(v), \max_{u \in preds(v)}(\Delta(s, u) + D_{uv})) \tag{4.2}$$

In linear-plus-latest constraint graphs, McMillan and Dill gave the following algorithm to calculate the maximum time separation $\Delta(u, v)$ in G^f in [62], where $u, v \in E^f$:

```
Step 1: for all u, v in $G^f$, let $s_{uv} = \min(B_{uv}, -b_{vu}, -d_{vu})$,
repeat
        Step 2: for all u, v, w in $G^f$, let $s_{uv} = \min(s_{uv}, s_{uw} + s_{wv})$,
        Step 3: for all u, v in $G^f$, let $s_{uv} = \min(s_{uv}, \max_{w \in preds(v)}(s_{uw} + D_{wv}))$,
until condition 1: for some u, $s_{uu} < 0$, or
```

condition 2: no change from the previous iteration.

The algorithm requires iterations on all events in the graph for both u and v. This makes it difficult to extend the algorithm to compute the maximum time separation in constraint graphs defined in Chapter 3, where our strategy is to compute on Δ^{α}, and then compute $\Delta^{\alpha+1}$, $\Delta^{\alpha+2}$, ..., incrementally while adding more unfoldings. With the algorithm in [62], whenever more unfoldings are added, iterations on all events in the new graph are inevitable. This prevents the reuse of computation results obtained when computing Δ^{α} to calculate $\Delta^{\alpha+1}$. In the following, we present a variation of the above algorithm which lets us compute time separation from one start event to any event in the graph. We will extend this variation algorithm to compute time separations in the infinite unfolded constraint graph G^{∞}.

To facilitate understanding of how the linear-only and the max-only algorithms are used to develop an algorithm for the linear-plus-latest constraint systems, we first rewrite the algorithm for the finite linear constraint graphs into a similar form as for the max only case:

Step 1: Construct G^f_m from G^f. For all events $v \in E^f$, calculate m(v) using Equation (4.1).

Step2: Set $\Delta(s, \mathit{root}) = m(\mathit{root})$, and for all events $v \in E^f$, define msources(v) as all source events of v in G^f_m, calculate $\Delta(s, v) = m(v) = \min_{u \in \mathrm{msources}(v)}(\Delta(s, u) + \mathrm{sd}(u, v))$. This can be written as follows:

$$\Delta(s, v) = \min(m(v), \min_{u \in \mathrm{msources}(v)}(\Delta(s, u) + \mathrm{sd}(u, v))) \tag{4.3}$$

We can now combine the algorithms for linear only and max only graphs to obtain the following algorithm for finite linear plus max constraint graphs.

Algorithm 4.1: Time separation of events in a finite constraint graph with linear-plus-latest constraints

Given a finite constraint graph $G^f = \langle E^f, R^f \rangle$, a start event s and an end event e, the time separation $\Delta(s, e)$ from s to e can be computed in the following three steps:

Step 1: Construct G^f_m from G^f. For all events $v \in E^f$, let m(v) be calculated as in Equation (4.1). For each vertex $v \in E^f$, let msources(v) be the set of all the source events of v in G^f_m, and let P and ~P be the sets of events reachable and not reachable from s in G^f_m, respectively.

Step 2: Set $\Delta(s, \mathit{root}) = m(\mathit{root})$, and for all $v \in P$, let

$$\Delta(s, v) = \min(m(v), \max_{u \in \mathrm{preds}(v)}(\Delta(s, u) + D_{uv})), \tag{4.4}$$

where preds(v) is the set of source events to v involved in a max combination in G^f.

Step 3: Initialize $\Delta(s, v) = \infty$ for all $v \in {\sim}P$, repeat computing $\Delta(s, v)$ as in Equation (4.5) for all $v \in {\sim}P$, until either

a) there is a $v \in P$ such that $\min_{u \in \mathrm{msources}(v)}(\Delta(s, u) + sd(u, v)) < m(v)$ does not hold (report inconsistency of the constraint graph) or

b) no change in $\Delta(s, v)$ from the preceding iteration in which case report the maximum time separation $\Delta(s, v)$.

$$\Delta(s, v) = \min(m(v), \min_{u \in \mathrm{msources}(v)}(\Delta(s, u) + sd(u, v)), \max_{u \in \mathrm{preds}(v)}(\Delta(s, u) + D_{uv})), \tag{4.5}$$

where sd(u, v) is the shortest distance from u to v in G^f_m.

Informally, the algorithm works as follows:

In Step 1, for every $v \in P$, m(v) determines the latest occurrence time when v can occur after s without considering the upper bounds of constraints involved in max combinations. For $v \in \sim P$, m(v) is set to ∞, meaning that there is no constraint to satisfy without considering these upper bounds. sd(u, v) is the latest occurrence time of v after the source u considering all path(s) from u to v except again the upper bounds of the max constraints. In Steps 2 and 3, for all events v, the occurrence time is restricted by considering the upper bounds of the max constraints, while still satisfying all the other constraints. As $\Delta(s, v)$ in Equation (4.5) depends on $\Delta(s, u)$ which is unknown at the beginning, iterations in Step 3 is necessary.

In the following section, we prove the correctness of Algorithm 4.1.

4.2 Proof of the Algorithm 4.1

Theorem 4.1: If the constraint system represented by G^f is consistent, then $\Delta(s, v)$ expressed in Equations (4.4) for $v \in P$ and for $v \in \sim P$ is an achievable upper bound on the time separation from s to v.

Proof:

The proof consists of three parts. First, for any consistent timing assignment τ satisfying constraints in G^f in the form of Relations (2.2) and (2.3), we prove that $\tau(v) - \tau(s) \le \Delta(s, v)$, where $\Delta(s, v)$ is given in Equation (4.5). This relation also holds for all $v \in P$ if $\Delta(s, v)$ is as expressed in Equation (4.4). These are proven in Lemma 4.2. Second, we show in Lemma 4.3 that the class of timing assignments satisfying $\tau(v) = \Delta(s, v) + \tau(s)$, where $\Delta(s, v)$ is given in Equation (4.5), corresponds to legal executions, and thus $\tau(v_k) - \tau(s) = \Delta(s, v)$ is an achievable upper bound for all $v \in E^f$. Then, we prove in Lemma 4.4 that if the constraint system represented by G^f is consistent, then $\Delta(s, v)$ as expressed in Equation (4.4) is also an achievable upper bound on the time separation from s to v for all $v \in P$. Therefore, $\Delta(s, v)$ as expressed in Equation (4.4) for $v \in P$ and in Equation (4.5) for $v \in \sim P$ is an achievable upper bound on the time separation from s to v.

Since Equation (4.5) is a strictly monotonous function and there are only a finite number of constraints in a constraint graph, Algorithm 4.1will terminate.

If we use Equation (4.5) to calculate $\Delta(s, v)$ for all events in G^f, then inconsistency of G^f is detected during iterations when $\Delta(s, s) < 0$ as in [62]. We use Equation (4.4) instead of Equation (4.5) to calculate $\Delta(s, v)$ for $v \in P$. This may lead to some kind of inconsistency that could be undetected during the iterations in Step 3 of the variation algorithm. However, from Lemma 4.3 and Lemma 4.4, we have that $m(v) \leq \min_{u \in msources(v)}(\Delta(s, u) + sd(u, v))$ for all $v \in P$ if the system is consistent. Therefore, we can use this relation to detect the inconsistency of the constraint system.

❑

To prove Lemmas 4.2, 4.3 and 4.4, we need the following Lemma 4.1.

Lemma 4.1: For any consistent timing assignment τ satisfying the constraints in G^f, the following relation holds for all v in G^f:

$$\tau(v) - \tau(s) \leq m(v) \qquad (4.6)$$

Proof:

If $v \in P$, then since m(v) is the shortest distance from s to v in G^f_m, we have $\tau(v) - \tau(s) \le \text{length}(\rho)$, where ρ is a path from the start event s to v in G^f_m. This relation holds for every path ρ. It follows $\tau(v) - \tau(s) \le \min(\text{length}(\rho))$. However, $\min(\text{length}(\rho)) = sd(s, v) = m(v)$ in Equation (4.1). Therefore, $\tau(v) - \tau(s) \le m(v)$ holds for all $v \in P$.

Else if $v \in \sim P$, then $m(v) = \infty$ and Relation (4.6) holds. ❑

Lemma 4.2: For any consistent timing assignment τ satisfying constraints in G^f, the following relation holds:

$$\tau(v) - \tau(s) \le \Delta(s, v), \tag{4.7}$$

where $\Delta(s, v)$ is computed according to Equation (4.5). Relation (4.7) also holds for all $v \in P$ if $\Delta(s, v)$ is expressed as in Equation (4.4).

Proof:

We first prove that Relation (4.7) holds for $\Delta(s, v)$ as expressed in Equation (4.5) by induction on events in G^f.

Basis: $v = root$, $\Delta(s, root) = m(root)$, by Lemma 4.1, $\tau(root) - \tau(s) \le m(root)$ is true, hence, Relation (4.7) holds.

The induction hypothesis is that Relation (4.7) holds for the source event of a constraint from u to v in G^f, i.e., $\tau(u) - \tau(s) \le \Delta(s, u)$ holds. We prove $\tau(v) - \tau(s) \le \Delta(s, v)$ by showing that for any consistent timing assignment, $\tau(v) - \tau(s)$ is not grater than each argument of the min function of Equation (4.5).

For the first argument, $\tau(v) - \tau(s) \leq m(v)$ holds due to Lemma 4.1.

For the second argument, since sd(u, v) is the shortest distance from u to v in G^f_m, we have $\tau(v) - \tau(u) \leq sd(u, v)$ for every path from u to v in G^f_m. Adding this relation to the induction hypothesis yields $\tau(v) - \tau(s) \leq \Delta(s, u) + sd(u, v)$. Hence, $\tau(v) - \tau(s) \leq \min_{u \in msources(v)}(\Delta(s, u) + sd(u, v))$ holds.

For the third argument of the min function in Equation (4.5), from Relation (2.3) we have $\tau(v) \leq \max_{u \in preds(v)}(\tau(u) + D_{uv}))$. Subtract $\tau(s)$ from both sides to get $\tau(v) - \tau(s) \leq \max_{u \in preds(v)}(\tau(u) - \tau(s) + D_{uv}))$. However, $\tau(u) - \tau(s) \leq \Delta(s, u)$, and thus $\tau(v) - \tau(s) \leq \max_{u \in preds(v)}(\Delta(s, u) + D_{uv}))$.

Therefore, Relation (4.7) holds for v. As G^f_m is a connected graph, starting from *root*, Relation (4.7) holds for all events in G^f_m (and also for all events in G^f).

If $\Delta(s, v)$ is as expressed in Equation (4.4), Relation (4.7) still holds, because $\Delta(s, v)$ in Equation (4.5) contains one more term in the min function than that in Equation (4.4). Therefore, it can not be grater than $\Delta(s, v)$ as expressed in Equation (4.4). ❑

Lemma 4.3: For all $v \in E^f$, the class of timing assignments

$$\tau(v) = \Delta(s, v) + \tau(s), \tag{4.8}$$

where $\Delta(s, v)$ is as expresses in Equation (4.5), corresponds to legal executions, i.e., the timing assignments of Equation (4.8) satisfy all the timing constraints in G^f.

Proof:

By substituting $\Delta(s, v)$ in Equation (4.5) to Equation (4.8), and by substituting $\tau(u) - \tau(s)$ by $\Delta(s, u)$, we get the timing assignments to v as follows:

$$\tau(v) = \min(m(v) + \tau(s), \min_{u \in msources(v)}(\tau(u) + sd(u, v)),$$

$$\max_{u \in preds(v)}(\tau(u) + D_{uv})). \tag{4.9}$$

Obviously, $\tau(v) \le \max_{u \in preds(v)}(\tau(u) + D_{uv})$ holds, i.e., the right-hand side of Relation (2.3) is satisfied.

From Equation (4.9), we have $\tau(v) \le \min_{u \in msources(v)}(\tau(u) + sd(u, v))$. As sources(v) $\subseteq$ msources(v), we have $\tau(v) \le \min_{u \in sources(v)}(\tau(u) + sd(u, v))$. However, $sd(u, v) \le B_{uv}$ for every $u \in$ sources(v), therefore, $\tau(v) \le \min_{u \in sources(v)}(\tau(u) + B_{uv})$ holds, i.e., the right-hand side of Relation (2.2) holds.

For every $u \in$ preds(v), $v \in$ msources(u) is true in G^f_m. When calculating $\Delta(s, u)$ using Equation (4.5), we have

$$\Delta(s, u) \le \min_{w \in msources(u)}(\Delta(s, w) + sd(w, u)) \le \Delta(s, v) + sd(v, u).$$

However $sd(v, u) \le -d_{uv}$, and we have $\Delta(s, u) \le \Delta(s, v) - d_{uv}$, i.e., $\Delta(s, v) \ge \Delta(s, u) + d_{uv}$. Adding $\tau(s)$ to both sides, and applying Equation (4.8) to u and v, we get $\tau(v) \ge \tau(u) + d_{uv}$. As this relation holds for every $u \in$ preds(v), we get $\tau(v) \ge \max_{u \in preds(v)}(\tau(u) + d_{uv})$, i.e., the left-hand side of Relation (2.3) holds.

For every $u \in$ sources(v), we have $v \in$ msources(u) is true in G^f_m. When calculating $\Delta(s, u)$ using Equation (4.5), we have

$$\Delta(s, u) \le \min_{w \in msources(u)}(\Delta(s, w) + sd(w, u)) \le \Delta(s, v) + sd(v, u).$$

However since $sd(v, u) \leq - b_{uv}$, we have $\Delta(s, u) \leq \Delta(s, v) - b_{uv}$, i.e., $\Delta(s, v) \geq \Delta(s, u) + b_{uv}$. By adding $\tau(s)$ to both sides, and applying Equation (4.8) to u and v, we get $\tau(v) \geq \tau(u) + b_{uv}$. This relation holds for every $u \in sources(v)$, i.e., the left-hand side of Relation (2.2) holds.

It follows that Relations (2.2) and (2.3) are satisfied. ❑

Lemma 4.4: If the constraint system represented by G^f is consistent, then for all $v \in P$ the class of timing assignments given by Equation (4.8) where $\Delta(s, v)$ is as expresses in Equation (4.4) corresponds to legal executions, i.e., the timing assignments of Equation (4.8) satisfy all the timing constraints in G^f.

Proof:

By substituting $\Delta(s, v)$ from Equation (4.4) to Equation (4.8) and by substituting $\tau(u) - \tau(s)$ by $\Delta(s, u)$, we get the following timing assignment to v:

$$\tau(v) = \min(m(v) + \tau(s), \max_{u \in preds(v)}(\tau(u) + D_{uv})) \qquad (4.10)$$

Obviously, $\tau(v) \leq \max_{u \in preds(v)}(\tau(u) + D_{uv})$ holds, i.e., the right-hand side of Relation (2.3) is satisfied.

Applying Equation (4.10) to u yields

$$\tau(u) = \min(m(u) + \tau(s), \max_{w \in preds(u)}(\tau(w) + D_{wu})), \qquad (4.11)$$

We prove the right-hand side of Relation (2.2) by showing that $\tau(v) - B_{uv}$ is not grater than each argument of the min function of Equation (4.11). If $\tau(u)$ takes the value $m(u) + \tau(s)$, then for every $u \in sources(v)$, $m(v) \leq m(u) + B_{uv}$ holds. Adding $\tau(s)$ to both sides yields $m(v) + \tau(s) \le \tau(u) + B_{uv}$. However, $\tau(v) \le m(v) + \tau(s)$ due to Lemma 4.1, therefore, $\tau(v) \le \tau(u) + B_{uv}$ holds. Else if $\tau(u) = \max_{w \in preds(u)}(\tau(w) + D_{wu})$, then $\tau(v) \le \max_{w \in sources(u)}(\tau(w) + D_{wu}) + B_{uv}$ has to hold, otherwise, the constraint system represented by G^f is inconsistent (meaning that $\tau(u)$ takes the maximum possible value and we still cannot find an assignment to $\tau(v)$ that satisfies the relation $\tau(v) \le \tau(v) + B_{uv}$). Therefore, the right-hand side of Relation (2.2) is satisfied if G^f is consistent.

We now prove the left-hand side of Relation (2.3) by showing that $\max_{u \in preds(v)}(\tau(u) + d_{uv})$ is not grater than each argument of the min function in Equation (4.10). For the first argument, it can be proven as follows: for every $u \in preds(v)$, $m(u) \le m(v) - d_{uv}$ holds, i.e., $m(u) + d_{uv} \le m(v)$, hence $\max_{u \in preds(v)}(m(u) + d_{uv}) \le m(v)$. Adding $\tau(s)$ to both sides yields $\max_{u \in preds(v)}(m(u) + \tau(s) + d_{uv}) \le m(v)+ \tau(s)$. However, by Lemma 4.1, $\tau(u) \le m(u) + \tau(s)$, hence $\max_{u \in preds(v)}(\tau(u) + d_{uv}) \le m(v)+ \tau(s)$ holds. For the second argument, as $d_{uv} \le D_{uv}$, we immediately get that $\max_{u \in preds(v)}(\Delta(s, u) + \tau(s) + d_{uv})) \le \max_u (\Delta(s, u) + \tau(s) + D_{uv})$. Hence, the left-hand side of Relation (2.3) is satisfied, i.e., $\max_{u \in preds(v)}(\tau(u) + d_{uv}) \le \tau(v)$.

For the left-hand side of Relation (2.2), we show that, if the constraint system represented by G^f is consistent, then for every $u \in sources(v)$, $\tau(u) + b_{uv}$ is not grater than each argument of the min function in Equation (4.10). If $\tau(v)$ takes the value $m(v) + \tau(s)$, then for every $u \in sources(v)$, $m(u) \le m(v) - b_{uv}$ holds, i.e., $m(u) + b_{uv} \le m(v)$, $m(u) + \tau(s) + b_{uv} \le \tau(v)$. However, $\tau(u) \le m(u) + \tau(s)$ due to Lemma 4.1, therefore, $\tau(u) + b_{uv} \le \tau(v)$ is satisfied. Else if $\tau(v) = \max_{u \in preds(v)}(\tau(u) + D_{uv})$, then $\max_{u \in sources(v)}(\tau(u) + D_{uv}) \ge \tau(u) + b_{uv}$ has to be satisfied, otherwise, the constraint system represented by G^f is inconsistent (meaning that $\tau(v)$ takes the maximum possible value and we still cannot find an assignment to $\tau(u)$ that satisfies the relation $\tau(v) \ge \tau(u) + b_{uv}$). Therefore, the left hand side of Relation (2.2) is satisfied if G^f is consistent.

It follows that Relations (2.2) and (2.3) are satisfied if G^f is consistent. ❑

In the McMillan and Dill's algorithm [62], inconsistency of the constraint system can be detected during the iterations when $\Delta(u, u) < 0$. In the variation algorithm, instead of Equation (4.5) we use Equation (4.4) to calculate $\Delta(s, v)$ for all $v \in P$. This may leave some kind of inconsistency undetected by $\Delta(s, s) < 0$ during the iterations in Step 3. However, from Lemma 4.3 and Lemma 4.4, we have that $m(v) \leq \min_{u \in msources(v)}(\Delta(s, u) + sd(u, v))$ for all $v \in P$ if the system is consistent. Therefore, we can use this relation to detect the inconsistency, as stated in the following Corollary to Theorem 4.1.

Corollary 4.1: Inconsistency of the constraint graph containing max and linear constraints can be detected by a) the presence of a negative cycle in G^f_m, or b) there exists a $v \in P$ such that $\min_{u \in msources(v)}(\Delta(s, u) + sd(u, v)) < m(v)$.

4.3 Examples

Example 4.1: Consider a constraint graph G^f as shown in Figure 4.1, Let the start event be b, and the end event be g, i.e., we wish to calculate $\Delta(b, g)$.

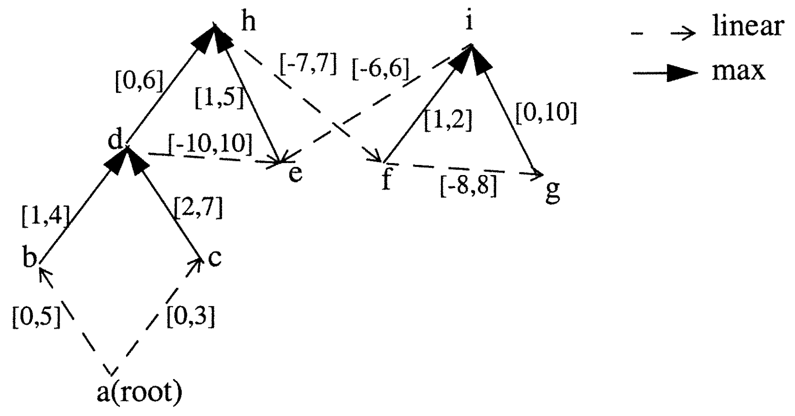


Figure 4.1 A constraint graph G^f for Example

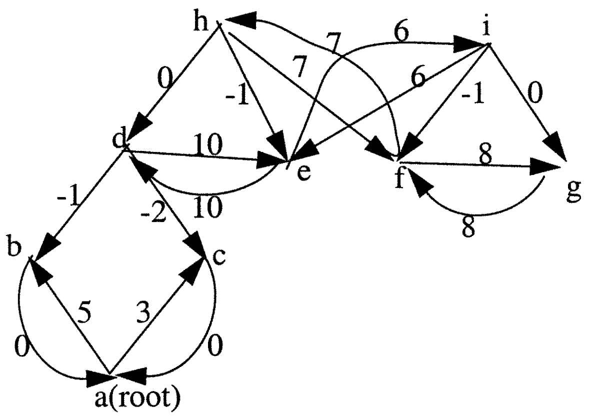


Figure 4.2 G^f_m for constraint graph G^f in Figure 4.1

Step 1:

m(b) = 0, m(a) = 0, m(c) = 3, m(d) = m(e) = m(f) = m(g) = m(h) = m(i) = ∞, P = {a, c}, ~P = {d, e, f, g, h, i}, msources(a) = {b, c}, msources(b) = (a, d), msources(c) = {d, a}, msources(d) = (h, e), msources(e) = (d, h, i), msources(f) = (h, i, g), msources(g) = (f, i), msources(h) = (f), msources(i) = (e).

Step 2: Δ(b, a) = m(a) = 0, Δ(b, c) = m(c) = 3.

Step 3: Initialize $\Delta(b, d) = \Delta(b, e) = \Delta(b, f) = \Delta(b, g) = \Delta(b, h) = \Delta(b, i) = \infty$.

First iteration:

$\Delta(b, d) = \min(\infty, \min(\infty+sd(e,d), \infty+sd(e,d)), \max(0+4, 3+7)) = 10$,

$\Delta(b, e) = \min(\infty, \min(10+10, \infty-1, \infty+5)) = 20$,

$\Delta(b, h) = \min(\infty, \infty+sd(f,h), \max(10+6, 20+5)) = 25$,

$\Delta(b, f) = \min(\infty, \min(25+4, \infty-1, \infty+8)) = 29$,

$\Delta(b, g) = \min(\infty, \min(29+8, \infty+0)) = 37$,

$\Delta(b, i) = \min(\infty, 20+6, \max(29+2, 37+10)) = 26$.

No inconsistency is detected.

Second iteration:

$\Delta(b, d) = \min(\infty, \min(20+10, 25+0), \max(0+4, 3+7)) = 10$,

$\Delta(b, e) = \min(\infty, \min(10+10, 25-1, 26+5)) = 20$,

$\Delta(b, h) = \min(\infty, 29+7, \max(10+6, 20+5)) = 25$,

$\Delta(b, f) = \min(\infty, \min(25+4, 26-1, 37+8)) = 25$,

$\Delta(b, g) = \min(\infty, \min(25+8, 26+0)) = 26$,

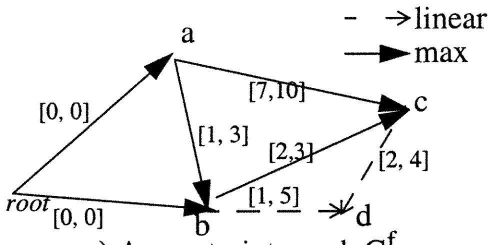
$\Delta(b, i) = \min(\infty, 20+6, \max(25+2, 26+10)) = 26$.

No inconsistency is detected.

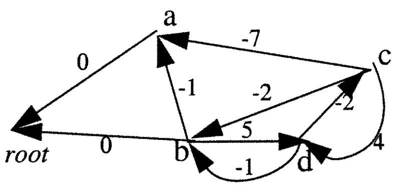
Third iteration, a fixed point is reached.

Consistency check: $\Delta(b, a) + sd(a, c) = 0 + 3 = 3 = m(c)$, $\Delta(b, b) = \min(0, \min(0+5, 10-1)) = 0 = m(b)$, therefore, the constraint system is consistent.

Example 4.2: An inconsistent constrain graph is shown in Figure 4.3. The start event is a and the end event is d, i.e., we wish to compute $\Delta(a, d)$.



a) A constraint graph G^f



b) corresponding G^f_m

Figure 4.3 An inconsistent constraint graph

Step 1: G^f_m is constructed as shown in Figure 4.3b. The m values are: m(a) = 0, m(*root*) = 0, m(b) = m(c) = m(d) = ∞. P = {*root*, a}, ~P = {b, c, d}.

Step 2: Δ(a, root) = m(root) = 0,

Step 3: Initialize Δ(a, b) = Δ(a, c) = Δ(a, d) = ∞.

First iteration:

Δ(a, b) = min(∞, min(∞-2, ∞-4), max(0 + 0, 0 + 3)) = 3,
Δ(a, c) = min(∞, ∞-2, max(0 + 10, 3 + 3)) = 10,
Δ(a, d) = min(∞, min(3 + 5, 10 + 4)) = 8.
Δ(a, a) = min(0, min(3-4, 10-7)) = -1 < 0, therefore, G^f is inconsistent.

In this example, there is no negative cycle in G^f_m, but the constraint system is not consistent, because to satisfy the constraints from a to c and then from c to d, $\tau(d) - \tau(a) \geq 9$ must hold, on the other hand, to satisfy the constraints from a to b and then from b to d, $\tau(d) - \tau(a) \leq 8$ has to hold.

4.4 Complexity of the Variation Algorithm

Compared to the algorithm in [62], we calculate $\Delta(s, v)$ for $v \in P$ in one step using Equation (4.5) instead of doing an iteration. In the worst case, no event in $G^f{}_m$ is reachable from the start point except the start point itself. In that case the complexity of the variation algorithm is comparable to that in [62], it is pseudo polynomial in $O(n^2\Sigma_{ij}s_{ij})$, where n is the number of events in G^f, $\Sigma_{ij}s_{ij}$ is the sum of all initial timing separations (timing bounds in Relations (2.2) and (2.3)). In the best case, all events are reachable from the start point, in which case the complexity of the algorithm is polynomial.

Chapter 5

Maximum Time Separation Algorithms in Cyclic Systems

In this Chapter, we give algorithms of computing the maximum time separation of events in cyclic systems. The chapter consists of 2 sections dealing with cyclic systems containing linear-only and linear-plus-latest constraints, respectively.

5.1 Cyclic Linear-Only Systems

In this section, we develop an algorithm of computing the maximum time separation in a cyclic system with linear-only constraints. We first show that if the constraint system represented by G is consistent, then the time separation from s_α to $e_{\alpha+\beta}$ computed in a finite unfolded constraint graph $G^{\alpha+\beta+r}$ is the same as computed in the infinite unfolded constraint graph G^∞, where r is an integer. We then give an upper bound of r. The upper bound can be determined by the number of events in the constraint graph. At last, we obtain an algorithm with complexity in $O(n^9)$ for computing the maximum time separation between events in linear-only cyclic systems, where n is the number of events in G.

From the definition in Section 3. 4, the time separation Δ^α from s_α to $e_{\alpha+\beta}$ in G^∞ is the shortest distance from s_α to $e_{\alpha+\beta}$ in the directed graph G_m^∞, where G_m^∞ is derived from G^∞ in the same way as G_m is derived from G (Definition 3.7). Although both the start event s_α and the end event $e_{\alpha+\beta}$ are in $G^{\alpha+\beta+1}$, the shortest path from s_α to $e_{\alpha+\beta}$ in G_m^∞ may pass through events in $U_{\alpha+\beta+1}$[1], $U_{\alpha+\beta+2}$, ... In other words, we may not compute Δ^α by considering only constraints in $G^{\alpha+\beta+1}$. The following theorem states that the shortest path from s_α to $e_{\alpha+\beta}$ in a consistent G_m^∞ cannot pass

1. As defined in Equation (3.4), $U_\alpha = \{v_\alpha \mid v \in E\}$, $\alpha \geq 0$.

through any event in $U_{\alpha+\beta+r+1}$, $U_{\alpha+\beta+r+2}$, …, where r is an integer constant, i.e., Δ^{α} can be computed in $G_m^{\alpha+\beta+r}$.

Theorem 5.1: If the constraint system represented by G^{∞} is consistent, then $\forall\ \alpha > K$, where K is as defined in Equation (3.5), the shortest path between any two events u_{α} and v_{α}, $u, v \in E$, will not pass through any event in $U_{\alpha+r+1}$ for $r \geq P(n, 2) + 1$, where $P(n, 2) = n \cdot (n-1)$ is the 2 permutation of n, and n is the number of events in G.

Proof: By contradiction.

Suppose that the shortest path from u_{α} to v_{α} passes through at least one event in $U_{\alpha+r+1}$, then the shortest path passes through at least two events in each of $U_{\alpha+1}$, $U_{\alpha+2}$, …, $U_{\alpha+r}$. As there are n events in U_{α} for all $\alpha \geq 0$, and r is greater than P(n, 2), there exist some $u', v' \in E$, $p, q \in [\alpha+1, \ldots, \alpha+r]$, $p \neq q$, such that u'_p, v'_p, u'_q, and v'_q lie on the shortest path, as shown in Figure 5.1.

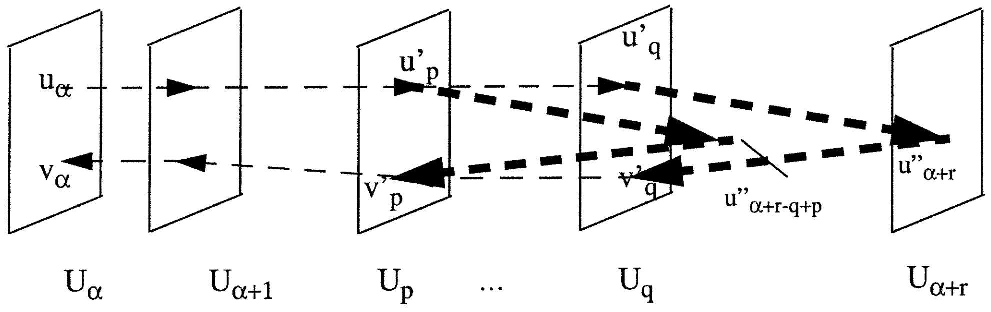


Figure 5.1 The shortest path from u_{α} to v_{α}

As $\alpha > K$, the constraints between events in $U_{\alpha+i}$ and those between events in $U_{\alpha+i}$ and $U_{\alpha+i+1}$ are the same for all $i > 0$. Hence, as shown in Figure 5.1, corresponding to the path from u'_q to v'_q passing through $u''_{\alpha+r}$ (which is on the shortest path from

u_α to v_α), there is a path from u'_p to v'_p passing through $u''_{\alpha+r-q+p}$. The lengths of these two paths are identical. As the shortest path from u'_p to v'_p passes through u'_q to v'_q and $u''_{\alpha+r}$, we have that $d = d(u'_p, u'_q) + d(v'_q, v'_p) < 0$, where $d(u'_p, u'_q)$ and $d(v'_q, v'_p)$ are the distances from u'_p to u'_q and from v'_q to v'_p, respectively, along the shortest path from u'_p to v'_p. Consequently, when further (q - p) unfoldings are added, the shortest distance from u'_p to v'_p will decrease since $d < 0$. That is, when more and more unfoldings are added, the shortest distance from u'_p to v'_p will be approaching $-\infty$. On the other hand, from the well-formedness Condition 1, G_m is strongly connected, this means that there is a path from u'_p to v'_p with a finite distance, hence there is a negative cycle containing events u'_p, u'_q, $u'_{p+2(q-p)}$, ..., $v'_{p+2(q-p)}$, v'_q, v'_p in the G_m^∞. Therefore, the constraint system represented by G_m^∞ (and thus G^∞) would be inconsistent. ❑

Theorem 5.1 gives a condition which allows us to determine whether the constraint system G^∞ is consistent.

Corollary 5.1: If there exist some $\alpha > K$, $0 < i < r$, and $u, v \in E$, where K is defined as in Equation (3.5), such that the shortest path between u_α and v_α in $G_m^{\alpha+r}$ passes through $u_{\alpha+i}$ and $v_{\alpha+i}$, where $r = P(n, 2) +1$, $P(n, 2)$ is the 2 permutation of n, then G^∞ is inconsistent.

Consequently, if G^∞ is consistent and $\alpha > K$, then the shortest distance from s_α to $e_{\alpha+\beta}$ calculated in $G_m^{\alpha+\beta+r}$ is the same as that calculated in G_m^∞.

Symmetrically to Theorem 5.1, we also have that if G is consistent, then $\forall\ \alpha > K + r$, the shortest path between two events u_α to v_α cannot pass through any event in $U_{\alpha-r-1}$. In other words, the time separation from u_α to v_α is determined by constraints in $U_{\alpha-r}$, $U_{\alpha-r+1}$, ..., U_α, ..., $U_{\alpha+r}$, for $\alpha > K + r$, otherwise there is a negative cycle in G_m^∞. Since the constraints in $U_{\alpha-r}$, $U_{\alpha-r+1}$, ..., U_α, ..., ..., $U_{\alpha+r}$ are the same for all $\alpha > K + r$, we have that the time separation from u_α to v_α are the same for all $\alpha > K + r$. Similary, the time separation from u_α to $v_{\alpha+\beta}$ is determined by constraints between events in $U_{\alpha-r}$, $U_{\alpha-r+1}$, ..., U_α, ..., $U_{\alpha+\beta+r}$, for $\alpha > K + r$, otherwise there is a negative cycle in G_m^∞. Since the constraints in $U_{\alpha-r}$, $U_{\alpha-r+1}$, ..., U_α, ..., ..., $U_{\alpha+\beta+r}$ are the same for all $\alpha > K + r$, we have that the time separation from u_α to $v_{\alpha+\beta}$ are the same for all $\alpha > K + r$. We thus can compute time separation from u_α to $v_{\alpha+\beta}$ in $G^{\alpha+\beta+r}$.

To find the maximum time separation Δ in the infinite constraint graph G^∞, we first calculate the shortest distance Δ^α from s_α to $e_{\alpha+\beta}$ in $G^{\alpha+\beta+r}$ where $r = P(n, 2) + 1$ for $\alpha \leq K$ (K is as in Equation (3.5)). Second, we add unfolding $U_{K+i+\beta+r}$ to $G^{K+i+\beta+r}$, $0 < i \leq r$, and calculate Δ^{K+i} as the shortest distance from s_{K+i} to $e_{K+i+\beta}$ in $G^{K+i+\beta+r}$. The maximum time separation is then $\Delta = \max(\Delta^0, \Delta^1, \ldots, \Delta^{K+r}, \Delta^{K+r+1}, \ldots) = \max(\Delta^0, \Delta^1, \ldots, \Delta^{K+r})$.

Algorithm 5.1: Maximum time separation in cyclic systems with linear-only constraints

Step 1: Construct $G_m^{K+\beta+r}$ from $G^{K+\beta+r}$, if there is a negative cycle in it, then stop and report inconsistency of G^∞. Else if there is a pair of events u_p and v_p, such that the shortest path from u_p to v_p passes through another pair of events, u_q and v_q, $p \neq q$, $p, q > K$, then stop and report inconsistency of G^∞. Else for $\alpha = 0, \ldots, K+r$, find the shortest distance Δ^α from s_α to $e_{\alpha+\beta}$ in $G_m^{\alpha+\beta+r}$.

Step 2: $\Delta = \max(\Delta^0, \Delta^1, \ldots, \Delta^{K+r})$.

The complexity of Algorithm 2 can be analysed as follows:

In Step 1, the number of events in $G^{K+\beta+r}$ is $(K + r + \beta) \cdot n + 1$. The complexity of detecting a negative cycle in a graph is in the order of the cube of the number of events in $G^{K+\beta+r}$, i.e., it is in $O(((K + r + \beta) \cdot n + 1)^3)$. The complexity of detecting a negative cycle is thus in $O(n^9)$, since $r = P(n, 2) + 1$ is in $O(n^2)$ and K is a constant. The number of pairs of events in U_{K+1} is in $O(n^2)$, the complexity to find the shortest path between one pair of events is in the square of the number of events in the graph, hence the time needed to check the consistency of G^{∞} is in $O(n^2 \cdot ((K + r + \beta) \cdot n + 1)^2)$, because the number of events in $G^{K+\beta+r}$ is $(K + r + \beta) \cdot n + 1$. The complexity to check the consistency of G^{∞} is thus in $O(n^8)$. The complexity to find $\Delta^0, \Delta^1, \ldots, \Delta^{K+r}$ is $(K + r + 1) \cdot (K + 2r + \beta)^2$, which is in $O(n^6)$. The time needed in Step 2 is in $O(K+r)$ where K is a constant and r is in $O(n^2)$. It follows that the complexity of Algorithm 5.1 is in $O(n^9)$.

5.2 Cyclic Linear-plus-Latest Systems

In this section, we address the maximum time separation problem in cyclic systems with linear-plus-latest constraints. Similar to the linear-only systems, we begin with one Δ^{α}, the time separation from s_{α} to $e_{\alpha+\beta}$ satisfying all the constraints in the infinite unfolded constraint graph G^{∞}.

Even though both the start and the end events are in $G^{\alpha+\beta+1}$, the time separation Δ^{α} may depend on the occurrence times of the events in later unfoldings ($U_{\alpha+\beta+1}$, $U_{\alpha+\beta+2}$, ...) because of the presence of linear constraints in G^{∞}. We thus may not be able to compute Δ^{α} in $G^{\alpha+\beta+1}$. In this section, we show that Δ^{α} can be calculated in a finitely unfolded constraint graph $G^{\alpha+\beta+r}$ where r is an integer such that the time separation computed in $G^{\alpha+\beta+r}$ is the same as that computed in G^{∞}. In Section

5.2.1, we prove that such r exists and that it takes the same value P(n, 2) + 1 as in linear-only systems, where n is the number of events in G. In Section 5.2.2, Algorithm 4.1 is extended to compute $\Delta^{\alpha+i}$ (i > 0) in $G^{\alpha+i+\beta+r}$. We show that the time separation will eventually become a periodic function of i. We give a condition to detect this situation. Thereafter further unfoldings are unnecessary for computing the maximum time separation. We show that the number of unfoldings needed for the time separations to become periodic depends on the delay values in the specification. We propose an algorithm for computing the maximum time separations by unfolding the constraint graph step by step while checking whether the condition is satisfied until the separation exhibits a periodic behaviour.

5.2.1 Algorithm to Compute Δ^{α}

Δ^{α} can be calculated by applying Algorithm 4.1 in Section 4.1 to $G^{\alpha+\beta+r}$ due to the following theorem:

Theorem 5.2: If the constraint system G^{∞} is consistent, then $\forall\ \alpha \geq K$, Δ^{α} can be calculated in $G^{\alpha+\beta+r}$, where n is the number of events in G, K is defined as in Equation (3.5), r = P(n, 2) +1, and P(n, 2) is the 2 permutation of n.

Proof:

All the latest constraints are precedence, hence the occurrence times of the sink events of latest constraints are determined by their predecessors. However, the occurrence times of the predecessors do not depend on these max constraints. In other words, the max constraints in the later unfoldings cannot influence the occurrence times of events in earlier unfoldings. The linear constraints in later unfoldings are considered when the m values are calculated. If G^{∞} is consistent, then G_m^{∞} is consistent too. By applying Theorem 5.1 to G_m^{∞}, we get that $\forall\ \alpha \geq K$, the shortest path between any two events u_α and v_α, where $u, v \in E$, will not pass through any event in $U_{\alpha+r+1}$, i.e., the m-values in Step 1 of the Algorithm 4.1 can be calculated in $G_m^{\alpha+\beta+r}$. Therefore, Δ^α can be calculated in $G^{\alpha+\beta+r}$ if the constraint system represented by G^∞ is consistent. ❑

Algorithm 5.2: Δ^α in linear-plus-latest constraint graphs

Step1: Check the consistency of G_m^∞ based on Corollary 5.1. If G_m^∞ is inconsistent, then stop, otherwise, go to Step 2.

Step 2: Construct $G_m^{\alpha+\beta+r}$ from $G^{\alpha+\beta+r}$, and set $m(s_\alpha) = 0$. For all events $u \in E$, $0 \leq i < \alpha + \beta$, calculate the m-values using Equation (4.1).

Step 3: Set $\Delta(s_\alpha, root) = m(root)$, initialize $\Delta(s_\alpha, v_i) = m(v_i)$ for all the events reachable from the start point s_α in $G_m^{\alpha+\beta+r}$, initialize $\Delta(s_\alpha, v_i) = \infty$ for all v not reachable from s_α.

Step 4: Compute $\Delta(s_\alpha, v_i)$, $v_i \in U_i$, repeatly using Equation (4.5), until either there is a v reachable from s_α such that $\min_{u \in msources(v)}(\Delta(s, u) + sd(u, v)) < m(v)$ in which case report inconsistency of the specification, or there is no change in $\Delta(s_\alpha, v_i)$ in which case report the maximum time separation is $\Delta^\alpha = \Delta(s_\alpha, e_{\alpha+\beta})$. ❑

Example 5.1: Consider a constraint graph as shown in Figure 5.2. We wish to calculate Δ^2 from a_2 to b_2. We have $n = 2$, $P(n, 2) + 1 = 3$, therefore, additional three unfoldings are needed to calculate Δ^2.

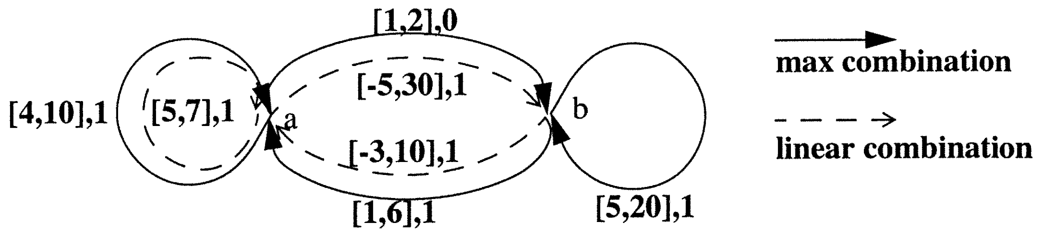


Figure 5.2 . The constraint graph G for Example 5.1

Step 1: Based on Corollary 5.1, the constraint system $G^{\infty}{}_{m}$ is consistent.

Step 2: Construct $G^{6}{}_{\mathrm{m}}$ as in Figure 5.3. The m-values are:

$m(a_2) = 0$, $m(b_2) = \min(7\text{-}1, \text{-}5\text{+}30) = 6$, $m(b_1) = \min(\text{-}1, 7\text{-}1\text{-}5) = \text{-}1$, $m(a_1) = \min(\text{-}5, \text{-}1\text{-}1) = \text{-}5$, $m(b_0) = \min(\text{-}1\text{-}5, \text{-}5\text{-}1) = \text{-}6$, $m(a_0) = \min(\text{-}5\text{-}5, \text{-}6\text{-}1, \text{-}1\text{-}5) = \text{-}10$, $m(\mathit{root}) = \min(0\text{-}10, 0 - 6) = \text{-}10$.

All events are reachable from the start point a_2.

Step 3: $\Delta(a_2, \mathit{root}) = m(\mathit{root}) = \text{-}10$, $\Delta(a_2, a_0) = m(a_0) = \text{-}10$, $\Delta(a_2, b_0) = m(b_0) = \text{-}6$, $\Delta(a_2, a_1) = m(a_1) = \text{-}5$, $\Delta(a_2, b_1) = m(b_1) = \text{-}1$, ...

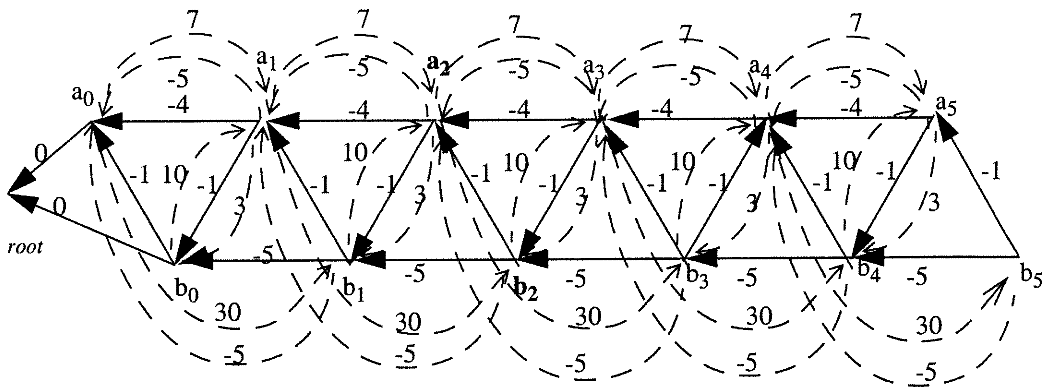


Figure 5.3 The graph G^6_m of constraint graph in Figure 5.2

Step 4: $\Delta(a_2, root) = m(root) = -10$, $\Delta(a_2, a_0) = \min(-10, -10 + 0) = -10$, $\Delta(a_2, b_0) = \min(-6, \max(-10 + 2, -10 + 0)) = -8$, $\Delta(a_2, a_1) = \min(-5, \max(-10 + 10, -8 + 6)) = -5$, $\Delta(a_2, b_1) = \min(-1, \max(-5 + 2, -8 + 20)) = -1$, $\Delta(a_2, b_2) = \min(6, \max(0 + 2, -1 + 20)) = 6$.

Step 5: The constraint system is consistent, and $\Delta(a_2, b_2) = 6$.

If we calculate the maximum time separation only in G^3, then $\Delta(a_2, b_2) = 19$ which can not to satisfy all the constraints in the original constraint graph.

This example illustrates the situation when the time separation between a_2 and b_2 is influenced by the constraints in later unfoldings U_3, U_4, …, but as the system itself is consistent, we can calculate the time separation by considering only a few more unfoldings. In fact, in this example, one additional unfolding is enough, after that, the time separation does not change with the addition of further unfoldings.

5.2.2 Algorithm for Cyclic Linear-Plus-Latest Constraint Systems

As shown in Section 5.2.1, $\Delta(s_{\alpha}, e_{\alpha+\beta})$, $\alpha \geq K$, can be computed in $G^{\alpha+\beta+r}$. Starting from Δ^K, Δ^{K+i}, $i > 0$, can be computed step by step in $G^{K+i+\beta+r}$. Due to the fact that the newly added constraints keep repeating (because $\alpha \geq K$), $\Delta(s_{K+i}, e_{K+i+\beta})$ will become a periodic function of i, i.e., $\Delta(s_{K+i}, e_{K+i+\beta}) = \Delta(s_{K+i+c}, e_{K+i+c+\beta})$ for a large enough i, where c is a constant. When this happens, the maximum time separation Δ can be obtained by

$$\Delta = \max(\max_{0 \leq \alpha < K+i}(\Delta(s_{\alpha}, e_{\alpha+\beta})), \max_{\alpha \geq K+i}(\Delta(s_{\alpha}, e_{\alpha+\beta})))$$

$$= \max(\max_{0 \leq \alpha < K+i}(\Delta(s_{\alpha}, e_{\alpha+\beta})), \max_{K+i \leq \alpha \leq K+i+c-1}(\Delta(s_{\alpha}, e_{\alpha+\beta}))) \qquad (5.1)$$

In Equation (5.1), there are only a finite number of $\Delta(s_{\alpha}, e_{\alpha+\beta})$ and every $\Delta(s_{\alpha}, e_{\alpha+\beta})$ can be computed in a finite unfolded constraint graph $G^{\alpha+\beta+r}$. In other words, we can compute the maximum time separation Δ in a finite number of finite unfolded constraint graphs.

We show next that $\Delta(s_{K+i}, e_{K+i+\beta})$ will in fact become a periodic function of i.

5.2.2.1 Repetition of sd(s_{α}, v_j)

When computing $\Delta(s_{\alpha}, e_{\alpha+\beta})$ by applying Algorithm 4.1 to $G^{\alpha+\beta+r}$, $m(v_j)$ is the shortest distance from s_{α} to v_j in $G_m{}^{\alpha+\beta+r}$, where v_j is an event in unfolding U_j, $0 \leq j < \alpha + \beta + r$. As the structure of $G_m{}^{\alpha+\beta+r}$ is repeating after K unfoldings, it is well known that the $m(v_j)$ values are determined by the minimum ratio cycles of G_m [38] which is derived from the constraint graph G in the way as described in Definition 3.7. Here, a minimum ratio cycle c is such that $w(c)/\varepsilon(c) = \rho$ where

$$\rho = \min\left\{ \frac{w(c)}{\varepsilon(c)} \middle| c \text{ is a simple cycle in } G_m \right\} \tag{5.2}$$

A simple cycle in G_m is a cycle on which no vertex is repeated.

Then [38], there exist integers K^* and ε^* such that for all $\alpha \geq K^*$,

$$m(v_j) - m(v_{j+\varepsilon^*}) = \rho\varepsilon^*, 0 \leq j \leq \alpha - K^*, \tag{5.3}$$

where K^* is the number of unfoldings such that all the m-values start repeating and ε^* is the period of this repetition. From the well-formedness conditions in Section 3. 2, we have $\rho > 0$.

Example 5.2: Consider the constraint graph G of the repeated microprocessor READ cycle in Figure 3.3. Its G_m is shown in Figure 5.4.

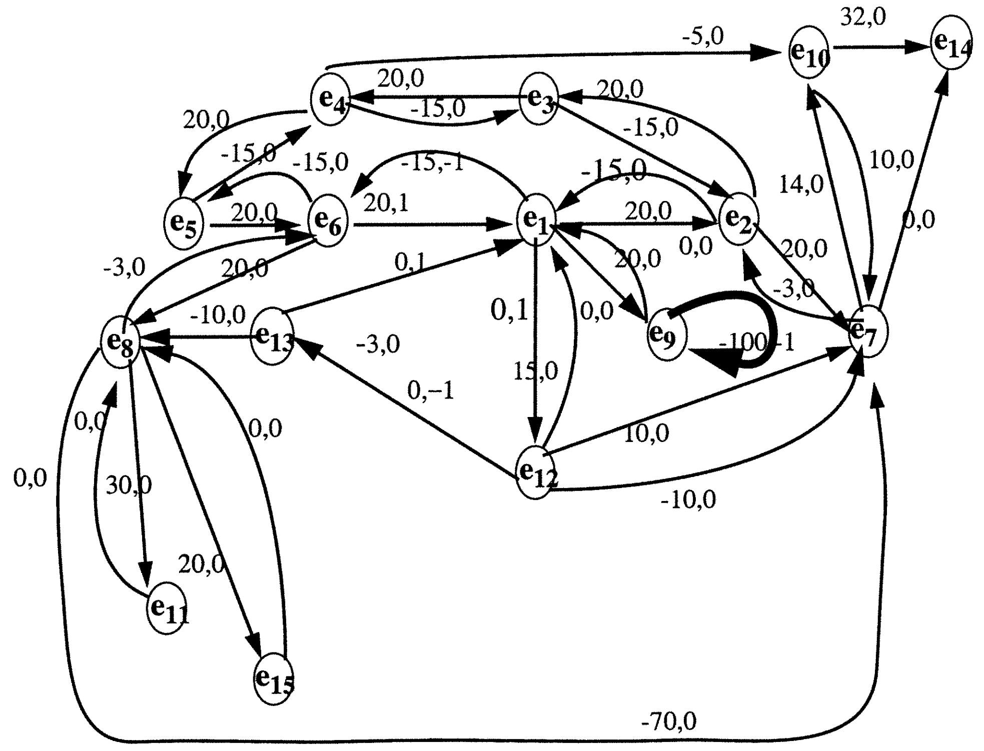


Figure 5.4 G_m of the constraint graph in Figure 3.3.

In G_m, we get $\varepsilon^* = 1$, $K^* = 3$ and $\rho = 100$ (the minimum ratio cycle is highlighted in Figure 5.4).

Then, $m(v_j)$ in $G_m^{\alpha+\beta+r}$, $0 \le j \le \alpha - K^*$, can be calculated by $m(v_i)$ also in $G_m^{\alpha+\beta+r}$, $\alpha - K^* < i < \alpha + \beta + r$, using Equation (5.3) for $\alpha \ge K^*$.

In the following Lemma, we show that the m values of events in $G^{\alpha+\beta+\varepsilon^*+r}$ can be computed from the m values of events in $G^{\alpha+\beta+r}$ for $\alpha \ge \max(K^*, K)$.

Lemma 5.1: If $\alpha \ge \max(K^*, K)$, then

$$\mathrm{sd}(s_{\alpha+\varepsilon^*}, v_j) = \mathrm{sd}(s_\alpha, v_j) + \rho\varepsilon^*,\ 0 \le j \le K + r + \varepsilon^*, \tag{5.4}$$

$$\mathrm{sd}(s_{\alpha+\varepsilon^*}, v_j) = \mathrm{sd}(s_\alpha, v_{j-\varepsilon^*}),\ K + r + \varepsilon^* < j < \alpha + \beta + r + \varepsilon^*, \tag{5.5}$$

where $\mathrm{sd}(s_\alpha, v_j)$ and $\mathrm{sd}(s_\alpha, v_{j-\varepsilon^*})$ are the shortest distances from s_α to v_j and to $v_{j-\varepsilon^*}$ in $G_m^{\alpha+\beta+r}$, respectively, while $\mathrm{sd}(s_{\alpha+\varepsilon^*}, v_j)$ and $\mathrm{sd}(s_{\alpha+\varepsilon^*}, v_{j+\varepsilon^*})$ are the shortest distances from $s_{\alpha+\varepsilon^*}$ to v_j and to $v_{j+\varepsilon^*}$ in $G_m^{\alpha+\beta+\varepsilon^*+r}$, respectively.

Proof:

The G_m graphs corresponding to $G^{\alpha+\beta+r}$ and $G^{\alpha+\beta+\varepsilon^*+r}$ are as shown in Figure 5.5. We first prove Equation (5.5).

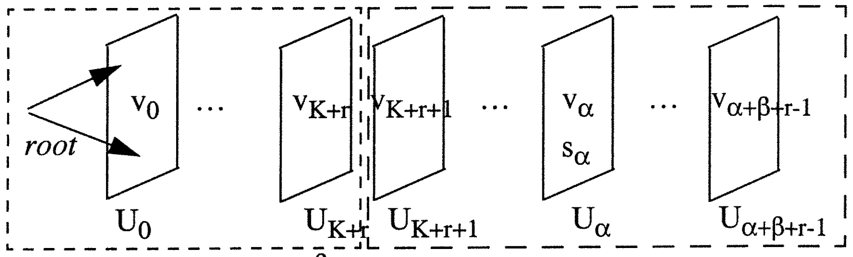


Figure 5.5a. $G_m^{\alpha+\beta+r}$

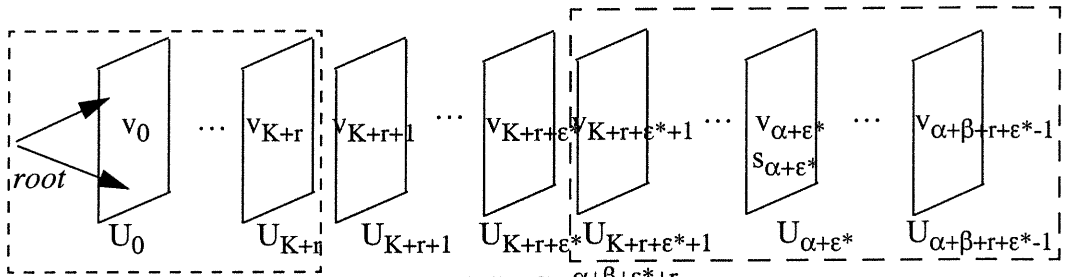

v_0
root
U_0
v_{K+r}
U_{K+r}
v_{K+r+1}
U_{K+r+1}
$v_{K+r+\varepsilon^*}$
$U_{K+r+\varepsilon^*}$
$v_{K+r+\varepsilon^*+1}$
$U_{K+r+\varepsilon^*+1}$
$v_{\alpha+\varepsilon^*}$
$s_{\alpha+\varepsilon^*}$
$U_{\alpha+\varepsilon^*}$
$v_{\alpha+\beta+r+\varepsilon^*-1}$
$U_{\alpha+\beta+r+\varepsilon^*-1}$
Figure 5.5b. $G_m^{\alpha+\beta+\varepsilon^*+r}$

Figure 5.5 The $G_m^{\alpha+\beta+r}$ and $G_m^{\alpha+\beta+\varepsilon^*+r}$

From Theorem 5.1, if the cyclic system G is consistent, then $\mathrm{sd}(s_{\alpha+\varepsilon^*}, v_j)$, K + r +

$\varepsilon^* \leq j < \alpha + \beta + r + \varepsilon^*$, is the shortest distance from $s_{\alpha+\varepsilon^*}$ to v_j in $G_m^{\alpha+\beta+\varepsilon^*+r}$. By applying Theorem 5.1 to $G_m^{\alpha+\beta+\varepsilon^*+r}$, the corresponding shortest path can only pass through events in $U_{K+\varepsilon^*+1}, \ldots, U_{\alpha+\beta+r+\varepsilon^*-1}$. On the other hand, $sd(s_\alpha, v_{j-\varepsilon^*})$, $K + r + \varepsilon^* \leq j \leq \alpha + \beta + r + \varepsilon^*$ (i.e., $K + r \leq j - \varepsilon^* \leq \alpha + \beta + r$), is the shortest distance from s_α to $v_{j-\varepsilon^*}$ in $G_m^{\alpha+\beta+r}$. By applying Theorem 5.1 to $G_m^{\alpha+\beta+r}$, the corresponding shortest path can only pass through events in $U_{K+1}, \ldots, U_{\alpha+\beta+r-1}$. But, the constraints in $U_{K+\varepsilon^*+1}, \ldots, U_{\alpha+\beta+\varepsilon^*+r-1}$ are the same as in $U_{K+1}, \ldots, U_{\alpha+\beta+r-1}$, because all these unfoldings are after U_K. Therefore, Equation (5.5) holds.

Now, we prove Equation (5.4). Since $\alpha \geq \max(K^*, K) \geq K^*$, we can apply Equation (5.3) to v_j in $G_m^{\alpha+\beta+\varepsilon^*+r}$, where $0 \leq j \leq \alpha + \varepsilon^* - K^*$, and obtain

$$sd(s_{\alpha+\varepsilon^*}, v_j) = sd(s_{\alpha+\varepsilon^*}, v_{j+\varepsilon^*}) + \rho\varepsilon^*, \; 0 \leq j \leq \alpha + \varepsilon^* - K^* \tag{5.6}$$

In the following, we consider two cases for j:

Case 1: $K + r < j \leq K + r + \varepsilon^*$. That is, $j + \varepsilon^* > K + r + \varepsilon^*$, by applying Equation (5.5) to $v_{j+\varepsilon^*}$, we have $sd(s_{\alpha+\varepsilon^*}, v_{j+\varepsilon^*}) = sd(s_\alpha, v_j)$. By substituting this relation to Equation (5.6), we get $sd(s_{\alpha+\varepsilon^*}, v_{j+\varepsilon^*}) = sd(s_\alpha, v_j) + \rho\varepsilon^*$. I.e., Equation (5.4) holds for $K + r < j \leq K + r + \varepsilon^*$.

Case 2: $0 \leq j \leq K + r$. By applying Equation (5.6) x times where $x = \lceil K + r - j/ \varepsilon^* \rceil$, we obtain

$$sd(s_{\alpha+\varepsilon^*}, v_j) = sd(s_{\alpha+\varepsilon^*}, v_{j+x\varepsilon^*}) + x\rho\varepsilon^*$$

$$= sd(s_{\alpha+\varepsilon^*}, v_{j+x\varepsilon^*+\varepsilon^*}) + x\rho\varepsilon^* + \rho\varepsilon^* \tag{5.7}$$

Since $x = \lceil K + r - j/ \varepsilon^* \rceil$ and $0 \leq j \leq K + r$, we have $j + x\varepsilon^* + \varepsilon^* > K + r + \varepsilon^*$. We can now apply Equation (5.5) to Equation (5.7) and obtain

$$\mathrm{sd}(s_{\alpha+\varepsilon^*}, v_j) = \mathrm{sd}(s_\alpha, v_{j+x\varepsilon^*}) + x\rho\varepsilon^* + \rho\varepsilon^*,\ 0 \le j \le K + r. \tag{5.8}$$

On the other hand, if $0 \le j \le K + r$, by applying Equation (5.3) x times in $G_m^{\alpha+\beta+r}$, we obtain

$$\mathrm{sd}(s_\alpha, v_j) = \mathrm{sd}(s_\alpha, v_{j+x\varepsilon^*}) + x\rho\varepsilon^*,\ 0 \le j \le K + r. \tag{5.9}$$

Comparing Equations (5.8) and (5.9), we can see that Equation (5.4) holds for $0 \le j \le K + r$.

By combining the two cases, we conclude that Equation (5.4) holds. ❑

In the following, we show that the periodicity of the sd values as indicated in Equations (5.4) and (5.5) also applies to the time separations from s_α to v_j^*, where v_j^* is an event on the shortest path from s_α to the *root* in $G_m^{\alpha+\beta+r}$.

5.2.2.2 Repetition of $\Delta(s_\alpha, v_j^*)$

Lemma 5.2: In a finite graph G^f containing linear-plus-latest constraints, if v is on the shortest path from s to *root* in G^f_m, then $\Delta(s, v) = m(v)$.

Proof:

By induction on the events in the reverse order as they appear on the shortest path from s to *root* in G^f_m.

Basis: v = *root*, $\Delta(s, root) = m(root)$ is true.

The induction hypothesis is that $\Delta(s, v) = m(v)$ holds. We then prove that $\Delta(s, u^*) = m(u^*)$ also holds, where $u^* \in \text{msources}(v)$ and u^*, v are on the shortest path from s to *root*.

As both u^* and v are on the shortest path from s to *root* in G^f_m, and $u^* \in \text{msources}(v)$, then u^* must be on the shortest path from s to v, i.e., $m(v) = \min_{u \in \text{msources}(v)}(m(u) + sd(u, v)) = m(u^*) + sd(u^*, v)$. Then, from Equation (4.5), $\forall u \in \text{msources}(v)$, $\Delta(s, u) + sd(u, v) \geq \Delta(s, v)$, and $\Delta(s, v) = m(v)$ due to the induction hypothesis. Hence, $\forall u \in \text{msources}(v)$, $\Delta(s, u) + sd(u, v) \geq m(v) = m(u^*) + sd(u^*, v)$. Applying this relation to u^* yields $\Delta(s, u^*) \geq m(u^*)$. On the other hand, from Equation (4.5), we have $\Delta(s, u^*) \leq m(u^*)$. Therefore, $\Delta(s, u^*) = m(u^*)$. ❑

From Lemma 5.1, we have that the shortest path from $s_{\alpha+\varepsilon^*}$ to v_j^* which is on the shortest path from $s_{\alpha+\varepsilon^*}$ to *root* in $G_m^{\alpha+\beta+\varepsilon^*+r}$, $\alpha \geq \max(K^*, K)$, can be computed using Equations (5.4) and (5.5). From Lemma 5.2, we have that $\Delta(s_{\alpha+\varepsilon^*}, v_j^*) = sd(s_{\alpha+\varepsilon^*}, v_j^*)$ in $G_m^{\alpha+\beta+\varepsilon^*+r}$ and $\Delta(s_\alpha, v_{j-\varepsilon^*}^*) = sd(s_\alpha, v_{j-\varepsilon^*}^*)$ in $G_m^{\alpha+\beta+r}$. Combining this result with Equation (5.5), we obtain

$$\Delta(s_{\alpha+\varepsilon^*}, v_j^*) = \Delta(s_\alpha, v_{j-\varepsilon^*}^*),\ K + r + \varepsilon^* < j < \alpha + \beta + r + \varepsilon^* \tag{5.10}$$

For $\alpha \geq K_{max}$, where

$$K_{max} = K^* + K + r + \varepsilon^*, \tag{5.11}$$

we have $K + r + \varepsilon^* \leq \alpha - K^*$. Form Equation (5.3), we get $sd(s_\alpha, v_j^*) + \rho\varepsilon^* = sd(s_\alpha, v_{j-\varepsilon^*}^*)$. By combining this with Equation (5.4), we obtain

$$\Delta(s_{\alpha+\varepsilon^*}, v_j^*) = \Delta(s_\alpha, v_{j-\varepsilon^*}^*),\ \varepsilon^* \leq j \leq K + r + \varepsilon^*. \tag{5.12}$$

Combining Equations (5.10) and (5.12) yields

$$\Delta(s_{\alpha+\varepsilon^*}, v_j^*) = \Delta(s_\alpha, v_{j-\varepsilon^*}^*), j \geq \varepsilon^* \tag{5.13}$$

In other words, the time separations for events on the shortest path from s_α to *root* are periodic functions with period ε^*. In the same way as we compute $sd(s_{\alpha+\varepsilon^*}, v_j^*)$ values using Equation (5.4) and (5.5), $\Delta(s_{\alpha+\varepsilon^*}, v_j^*)$ can be calculated as follows:

$$\Delta(s_{\alpha+\varepsilon^*}, v_j^*) = \Delta(s_\alpha, v_j^*) + \rho\varepsilon^*, 0 \leq j \leq K + r + \varepsilon^*, \tag{5.14}$$

$$\Delta(s_{\alpha+\varepsilon^*}, v_j^*) = \Delta(s_\alpha, v_{j-\varepsilon^*}^*), K + r + \varepsilon^* < j < \alpha + \beta + r + \varepsilon^*, \tag{5.15}$$

where $\Delta(s_\alpha, v_j^*)$ and $\Delta(s_\alpha, v_{j-\varepsilon^*}^*)$ are the time separations from s_α to v_j^* and to $v_{j-\varepsilon^*}^*$ in $G^{\alpha+\beta+r}$, respectively, while $\Delta(s_{\alpha+\varepsilon^*}, v_j^*)$ and $\Delta(s_{\alpha+\varepsilon^*}, v_{j+\varepsilon^*}^*)$ are the time separations from $s_{\alpha+\varepsilon^*}$ to v_j^* and to $v_{j+\varepsilon^*}^*$ in $G^{\alpha+\beta+\varepsilon^*+r}$, respectively.

That is, for $\alpha \geq K_{max}$ the Δ values of events which are on the shortest path from s_α to *root* in $G_m^{\alpha+\beta+r}$ will present the same periodic behaviour as the sd values.

5.2.2.3 Repetition of $\Delta(s_\alpha, v_j)$

Due to the repetition of the m values and of the structure of the unfolded constraint graphs, all $\Delta(s_\alpha, v_j)$ values will eventually repeat periodically as indicated in Equation (5.13). But, we do not know when the periodic behaviour begins. In the following, we show that if there is an $\alpha = K' \geq K_{max}$ such that the equations between the Δ values of all events v_j, in $G^{K'+\beta+\varepsilon^*+r}$ and those in $G^{K'+\beta+r}$ are as v_j^* in Equations (5.14) and (5.15), then K' is an upper bound on the number of unfoldings such that for all $i > 0$, $\Delta(s_{K'+i}, e_{K'+i+\beta})$ will be a periodic function with a period of ε^*.

Theorem 5.3: Let $\Delta(s_{K'}, v_j)$ and $\Delta(s_{K'}, v_{j-\varepsilon*})$ be the time separations from $s_{K'}$ to v_j and to $v_{j-\varepsilon*}$ in $G^{K'+\beta+r}$, respectively, and let $\Delta(s_{K'+\varepsilon*}, v_j)$ and $\Delta(s_{K'+\varepsilon*}, v_{j+\varepsilon*})$ be the time separations from $s_{K'+\varepsilon*}$ to v_j and to $v_{j+\varepsilon*}$ in $G^{K'+\beta+\varepsilon*+r}$, respectively, where K' is an integer. If K' satisfies $K' \geq K_{max}$ such that

$$\Delta(s_{\alpha+\varepsilon*}, v_j) = \Delta(s_\alpha, v_j) + \rho\varepsilon*,\ 0 \leq j \leq K + r + \varepsilon*, \tag{5.16}$$

$$\Delta(s_{\alpha+\varepsilon*}, v_j) = \Delta(s_\alpha, v_{j-\varepsilon*}),\ K + r + \varepsilon* < j < \alpha + \beta + r + \varepsilon*, \tag{5.17}$$

hold for $\alpha = K'$, then Equations (5.16) and (5.17) also hold for $\alpha = K' + 1$.

Proof (of Theorem 5.3):

By induction on the iterative computation of $\Delta(s_{\alpha+2\varepsilon*}, v_j)$ in $G^{K'+2\varepsilon*+\beta+r}$.

In the first iteration, $\Delta_1(s_{\alpha+2\varepsilon*}, v_j) = sd(s_{\alpha+2\varepsilon*}, v_j)$, $0 \leq j \leq K' + \beta + r + 2\varepsilon*$, where Δ_1 is the time separation in the first iteration. By applying Lemma 5.1 to $G^{K'+2\varepsilon*+\beta+r}$, we obtain that

$$sd(s_{\alpha+2\varepsilon*}, v_j) = sd(s_{\alpha+\varepsilon*}, v_j) + \rho\varepsilon*,\ 0 \leq j \leq K + r + 2\varepsilon*, \tag{5.18}$$

$$sd(s_{\alpha+2\varepsilon*}, v_j) = sd(s_{\alpha+\varepsilon*}, v_{j-\varepsilon*}),\ K + r + 2\varepsilon* < j < K' + \beta + r + 2\varepsilon*, \tag{5.19}$$

That is, Equations (5.16) and (5.17) hold for $\alpha = K' + 1$ in the first iteration.

The induction hypothesis is that Equations (5.16) and (5.17) hold in (κ-1)-th iteration. From Equation (4.5), it follows that

$\Delta_\kappa(s_{K'+2\varepsilon*}, v_j) = \min(sd(s_{K'+2\varepsilon*}, v_j), \min_{u \in msources(v_j)} (\Delta_{\kappa-1}(s_{K'+2\varepsilon*}, u) + sd(u,$

$$v_j)),\ \max_{u \in prdes(v_j)} (\Delta_{\kappa-1}(s_{K'+2\varepsilon^*}, u) + D_{uv_j}))\ \text{and} \tag{5.20}$$

$$\Delta_\kappa(s_{K'+\varepsilon^*}, v_j) = \min (sd(s_{K'+\varepsilon^*}, v_j), \min_{u' \in msources(v_j)} (\Delta_{\kappa-1}(s_{K'+\varepsilon^*}, u') + sd(u', v_j)), \max_{u' \in prdes(v_j)} (\Delta_{\kappa-1}(s_{K'+\varepsilon^*}, u) + D_{u'v_j})), \tag{5.21}$$

where Δ_κ and $\Delta_{\kappa-1}$ are the time separations in the κ-th and the (κ - 1)-th iteration, respectively.

If $0 \le j \le K + r + \varepsilon^*$, by comparing the computation of $\Delta_\kappa(s_{K'+2\varepsilon^*}, v_j)$ and that of $\Delta_\kappa(s_{K'+\varepsilon^*}, v_j)$, in Equations (5.20) and (5.21), we can see the following 3 facts:

a) all events u in Equation (5.20) are in one to one correspondence with events u' in Equation (5.21);

b) $sd(s_{K'+2\varepsilon^*}, v_j) = sd(s_{K'+\varepsilon^*}, v_j) + \rho\varepsilon^*$ (Equation (5.4)), $\Delta_{\kappa-1}(s_{K'+2\varepsilon^*}, u) = \Delta_{\kappa-1}(s_{K'+\varepsilon^*}, u') + \rho\varepsilon^*$ (due to (5.16) in the hypothesis), $sd(u, v_j) = sd(u', v_j)$;

c) $D_{uv_j} = D_{u'v_j}$.

Therefore, $\Delta(s_{K'+2\varepsilon^*}, v_j) = \Delta(s_{K'+\varepsilon^*}, v_j) + \rho\varepsilon^*$ for $0 \le j \le K + r + \varepsilon^*$, i.e., Equation (5.16) holds in the κ-th iteration.

If $K + r + \varepsilon^* < j \le K + r + 2\varepsilon^*$, from Equation (4.5) we have

$$\Delta_\kappa(s_{K'+\varepsilon^*}, v_{j-\varepsilon^*}) = \min (sd(s_{K'+\varepsilon^*}, v_{j-\varepsilon^*}), \min_{u' \in msources(v_j)} (\Delta_{\kappa-1}(s_{K'+\varepsilon^*}, u') + sd(u', v_{j-\varepsilon^*})), \max_{u' \in prdes(v_j)} (\Delta_{\kappa-1}(s_{K'+\varepsilon^*}, u) + D_{u'v_j})) \tag{5.22}$$

By comparing the computation of $\Delta_\kappa(s_{K'+2\varepsilon^*}, v_j)$ to the computation of $\Delta_\kappa(s_{K'+\varepsilon^*}, v_{j-}$

$_{\varepsilon*}$) in Equations (5.20) and (5.22), we can see that:

a) all events u in Equation (5.20) are in one to one correspondence with events u' in Equation (5.22);

b) $sd(s_{K'+2\varepsilon*}, v_j) = sd(s_{K'+\varepsilon*}, v_{j-\varepsilon*})$ (Equation (5.5)), $\Delta_{\kappa-1}(s_{K'+2\varepsilon*}, u) = \Delta_{\kappa-1}(s_{K'+\varepsilon*}, u')$ (due to Equation (5.17) in the hypothesis), and $sd(u, v_j) = sd(u', v_j)$;

c) $D_{uv_j} = D_{u'v_j}$.

Therefore, $\Delta(s_{K'+2\varepsilon*}, v_j) = \Delta(s_{K'+\varepsilon*}, v_{j-\varepsilon*})$ for $K + r + \varepsilon* < j \leq K + r + 2\varepsilon*$, i.e., Equation (5.17) holds in the κ-th iteration.

If $K + r + 2\varepsilon* < j < K + \beta + r + 2\varepsilon*$, by comparing the computation of $\Delta_\kappa(s_{K'+2\varepsilon*}, v_j)$ to the computation of $\Delta_\kappa(s_{K'+\varepsilon*}, v_j)$, the right hand sides of Equations (5.20) and (5.21), we can see that:

a) all events u in Equation (5.20) are in one to one correspondence with events u' in Equation (5.21);

b) $sd(s_{K'+2\varepsilon*}, v_j) = sd(s_{K'+\varepsilon*}, v_j)$ (Equation (5.5)), $\Delta_{\kappa-1}(s_{K'+2\varepsilon*}, u) = \Delta_{\kappa-1}(s_{K'+\varepsilon*}, u')$ (due to (5.17) in the hypothesis), and $sd(u, v_j) = sd(u', v_j)$;

c) $D_{uv_j} = D_{u'v_j}$.

Therefore, $\Delta(s_{K'+2\varepsilon*}, v_j) = \Delta(s_{K'+\varepsilon*}, v_j)$ for $K + r + 2\varepsilon* + 1 < j < K + \beta + r + 2\varepsilon*$, i.e., Equation (5.17) holds in the κ-th iteration.

It follows that Equations (5.16) and (5.17) hold in the κ-th iteration, and thus Theorem 5.3 is true.

□

Theorem 5.4: If Equations (5.16) and (5.17) are true for $\alpha = K'$, then, $\forall\, i \geq 0$

$$\forall\, i \geq 0,\ \Delta(s_{K'+(i+1)\varepsilon^*}, v_{j+(i+1)\varepsilon^*}) = \Delta(s_{K'+i\varepsilon^*}, v_{j+i\varepsilon^*}),\ K + r < j < K' + \beta + r. \quad (5.23)$$

I.e., $\Delta(s_{K'+i\varepsilon^*}, v_{j+i\varepsilon^*})$ is a periodic function of i with period ε^*.

Proof:

By induction on i. The induction base is i = 0, then by applying Theorem 5.3 to K', Equation (5.23) holds for i = 0. The induction hypothesis is that Equation (5.23) holds for i = j. Then, by applying Theorem 5.3 to K' + j, we have that Equation (5.23) holds for i = j + 1. Therefore, Theorem 5.4 is true.

□

Equations (5.16) and (5.17) indicate that the time separation from $s_{K'+\varepsilon^*}$ to v_j, $0 \leq j \leq K + r + \varepsilon^*$ behaves the same way as $sd(s_{K'+\varepsilon^*}, v_j)$. In other words, the time separation $\Delta(s_{K'+\varepsilon^*}, v_j)$ is determined by $\Delta(s_{K'+\varepsilon^*}, v_j^*)$ and the constraints from v_j^* to v_j, where v_j^* is the set of events in unfolding U_j which are on the shortest path from $s_{K'+\varepsilon^*}$ to the *root*.

The maximum time separation $\Delta = \max_{\alpha \geq 0}(\Delta(s_\alpha, e_{\alpha+\beta}))$ can thus be calculated by finding an upper bound $K' \geq K_{max}$ on the number of unfoldings such that $\Delta(s_{K'+i}, e_{K'+\beta+i})$ is a periodic function for all i > 0. If $\varepsilon^* = 1$, then from Theorem 5.3, $\Delta = \max_{\alpha \geq 0}(\Delta(s_\alpha, e_{\alpha+\beta})) = \max_{0 \leq \alpha \leq K'}(\Delta(s_\alpha, e_{\alpha+\beta}))$. If $\varepsilon^* \neq 1$, then we have to find K_0', ..., K_{ε^*-1}' such that the corresponding conditions in Theorem 5.3 are satisfied. Thereafter, Δ can be computed as

$$\Delta = \max_{0 \le j \le \varepsilon^*-1}(\max_{0 \le \alpha \le K_j'}(\Delta(s_\alpha, e_{\alpha+\beta}))).$$

The above discussion leads to the following algorithm for computing the maximum time separation in cyclic linear-plus-latest constraint systems.

Algorithm 5.3: Maximum time separation in cyclic linear-plus-latest constraint systems.

Step 1: Pre-calculate K^*, ε^*, and K_{max}, where K^*, ε^* are calculated such that Equation (5.3) are satisfied, and K_{max} is computed using Equation (5.11)

Step 2: For $0 \le \alpha < K_{max}$, apply Algorithm 4.1 to compute $\Delta(s_\alpha, e_{\alpha+\beta})$ in $G^{\alpha+\beta+r}$

Step 3: Initialize $i = 0$, $J = \{0, \ldots, \varepsilon^*\}$

```
    Repeat
        $\forall j \in J$, compute $\Delta(s_{K_{max}+j+i\varepsilon^*}, e_{K_{max}+j+i\varepsilon^*+\beta})$ in $G^{K_{max}+j+i\varepsilon^*+\beta+r}$,
        add $\varepsilon^*$ unfoldings to the graphs $G^{K_{max}+j+i\varepsilon^*+\beta+r}$
        if the conditions in Theorem 5.3 hold for some j, then
            $J = J - \{j\}$, $K_j' = K_{max} + i\varepsilon^*$
        else increment i and go back to the repeat loop
        endif
    until $J = \varnothing$
```

Step 4: return $\Delta = \max_{0 \leq j \leq \varepsilon^*-1}(\max_{0 \leq \alpha \leq K_j'}(\Delta(s_\alpha, e_{\alpha+\beta})))$.

Complexity of Algorithm 5.3:

Similar to the max-only systems in [42], the values of K^*, K_j' and ε^* are dependent on the values of the bounds of the constraints in the constraint graph G. The upper bound on the number of unfoldings, K_j', for the time separation to become a periodic function may be very large. As stated in [42], in reality, ε^* is a small value, but K^* can be very large if there is a cycle in G_m with $w(c)/\varepsilon(c)$ very close to ρ. We do not know anything about K_j' except that $K_j' \geq K^*$.

In the next Chapter, we restrict the constraint graph G such that all the unfolded constraint graphs of G are causal. We find a way to determine the upper bounds of K_j' and thus get a practical algorithm of computing the maximum time separation of events in the restricted constraint graphs.

Chapter 6

Maximum Time Separations in Causal Linear-plus-latest Cyclic Systems

As mentioned in Section 5.2, even though we gave a condition in Theorem 5.3 to check when Δ_α becomes periodic function of α, the number of unfoldings needed for Δ_α to become periodic depends on the delay values in the specification and can be very large. Due to the fact that finding even one Δ_α is fairly complex (pseudo polynomial in the number of events and additional iterations are normally inevitable due to linear constraints). In other words, Algorithm 5.3 is impractical for realistic complex problems.

We reduce the complexity of the problem by restricting the constraint graph specifications. The constraint graphs are restricted in such a way that events in the unfolded constraint graphs can be partitioned into a set of topologically ordered blocks that respect the causality conditions. The restricted graphs reflect realizable designs and can thus model most realistic systems. We can now determine an upper bound on the number of unfoldings to compute the maximum time separation Δ and obtain a practical algorithm.

This chapter consists of five sections. In Section 6.1, we give the restrictions on the cyclic constraint graphs. In Section 6.2, we simplify Algorithm 4.1 for finding the time separation in a restricted finite unfolded graph with linear-plus-latest constraints to an algorithm applicable only in causal finite graphs. In Section 6.3, we derive an algorithm of computing the maximum time separation in restricted cyclic constraint graphs. Since the causality conditions of constraint graphs need the information of the maximum time separation between events and our algorithm is developed under the assumption that the system is causal, there is an interleaving between computing the maximum time separations and causality checking. In Section 6.4, we present sufficient causal conditions and we apply our solution technique to verify two safety timing properties of the interface specification describing a repeated microprocessor READ from memory as stated in Section 3. 4. Finally in Section 6.5, we conclude this Chapter.

6.1 Restricted Constraint Graphs

We restrict constraint graphs with linear-plus-latest timing constraints to causal ones. A constraint graph is said to be causal if all the unfolded constraint graphs are causal. More formally, we give the definition of a causal constraint graph as follows:

Definition 6.1: A constraint graph $G = <E, R>$ is causal if $\forall \alpha \geq 0$, the unfolded constraint graph G^{α} is causal.

Recall that in Section 2.2, we give causality conditions of leaf TD specifications. Here we define causality of a finite constraint graph $G^f = <E^f, R^f>$ containing linear-plus-latest timing constraints [36].

Definition 6.2: A finite graph $G^f = <E^f, R^f>$ with linear-plus-latest constraints is causal if the following three conditions are satisfied:

1. For every event $v \in E^f$ where v is involved in a latest constraint, B(v) is a singleton, i.e., $B(v) = \{v\}$. In this case, B(v) is called a latest block.

2. For every event $v \in E^f$ where v is involved in a linear constraint, all other events in B(v) are also involved in linear constraints. Furthermore, for all $v \in B$ and for all triggers $u \in trigs(B(v))$, the trigger u must occur in the past of all v in B (called well-defined triggers [50]). In this case, B(v) is called a linear block.

3. For all $B \in P$ (see Definition 2.6 for a block partition P), the separation between triggers of B satisfying all the constraints in the graph G^f are strictly tighter than those computed using the local constraints of B.

From these conditions it follows that there is a topological order $\leq$ of the blocks, i.e., $\forall(u, v) \in R^f$, $B(u) \leq B(v)$.

In Definition 6.1, we define a constraint graph as causal if all the unfolded constraint graphs are causal. Based on the semantics of causal finite constraint graphs described in [50], the block partition of the unfolded causal constraints graphs is past-dominated.

Definition 6.3[50]: A block partition P of a finite graph G^f is past-dominated if each block B, $B \in P$, is past-dominated.

As stated in [50] the occurrence times of the events in B are independent of the occurrence times of events in topologically later blocks. More formally, consider a finite graph G^f with a causal block partition $B_1 < B_2 \dots < B_n$, let $G^{f'}$ be the finite constraint graph induced by the vertices $E^{f'} = \bigcup_{i=1}^{k} B_i$ for some $1 \leq k \leq n$, then for any consistent timing assignment τ' for $G^{f'}$, there exists a consistent timing assignment τ for G^f such that $\forall v \in E^{f'}$, $\tau(v) = \tau'(v)$.

In the next section, based on the properties of finite causal constraint graphs, we derive a simplified Algorithm 4.1 for computing the time separation between events in finite causal constraint graphs.

6.2 Simplification of Algorithm 4.1 in Restricted Acyclic Graphs

Recall that the time separation from event s to an event $v \in E$ in a finite constraint graph $G^f = <E^f, R^f>$ can be expressed as in Equation (4.5), i.e.,

$$\Delta(s, v) = \min(m(v), \min_{u \in msources(v)}(\Delta(s, u) + sd(u, v)), \max_{u \in preds(v)}(\Delta(s, u) + D_{uv})) \quad (6.1)$$

where msources(v) are all the source events to v in G^f_m, preds(v) is the set of source events to v involved in a max combination in G^f, and sd(u, v) is the shortest distance from u to v in G^f_m. In the following, we will simplify Equation (6.1) based on the properties of causal finite graphs.

Consider an event v in a latest block B(v): As all constraints from earlier blocks to v are latest constraints (from preds(v) to v), these max constraints are transformed to edges from v to preds(v) in G^f_m (Section 3. 2). Hence, $\forall u \in$ preds(v) in G^f, $v \in$ msources(u) in G^f_m. In other words, $\forall u \in$ preds(v) in G^f, $u \notin$ msources(v) in G^f_m, and all the events in msources(v) are in topologically later blocks than B(v). It follows that $\min_{u \in msources(v)}\{\Delta(s, u) + sd(u, v)\}$ will not play any role in Equation (6.1). Therefore, for an event v in a latest block, Equation (6.1) can be rewritten as

$$\Delta(s, v) = \min(m(v), \max_{u \in preds(v)}\{\Delta(s, u) + D_{uv}\}) \quad (6.2)$$

Consider now an event v in a linear block B(v): Since B(v) is past-dominated, then the occurrence time $\tau(v)$ of v is determined by the occurrence times of the triggers and the local constraints of B(v). The relationship between the latest occurrence times of u and v, $\tau_{max}(u)$ and $\tau_{max}(v)$, $u \in$ trigs(v), can be expressed as follows:

$$\tau_{max}(v) = \min_{u \in trigs(v)}\{\tau_{max}(u) + sdl(u, v)\} \quad (6.3)$$

where sdl(u, v) is the shortest distance from u to v satisfying the local constraints of

B(v).

As $\Delta(s, v)$ is the maximum time separation from s to v satisfying all the constraints in G^f, by setting s as the reference point and subtracting $\tau(s)$ from both sides in Equation (6.3), we obtain

$$\Delta(s, v) = \min_{u \in trigs(v)}\{\Delta(s, u) + sdl(u, v)\} \tag{6.4}$$

From Equation (6.1), we know that $\forall v \in E^f$,

$$\Delta(s, v) \leq m(v). \tag{6.5}$$

Therefore, $\Delta(s, v) = \min_{u \in trigs(v)}\{\Delta(s, u) + sdl(u, v)\} \leq m(v)$.

This Equation (4.5) can be rewritten to a form similar to the equation for a latest block (Equation (6.2)), yielding

$$\Delta(s, v) = \min(m(v), \min_{u \in trigs(v)}\{\Delta(s, u) + sdl(u, v)\}) \tag{6.6}$$

Since all the linear blocks in G^f are past-dominated, starting from the *root* and performing the calculation in Step 3 of Algorithm 4.1 block by block in the topological order of the blocks, we can compute $\Delta(s, v)$ in one iteration. We thus get the following simplified algorithms of finding the time separation of events in a finite causal constraint graph.

Algorithm 6.1: Time separation in a finite causal constraint graph with linear-plus-latest constraints.

Given $G^f = <E^f, R^f>$ and its causal partition P, the start event s and the end event e, the time separation $\Delta(s, e)$ from s to e can be computed in the following three steps:

```
Step 1: Construct $G^f_m$ from $G^f$.

Step 2: $\forall (u, v) \in E^f$, compute the shortest distance sd(u,v) from u to v in $G^f_m$. For
all linear blocks $B \in P$, $\forall u, v \in B$, compute sdl(u,v) using local constraints of B.

Step 3: Set $\Delta(s, root) = m(root)$,
        For $B \in P$ in a topological order, do
            if B is a latest block, then $\{v\} = B$ and let
                $\Delta(s, v) = \min(sd(s, v), \max_{u \in preds(v)}(\Delta(s, u) + D_{uv}))$,
            else (B is a linear block), $\forall u, v \in B$, let
                $\Delta(s, v) = \min_{u \in trigs(v)}\{\Delta(s, u) + sdl(u, v)\}$.
            endif
```

Since the complexity of finding the shortest distances in Steps 1 and 2 of Algorithm 6.1 is in $O(n^3)$, the complexity of Step 3 can be analyzed as follows: the maximum number of events in preds(v) or trigs(v) is (n-1), the complexity of computing the min and the max functions in Step 3 is in O(n), there are totally n $\Delta(s, v)$ need to be computed, therefore, the complexity of Step 3 is in $O(n^2)$. Therefore, $\Delta(s, v)$ in a causal finite constraint graph G^f can be computed in $O(n^3)$.

6.3 Maximum Time Separation in Restricted Cyclic Constraint Graphs

We now consider the problem of determining the maximum time separation in restricted causal cyclic constraint graphs. We will only consider identical block partitions in each unfolding. That is, each unfolding U_i, $0 \le i \le \alpha - 1$ is partitioned into a set of disjoint blocks $B_i = \{B_{i1}, B_{i2}, \ldots, B_{ij}\}$ satisfying Equation (6.7),

$$\forall k \in [1, j], \text{ if } v_i \in B_{ik}, \text{ then } v_m \in B_{mk} \text{ for all } m \in [0, \alpha - 1] \qquad (6.7)$$

Example 6.1: For the example of the specification of repeated READ of the MC68360 processor to a memory specified in Figure 3.3, one possible block partition P in each unfolding is as shown in Figure 6.1: $B_0 = \{e_1, e_2, e_3, e_9, e_{12}\}$, $B_1 = \{e_{10}, e_{14}\}$, $B_2 = \{e_4, e_5, e_6, e_8, e_{13}\}$, $B_3 = \{e_{10}, e_{14}\}$, and $B_4 = \{e_7\}$, where $B_0 \sim B_3$ are linear blocks and B_4 is a latest block. The (only) possible order of the blocks is B_0, B_4, B_1, B_2, B_3.

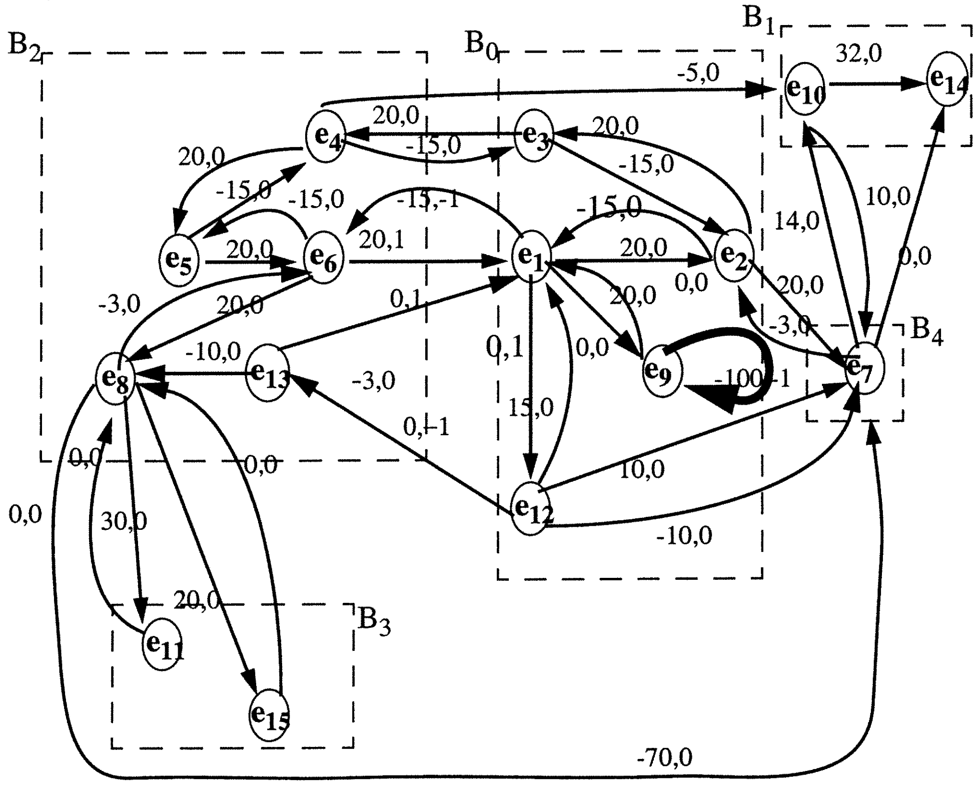


Figure 6.1 Block partition of the constraint graph of Figure 3.3

When the constraint graph is causal, we can determine the time separation Δ^{α} in $G^{\alpha+\beta+r}$ using Algorithm 6.1 (Section 6.2). However, the causality conditions (the existence of a past-dominated block partition with well-formed triggers) must be checked by determining the maximum time separation between the triggers and also the separation from the triggers to the events in the triggered blocks. In other words, there is an interleaving between causality checking of a constraint graph and the calculation of the maximum time separation. In this Section, we assume that the constraint graph G is causal, and based on this assumption, derive an algorithm for determining the maximum time separation in such graphs. We will then derive causality conditions in Section 6.4 using the maximum time separation algorithm developed here.

As defined in Section 3. 4, the maximum time separation Δ, is determined as the maximum over Δ^{α} for all $\alpha \geq 0$ (supposing $\beta \geq 0$). In causal systems, each Δ_{α} value can be computed in the unfolded constraint graph containing $\alpha + \beta + 1$ unfoldings $U_0, \ldots, U_{\alpha+\beta}$. That is, Δ^{α} is the time separation from s_{α} to $e_{\alpha+\beta}$ in $G^{\alpha+\beta+1} = (E^{\alpha+\beta+1}, R^{\alpha+\beta+1})$ where

$$E^{\alpha+\beta+1} = \{v_j \mid v \in E, 0 \leq j \leq \alpha + \beta\}, \text{ and}$$

$$R^{\alpha+\beta+1} = \{u_{j-\varepsilon} \xrightarrow{[d, D], \varepsilon} v_j \mid u_{j-\varepsilon}, v_j \in E^{\alpha+\beta},\ u \xrightarrow{[d_{uv}, D_{uv}], \varepsilon} v \in E\}.$$

We can avoid explicitly analysing Δ^{α} for all values of α by exploiting the periodicity of $\Delta(s_{\alpha}, v_{\alpha+\beta})$ for sufficiently large values of α.

As mentioned in Section 5.2, the values of $sd(s_{\alpha}, v_j)$ for $v_j \in E^{\alpha+\beta+1}$ are eventually determined by the minimum ratio cycle of G_m as given in Equation (5.2), i.e., there exists integers K^* and ε^* such that for all $\alpha \geq K^*$, the relation

$$m(v_j) - m(v_{j+\varepsilon^*}) = \rho\varepsilon^*, 0 \leq j \leq \alpha - K^* \qquad (6.8)$$

holds.

We have proved in Chapter 5 that in cyclic linear-plus-latest constraint graphs, $\Delta(s_{\alpha}, v_j)$ exhibit a periodic behaviour similar to the m values. In the following, we show that for causal constraint graphs, we can obtain an upper bound on the number of iterations for the time separation $\Delta(s_\alpha, e_{\alpha+\beta})$ to become a periodic function.

Theorem 6.1: Let G = <E, R> be a cyclic causal constraint graph. Consider determining the maximum time separation from s_α to $e_{\alpha+\beta}$ in $G^{\alpha+\beta+1}$, $\alpha \geq K^*$. There exists an integer F such that

$$\Delta(s_\alpha, v_{j+\varepsilon^*}) - \Delta(s_\alpha, v_j) = \rho\varepsilon^*,\ F \leq j \leq \alpha - K^* - \varepsilon^*$$

To prove this theorem, we need several properties of $\Delta(s_\alpha, v_j)$ in $G^{\alpha+\beta+1}$.

The first property is as given in Lemma 5.2 in Section 5.2.2, i.e., $\Delta(s_\alpha, v_j^*) = m(v_j^*)$ for events v_j^* which are on the shortest path from s_α to the *root* in $G_m^{\alpha+\beta+1}$. For such v_j^*, we have that

$$\Delta(s_\alpha, v_{j+\varepsilon^*}^*) - \Delta(s_\alpha, v_j^*) = m(v_{j+\varepsilon^*}^*) - m(v_j^*) \qquad (6.9)$$

Theorem 6.1 then follows immediately from Equation (6.8). For those v_j where $\Delta(s_\alpha, v_j) \neq m(v_j)$, $\Delta(s_\alpha, v_j)$ must be determined by the upper bounds of some latest constraints.

Definition 6.4: Given a constraint graph G = <E, R>, a graph G_M = <E, R_M> is constructed as follows: for each edge $u \xrightarrow{[d_{uv}, D_{uv}], \varepsilon} v$ in R involved in a max combination, there is an edge $v \xrightarrow{D_{uv}, \varepsilon} u$ in R_M, for each edge $u \xrightarrow{[b_{uv}, B_{uv}], \varepsilon} v$ in R involved in a linear combination, there are two edges in R_M: $u \xrightarrow{sdl(u, v)), \varepsilon} v$ if sdl(u, v) is not ∞ and $v \xrightarrow{sdl(v, u), -\varepsilon} u$ if sdl(v, u) is not ∞.

Let $D(c)$ denote the sum of the weights on the edges of a cycle c in G_M. A cycle c in G_M is said to be constraining v_j if $\Delta(s_\alpha, v_j) < m(v_j)$ and $\Delta(s_\alpha, v_j) = \Delta(s_\alpha, v_{j-\varepsilon(c)}) + D(c)$. In order to determine F in Theorem 6.1, we need to argue about $\varepsilon(c)$ of such constraining cycles. We first show that for a cycle c to be constraining, $D(c)$, must be strictly less than $\rho\varepsilon(c)$.

Lemma 6.1: Let c be a constraining cycle for $v_j \in E^{\alpha+\beta+1}$ with $j \leq \alpha - K^*$. Then

$$D(c) < \rho\varepsilon(c) \tag{6.10}$$

Proof: To simplify the notation, we assume $\varepsilon^* = 1$. Let u_i be a vertex on the shortest path from s_α to the *root* such that $B(v_{j-\varepsilon(c)}) < B(u_i) < B(v_j)$ (such a vertex always exists). As $j \leq \alpha - K^*$, $u_{i-\varepsilon(c)}$ is also on the shortest path from s_α to the *root* in $G_m^{\alpha+\beta+1}$.

$$\begin{aligned}
\Delta(s_\alpha, v_j) &< sd(s_\alpha, v_j) && \text{c is constraining } v_j. \\
&\leq sd(s_\alpha, u_i) + sd(u_i, v_j) && \text{property of shortest paths} \\
&= sd(s_\alpha, u_{i-\varepsilon(c)}) + \rho\varepsilon(c) + sd(u_i, v_j) && \text{from Equation (6.8).} \qquad (6.11)
\end{aligned}$$

In a causal constraint graph, $\Delta(s_\alpha, v_{j-\varepsilon(c)}) \geq \Delta(s_\alpha, u_{i-\varepsilon(c)}) + sd(u_{i-\varepsilon(c)}, v_{j-\varepsilon(c)})$, and $\Delta(s_\alpha, u_{i-\varepsilon(c)}) = sd(s_\alpha, u_{i-\varepsilon(c)})$ since $u_{i-\varepsilon(c)}$ is on the shortest path from s_α to the *root.* Combining these two facts, we obtain

$$-\Delta(s_\alpha, v_{j-\varepsilon(c)}) \leq -sd(s_\alpha, u_{i-\varepsilon(c)}) - sd(u_{i-\varepsilon(c)}, v_{j-\varepsilon(c)}) \tag{6.12}$$

By adding Equations (6.11) and (6.12) and realizing that $sd(u_{i-\varepsilon(c)}, v_{j-\varepsilon(c)}) = sd(u_i,$

v_j), we derive that

$$D(c) = \Delta(s_\alpha, v_j) - \Delta(s_\alpha, v_{j-\varepsilon(c)}) < \rho\varepsilon(c) \tag{6.13}$$

❑

Lemma 6.2: Consider two vertices u_i and v_j of $E^{\alpha+\beta+1}$, such that $j \le \alpha - K^*$, u_i has a path to v_j in $G^{\alpha+\beta+1}$, and u_i is on the shortest path from s_α to the *root*. Then there is a constant C(u, v, j - i) such that

$$\Delta(s_\alpha, v_j) - \Delta(s_\alpha, u_i) \ge C(u, v, j - i) \tag{6.14}$$

Proof: Let $<v^0, v^\iota, \ldots, v^n>$ be a path from u_i to v_j in $G^{\alpha+\beta+1}$ with $v^0 = u_i$ and $v^n = v_j$. In the following, we prove that $\Delta(s_\alpha, v^\iota) - \Delta(s_\alpha, v^{\iota-1}) \ge C^\iota$ for $1 \le \iota \le n$. The lemma follows by setting $C(u, v, j - i) = \sum^n_{\iota=1} C^\iota$.

There are two cases to consider:

a) If v^ι is a sink event of a latest constraint, we have from Equation (6.2) that

$$\Delta(s_\alpha, v^\iota) \ge \min(sd(s_\alpha, v^\iota), \Delta(s_\alpha, v^{\iota-1}) + D_{v^{\iota-1}v^\iota}).$$

Subtracting $\Delta(s_\alpha, v^{\iota-1})$ from both sides, we obtain

$$\Delta(s_\alpha, v^\iota) - \Delta(s_\alpha, v^{\iota-1}) \ge \min(sd(s_\alpha, v^\iota) - \Delta(s_\alpha, v^{\iota-1}), D_{v^{\iota-1}v^\iota})$$

As $\Delta(s_\alpha, v^{\iota-1}) \le sd(s_\alpha, v^{\iota-1})$ and $sd(s_\alpha, v^{\iota-1}) \le sd(s_\alpha, v^\iota) + sd(v^\iota, v^{\iota-1})$, it follows that

$$\Delta(s_\alpha, v^\iota) - \Delta(s_\alpha, v^{\iota-1}) \geq \min(-sd(v^\iota, v^{\iota-1}), D_{v^{\iota-1}v^\iota}).$$

Let $C^\iota = \min(-sd(v^\iota, v^{\iota-1}), D_{v^{\iota-1}v^\iota})$.

b) If v^ι is a sink event of a linear constraint, we have from Equation (6.5) that

$$\Delta(s_\alpha, v^{\iota-1}) \leq \Delta(s_\alpha, v^\iota) + sdl(v^\iota, v^{\iota-1})$$

$$\leq \Delta(s_\alpha, v^\iota) + sd(v^\iota, v^{\iota-1})$$

Since $v^\iota \in$ msources($v^{\iota-1}$), we obtain

$$\Delta(s_\alpha, v^\iota) - \Delta(s_\alpha, v^{\iota-1}) \geq -sd(v^\iota, v^{\iota-1}).$$

Let $C^\iota = -sd(v^\iota, v^{\iota-1})$.

Notice that $sd(v^\iota, v^{\iota-1})$ is a constant (independent of ι) since $j \leq \alpha - K^*$. ❑

This lemma allows us to bound the number of unfoldings within which a constraining cycle can determine the time separation. This is stated more formally in the following Lemma.

Lemma 6.3: Consider two vertices u_i and v_j of $E^{\alpha+\beta+1}$, such that $j \leq \alpha - K^*$, u_i has a path to v_j in $G^{\alpha+\beta+1}$, and u_i is on the shortest path from s_α to the *root*. If there is a constraining cycle c in G_M between $v_{j-f\varepsilon(c)}$ and v_j, then

$$f \leq \frac{\Delta(s_\alpha, v_{j-f\varepsilon(c)}) - \Delta(s_\alpha, u_{i-f\varepsilon(c)}) - C(u, v, j-i)}{\rho\varepsilon(c) - D(c)} \tag{6.15}$$

Proof: (We assume $\varepsilon^* = 1$). From Lemma 6.2, we know that

$$\Delta(s_\alpha, v_j) - \Delta(s_\alpha, u_i) \geq C(u, v, j - i)$$

Since u_i is on the shortest path from s_α to the *root*, from Lemma 5.2 and Equation (6.8), it follows that

$$\Delta(s_\alpha, v_j) \geq \Delta(s_\alpha, u_{i-f\varepsilon(c)}) + f\rho\varepsilon(c) + C(u, v, j - i) \tag{6.16}$$

From the definition of a constraining cycle, we have

$$\Delta(s_\alpha, v_j) = \Delta(s_\alpha, v_{j-f\varepsilon(c)}) + D(c),$$

combining this with Equation (6.16) results in the bound on f as given in Relation (6.15).

❑

Thus, eventually the time separation $\Delta(s_\alpha, v_j)$ is determined by those events u_i that are on the shortest path from s_α to the *root* in $G_m^{\alpha+\beta+1}$ and the constraints from u_i to v_j. Let u_i be an event on the shortest path from s_α to the *root* such that there is a path from u_i to v_j in $G^{\alpha+\beta+1}$, and let $\Delta'(s_\alpha, v_j)$ be the time separation determined in the sub-graph of $G^{\alpha+\beta+1}$ induced by the vertices in $\{v^\iota \in E^{\alpha+\beta+1} \mid$ there is a path from u_i to v^ι in $G^{\alpha+\beta+1}\}$. We have that:

$$\Delta'(s_\alpha, v_j) = \Delta(s_\alpha, u_i) + C(u, v, j - i) \text{ for } j \in [F, \alpha - \beta - K^*] \tag{6.17}$$

where C(u, v, j - i) is a constant, F is the upper bound such that Equation (6.15) holds for all j with v_j to which there are constraints from u_i. Using u_0 as a reference point and combining this identity with the bound on f in Equation (6.15), we obtain a bound on F in Theorem 6.1:

$$F \leq 1 + \max\{\frac{(s_\alpha, v_l) - \Delta'(s_\alpha, v_l}{\rho\varepsilon(c) - D(c)} v \in c \text{ is a simple cycle in } G_M \text{ with } D(c) < \rho\} \quad (6.18)$$

where l is such that there is a path from u_0, $u_0 \in U_0$, to v_ι in $G^{\alpha+\beta+1}$ for all $v \in E$, $\Delta(s_\alpha, v_\iota)$ and $\Delta'(s_\alpha, v_\iota)$ are determined in the $G^{K^*+\beta+1+\iota}$. Thus, after F unfoldings, and $\Delta(s_\alpha, v_j)$ is determined entirely by the constraints from u_0, i.e.,

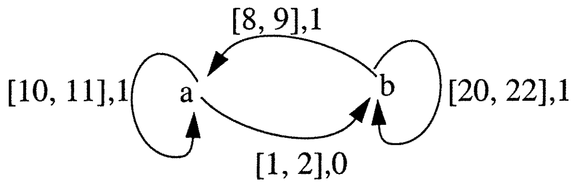
$$\Delta(s_\alpha, v_j) = \Delta'(s_\alpha, v_j), j \geq F \quad (6.19)$$

Proof (Theorem 6.1):

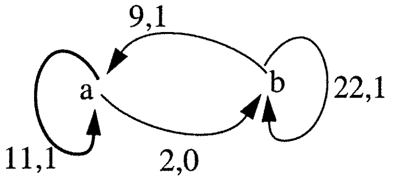
$$\begin{array}{lll} \Delta(s_\alpha, v_{j+\varepsilon^*}) - \Delta(s_\alpha, v_j) & = \Delta'(s_\alpha, v_{j+\varepsilon^*}) - \Delta'(s_\alpha, v_j), j \geq F & \text{From Equation (6.19)} \\ & = \Delta'(s_\alpha, u_{i+\varepsilon^*}) - \Delta'(s_\alpha, u_i) & \text{From Equation (6.17)} \\ & = sd(s_\alpha, u_{i+\varepsilon^*}) - sd(s_\alpha, u_i) & \text{From Lemma 5.2} \\ & = \rho\varepsilon^* & \text{From Equation (6.8)} \end{array}$$

$\square$

Example 6.2: Consider a constraint graph shown in Figure 6.2 where all the constraints are combined in max combination. Let a and b be the start and end event, respectively. We wish to determine F value.



a. A constraint graph



b. The correspondent G_M

Figure 6.2 A constraint graph G and its corresponding G_M.

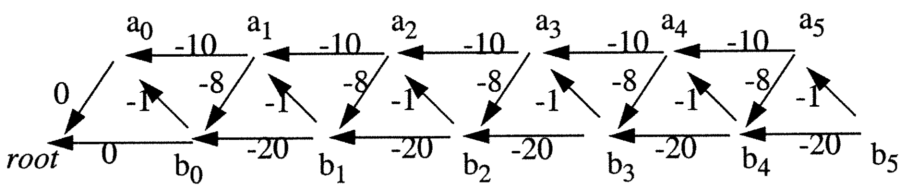


Figure 6.3 G_m^6 of the constraint graph in Figure 6.2

The following m values can be obtained in G_m^6 as shown in Figure 6.3:

$m(a_5) = 0$, $m(b_5) = \infty$, $m(a_4) = -10$, $m(b_4) = -8$, $m(a_3) = -29$, $m(b_3) = --28$, $m(a_2) =$ -49, $m(b_2) = -48$, $m(a_1) = -69$, $m(b_1) = --68$, $m(a_0) = -89$, $m(b_0) = --88$, $m(\mathit{root}) = -$ 89.

Comparing to Equation (6.8), we have that $K^* = 3$, $\varepsilon^* = 1$.

Consider the time separations from a_5. Since the shortest path from a_5 to *root* is (a_5, b_4, b_3, b_2, b_1, b_0, *root*), we know that the time separation from a_5 to these events equals to their m values. Let us look at the time separations to other events in the graph.

$\Delta(a_5, a_0) = \min(m(a_0), \Delta(a_5, \mathit{root}) + 0) = -89$.

$\Delta(a_5, a_1) = \min(m(a_1), \max(\Delta(a_5, a_0) + 11, \Delta(a_5, b_0) + 9) = \min(-69, \max(-89 + 11, -88 + 9) = -78$. $\Delta(a_5, a_1) < m(a_1)$, and $\Delta(a_5, a_1) = \Delta(a_5, a_0) + 11$, where 11 is the D(c) of the constraining cycle highlighted in Figure 6.2b. In this case, $\Delta'(a_5, a_1)$ which is determined locally by $\Delta(a_5, b_0)$ and the constraint from b_0 to a_1 equals to $-88 + 9 = -79$.

$\Delta(a_5, a_2) = \min(m(a_2), \max(\Delta(a_5, a_1) + 11, \Delta(a_5, b_1) + 9) = \min(-49, \max(-79 + 11, -68 + 9) = -59$. $\Delta(a_5, a_2) < m(a_2)$, but $\Delta(a_5, a_2)$ is determined locally by $\Delta(a_5, b_1)$ and the constraint from b_1 to a_2.

In other words, F = 1 in this example.

We now present an algorithm for determining $\Delta(s, e, \beta)$ in a causal constraint graph. In the following we need the notion of a cutset of $G^{\alpha+\beta+1}$. For a set of vertices $X \subseteq E^{\alpha+\beta+1}$, let R(X) denote the set of vertices of $G^{\alpha+\beta+1}$ reachable from a vertex in X. The set of vertices not reachable from any vertex in X is denoted as $\overline{R}(X)$. A cutset C is a finite subset of $E^{\alpha+\beta+1}$ such that every path from the *root* to a vertex in R(C) passes through some vertex in C. Furthermore, we also require that C be a cutset for any later unfolding, i.e., given a positive integer i, the set $\{v_{j+i} \mid v_j \in C\}$ must be a cutset of $G^{\alpha+\beta+i+1}$.

Recall that $\Delta(s, e, \beta)$ is defined as $\max_{\alpha \geq 0}(\Delta^{\alpha})$. Using Theorem 6.1 we show that only a finite number of unfoldings are necessary to determine Δ:

Theorem 6.2: Let G = <E, R> be a causal constraint graph. Then

$$\Delta(s, e, \beta) = \max\{\Delta^{\alpha}: 0 \le \alpha \le K^* + \varepsilon^* + F\}, \qquad (6.20)$$

where K^* and ε^* are as given in Equation (5.3), F can be calculated using Equation (6.18).

Proof: We will relate the values of $\Delta(s_{\alpha}, v_j)$ in two different finite unfolded constraint graphs $G^{\alpha+\beta+1}$ and $G^{\alpha+\beta+\varepsilon^*+1}$, $\alpha = K^* + \varepsilon^* + F$. Let C be the cutset $\{v_F: v \in E\}$, and let $\Delta^{\alpha}(v_j)$ denote the time separation from s_{α} to v_j in $G^{\alpha+\beta+1}$ and $\Delta^{\alpha+\varepsilon^*}(v_j)$ denote the time separation from $s_{\alpha+\varepsilon^*}$ to v_j in $G^{\alpha+\beta+\varepsilon^*+1}$. From Theorem 6.1, we have that

$$\text{for all } v_j \in C,\ \Delta^{\alpha+\varepsilon^*}(v_{j+\varepsilon^*}) = \Delta^{\alpha+\varepsilon^*}(v_j) + \rho\varepsilon^*.$$

Since $sd(s_{\alpha}, v_j)$ in $G^{\alpha+\beta+1}$ is equal to $sd(s_{\alpha+\varepsilon^*}, v_j) + \rho\varepsilon^*$ in $G^{\alpha+\beta+1+\varepsilon^*}$ for $j \le F$,

$$\Delta^{\alpha}(v_j) = \Delta^{\alpha+\varepsilon^*}(v_j) + \rho\varepsilon^* \qquad \text{for all } v_j \in R(C) \cup C$$

Combining these two facts, we get that

$$\Delta^{\alpha+\varepsilon^*}(v_{j+\varepsilon^*}) = \Delta^{\alpha}(v_j) \text{ for all } v_j \in C.$$

Since $sd(s_{\alpha}, v_j)$ in $G^{\alpha+\beta+1}$ is equal to $sd(s_{\alpha+\varepsilon^*}, {}_{j+\varepsilon^*})$ in $G^{\alpha+\beta+1+\varepsilon^*}$, it follows that

$$\Delta^{\alpha+\varepsilon^*}(v_{j+\varepsilon^*}) = \Delta^{\alpha}(v_j) \text{ for all } v_j \in R(C)$$

and thus $\Delta^{\alpha+\varepsilon^*}(v_{j+\varepsilon^*}) = \Delta^{\alpha}(v_j)$. ❑

This theorem leads to a straightforward algorithm: compute Δ^{α} for increasing values of α using Algorithm 6.1 and choose the maximum of Δ^{α}. The efficiency of this algorithm depends on the number of times Algorithm 6.1 is called, i.e., on the values of K^*, ε^*, and F. These numbers all depend on the delay ranges and are not polynomial in the size of the constraint graph. In most realistic constraint graphs, $\varepsilon^* = 1$ and F is a small constant (for all the 4 examples in [43], we have $F = 0$). K^* is more of a concern because it can be large if there exists a cycle G_m such that $w(c)/\varepsilon(c)$ is very close to ρ.

In practice, we can improve the performance of the algorithm by applying sufficient conditions that detect when further unfoldings cannot result in a larger value of Δ. We can observe from Equation (6.5) that $sd(s_{\alpha}, v_j)$ is an upper bound on $\Delta(s_{\alpha}, v_j)$. If $\Delta(s_{\alpha}, v_j)$ happens to be equal to $sd(s_{\alpha}, v_j)$ for all $v_j \in C$ and $\alpha \geq K^*$, then there is no reason to consider any further unfoldings since they can only make Δ^i, $i > \alpha$ smaller.

Lemma 6.4: Let C be a cutset for $G^{\alpha+\beta+1}$ where $\alpha \geq \beta_0$, If $\Delta(s_{\alpha},v_j) = sd(s_{\alpha}, v_j)$ for all $v_j \in C$ and $\alpha \geq K^*$, then

$$\Delta = \max\{\Delta^i\colon 0 \leq i \leq \alpha\}.$$

Proof: For $i \geq 0$, $\Delta(s_{k+i},v_{j+i})$, $v_j \in C$, in $G^{\alpha+\beta+i+1}$ is smaller or equal to $sd(s_{\alpha+i}, v_{j+i})$ which is equal to $sd(s_{\alpha}, v_j)$ in $G^{\alpha+\beta+1}$. That is, for all $v_j \in C$, $\Delta(s_{\alpha+i},v_{j+i})$ in $G^{\alpha+\beta+i+1}$ is smaller or equal to $\Delta(s_{\alpha},v_j)$ in $G^{\alpha+\beta+1}$. Since C is also a cutset for $G^{\alpha+\beta+i+1}$, it follows from Equations (6.2), the repetition of the m values, and the monotonicity of addition, maximization and minimization that $\Delta^{\alpha+\beta+i} \leq \Delta^{\alpha}$. ❑

The same observation can be used to determine an upper bound on Δ. Let $\Delta_{\alpha}^{\dagger}$ denote a number such that $\Delta_\alpha^\dagger \geq \max\{\Delta^i: i \geq \alpha\}$, i.e, $\Delta_\alpha^\dagger$ is an upper bound for any further unfoldings after $G^{\alpha+\beta+1}$. If $\Delta_\alpha^\dagger$ is smaller than or equal to $\Delta' = \max\{\Delta^i: 0 \leq i \leq \alpha\}$ then $\Delta = \Delta'$. To compute $\Delta_\alpha^\dagger$, we use Algorithm 6.1 except Equations (6.4) and (6.6) are not used for all $v_j \in C$ (this makes $\Delta(s_\alpha, v_j)$ equal to $sd(s_\alpha, v_j)$ for $v_j \in C$). With this optimization, the algorithm for determining the maximum time separation in a causal cyclic constraint graph can be stated as follows.

Algorithm 6.2: Maximum time separation in a causal cyclic linear-plus-latest constraint graph

```
Step 1: Pre-calculate $K^*$, $\varepsilon^*$, and F, where $K^*$ and $\varepsilon^*$ are as given in Equation (5.3),
        F can be calculated using Equation (6.18),

Step 2: $\alpha \leftarrow 0$, $\Delta \leftarrow -\infty$,

Step 3: Repeat
            $\alpha \leftarrow \alpha + 1$,
            $\Delta^\alpha \leftarrow \Delta(s_\alpha, e_{\alpha+\beta})$ in $G^{\alpha+\beta+1}$ using Algorithm 6.1
            $\Delta \leftarrow \max\{\Delta, \Delta^\alpha\}$
            if there exists a cutset $C$ in $G^{\alpha+\beta+1}$ and $\alpha \geq K^*$ such that $\forall v_j \in C$: $sd(s_\alpha, v_j)$
$= \Delta(s_\alpha, v_j)$ then
                return $\Delta$
            $\Delta_\alpha^\dagger \leftarrow$ Upperbound($s_\alpha$, $e_{\alpha+\beta}$) in $G^{\alpha+\beta+1}$
until $\Delta \geq \Delta_\alpha^\dagger$ or $\alpha = K^* + \varepsilon^* + F$
return $\Delta$
```

Example 6.3: We continue with the repeated microprocessor READ example, to determine the maximum time separations $\Delta 1$, $\Delta 2$ in Equations (3.11). Algorithm 6.2 yields $\Delta 1 = 0$ and $\Delta 2 = 0$, i.e., for all consistent timing assignments τ,

$\forall\alpha \geq 0, \tau(e_{11}, \alpha) - \tau(e_{10}, \alpha + 1) \leq 0 \Rightarrow \qquad \forall\alpha \geq 0, \tau(e_{11}, \alpha) \leq \tau(e_{10}, \alpha + 1)$ and

$\forall\alpha \geq 0, \tau(e_{15}, \alpha) - \tau(e_{14}, \alpha + 1) \leq 0 \Rightarrow \qquad \forall\alpha \geq 0, \tau(e_{15}, \alpha) \leq \tau(e_{14}, \alpha + 1)$.

That is, there will not be any overlap on the activation and deactivation of ACK and DATA signals in the specification.

6.4 Sufficient Causal Conditions

Algorithm 6.2 assumes that the constraint graph is causal, i.e., all blocks in the unfolded constraint graphs are past-dominated and have well-defined triggers. The past-dominated property can be checked using the causality conditions (2 and 3) in Definition 6.2. For convenience, we restate them here:

Theorem 6.3[50]: A block B in a finite constraint graph is past-dominated if

1. All events in B occur later than all the triggers of B (well-defined triggers), and

2. $\forall$ $u_1, u_2 \in$ trigs(B), $\Delta(u_1, u_2) \leq \Delta_{local}(u_1, u_2)$, where $\Delta_{local}(u_1, u_2)$ is the maximum time separation determined by only the local constraints of B.

Notice that if a block satisfies these two conditions, then all its triggers are well-defined and the block is past-dominated. Every latest block is inherently past-dominated since all constraints from the triggers to the events in the block are precedence constraints and are combined by max combinator. For linear blocks, we can use Algorithm 6.2 to verify whether the two conditions in Theorem 6.3 hold.

Theorem 6.4: Let G = (E, R) be a finite constraint graph with a block partition P. A linear block B $\in$ P is past-dominated if

1. $\forall$ u $\in$ trigs(B), $\forall$v $\in$ B, $\Delta_{com}(v, u) \leq 0$ and

2. $\forall$ $u_1, u_2 \in$ trigs(B), $\Delta_{com}(u_1, u_2) \leq \Delta_{local}(u_1, u_2)$,

where $\Delta_{com}(u_1, u_2)$ is the maximum time separation computed using Algorithm 6.2.

Proof: We show that the two conditions of Theorem 6.4 imply those in Theorem 6.3. Let u, v be two vertices of G. We shall show that the value $\Delta_{com}(u, v)$ computed by Algorithm 6.2 which is an exact separation if the graph is causal becomes an upper bound on the separation when the graph is not causal. This is because Δ_{com} is determined from Equations (6.2) and (6.6) while the real maximum time separation in a non-causal graph would be determined from Equation (6.1). The later contains more terms in the min function than Equations (6.2) and (6.6), and also sdl(u, v) $\geq$ sd(u, v). Therefore,

$$\Delta_{com}(u, v) \geq \Delta(u, v) \tag{6.21}$$

Combining Equation (6.21) with Condition 1 of Theorem 6.4, it follows that

$$\Delta(u, v) \leq \Delta_{com}(u, v) \leq 0 \tag{6.22}$$

Consequently, for all consistent timing assignments τ, $\tau(u) \leq \tau(v)$ is true. that is, condition 1 of Theorem 6.3 is satisfied.

Similarly, by combining Equation (6.21) with Condition 2 of Theorem 6.4, it follows that

$$\forall\ u_1, u_2 \in \text{trigs(B)}, \Delta_{com}(u_1, u_2) \leq \Delta_{local}(u_1, u_2), \tag{6.23}$$

and thus Condition 2 of Theorem 6.3 is also satisfied. ❑

Theorem 6.5: Let G = <E, R> be a constraint graph with a block partition P. A linear block $B \in P$ in $G^{\alpha+\beta}$ for $\alpha \geq 0$, is past-dominated if

1. $\forall u \in trigs(B)$, $v \in B$, $\Delta_{com}(v, u) \leq 0$ and

2. $\forall u_1, u_2 \in trigs(B)$, $\Delta_{com}(u_1, u_2) \leq \Delta_{local}(u_1, u_2)$,

where Δ_{com} is the maximum time separation computed under the assumption that the system is causal (i.e., the value determined by Algorithm 6.2) and Δ_{local} is the maximum separation considering only the local constraints of B.

Proof:

From the definition of Δ in a constraint graph in Equation (3.7), it follows that

$$\forall \alpha \geq \beta_0, \Delta^{\alpha} \leq \Delta.$$

Thus the theorem follows by applying Theorem 6.5 to each unfolding $G^{\alpha+\beta}$.

❑

The amount of calculation needed to verify that a cyclic constraint graph is causal can be reduced using the following observation.

Corollary 6.1: If all local constraints of a linear block B are precedence constraints, then B has well-formed triggers.

Example 6.4: To ensure that the result of the timing analysis in Example 6.3 is valid, we use Algorithm 6.2 to check that the block partition is past-dominated and has well-formed triggers. Only block $B_2 = \{e_{10}, e_{14}\}$ contains concurrent constraints. The only trigger of this block is e_7. Condition 1 of Theorem 5 is satisfied, because the results of the two separation analyses are non-positive.

$$\Delta_{com}(e_{10}, e_7) = 0 \text{ and } \Delta_{com}(e_{14}, e_7) = -10$$

Condition 2 of Theorem 6.5 does not apply since block B_2 has only a single trigger event.

Therefore, the system is causal and the time separations computed in Example 6.3 under the assumption that the system is causal are reachable maximum time separations, and there will be no conflict between the back-to-back cycle transitions on signals ACK and DATA.

6.5 Conclusion

In this Chapter, we addressed the problem of determining the maximum time separation between two events in a constraint graph which contains both linear and latest constraints. We developed an exact algorithm for solving this problem for the class of causal constraint graphs, i.e., those that have a past-dominated block partition with well-defined triggers. We also stated a sufficient condition for checking the causality of the cyclic specification.

In the last five chapters, we have discussed the compatibility and other safety timing property verification of interfaces specified by loop over a leaf HAAD. We solved the verification problems using maximum time separation between events in constraint graphs transformed from the specifications.

In the following two chapters, we will verify implementation of interface controllers against their specifications where we will also see the application of computing the maximum time separations.

Chapter 7

Verification of Real Time Controllers Against Timing Diagram Specifications

In the last five Chapters, we developed algorithms of computing the maximum time separation between events in constraint graphs transformed from Loop over a leaf HAAD specification containing linear-only, linear-plus-latest timing constraints. The algorithms can be applied directly in verifying compatibility and other safety timing properties of interface specifications. In the next two Chapters, we are interested in verifying correctness of interface controller implementations with respect to their leaf HAAD specifications with linear-only timing constraints.

7. 1 Introduction

In the design of Systems on a Chip (SoC) from predesigned building blocks, it is important to assure that these blocks can communicate correctly. This means that much effort is spent on designing and verifying bus controllers and other communication control logic. Even if the processor bus protocol is asynchronous, the controllers of such a bus in the connecting devices are often designed as synchronous Finite State Machines (FSM), operating on synchronized input signals from the bus. Timing simulation is the usual method for verifying that the controller can operate in the full range of the bus protocol. This is generally far from satisfactory due to the large number of different timing situations that could exist on the input signals. To perform this verification exhaustively, yet without the full explicit enumeration of all situations, we present in this Chapter a method for verifying whether a pseudo-synchronous (sampled input/output) finite-state machine (FSM) implementation of a real-time controller satisfies its timing diagram (TD) specification. That is, given a clock frequency and a TD specification, we check whether the FSM implementation (given as an RTL verilog or VHDL code) always produces outputs within the time constraints stated in the specification provided that the inputs meet the assumptions as stated in the specification.

Although there are many published results regarding static timing verification of synchronous sequential circuits, most of them are concerned with set-up and hold time checks on sequential elements (flip-flops, latches and memories etc.). In our work, we are interested in the timing information as determined by the functionality of the circuit, i.e., a mixture of timing and behavioural verification over a number of clock cycles. This kind of verification could potentially be carried out using models based on Timed Automata (TA) [2] or timed Petri Nets [40], however, the TA models and the accompanying verification techniques based on reachability analysis of the derived region graphs are unnecessarily difficult and complex in this practical context of verifying realistic RTL designs of hardware interface controllers. We show that constraint logic programming [44][66] provides an interesting alternative for solving the stated problem.

Clocksin [26] at the University of Cambridge used logic programming to carry out simulation of synchronous sequential circuits. He used rule based unit clauses which resemble standard truth tables to model transitions of the circuit to be simulated. As we shall show, such an approach leads to exponential time explosion, because most of the possible FSM executions are enumerated. Instead, we use constraints rather than rule based clauses to model the execution of sequential circuits in real time. This consists of unrolling the machine over a sufficient number of clock cycles (as determined by algorithms we developed in Chapters 2 to 6) to cover the range of time implied by the TD specification, expressing the unrolled instances of the FSM as a series of constraints. We then link these constraints to the timing constraints from the TD using a set of automata that convert between event occurrences on signals in the TD and signal levels required/ produced by the controller FSM. In addition, since we inherently deal with uncertainty intervals as to the time of occurrence of events, we can carry out this verification under variation of the clock frequency.

The original contributions of this chapter are:

- The modeling of a TD specification as communicating "TD" automata which accept event traces respecting the TD constraints (here we consider TD specifications with linear-only constraints); their states indicate the signal values of the waveforms that are read/output by the FSM controller. These machines permit establishing the correspondence between events and signal values.

- The representation of the finite unfolding of the implementation FSM and the finite execution of the TD automata and timing requirements in the form of constraints.

- Formulating the FSM versus TD timing verification problem as a consistency check of a constraint system.

- Acceleration of the solution by the addition of redundant constraints to the constraint system.

The representation of the set of states, the output traces of the TD automata, and the implementation FSM is compact (linear with time) in terms of constraints, while the number of traces thus characterized may grow exponentially with time. Although we verify only finite behaviours as described by the leaf timing diagram of HAAD specification, most realistic bus protocols usually return to an initial state after executing a particular operation cycle. We can thus verify that at the end of the finite trace the final state of the automaton and the maximum separations of the last events on each port correspond to the initial state. The approach can be compared to model checking on finite computations derived from a Kripke structure using a satisfiablility procedure [7].

The chapter is organized as follows: In Section 7. 2, we introduce pseudo-synchronous (sampled input/output) finite-state machine (FSM) real time controllers and TD specifications, and we define the verification problem. In Section 7. 3, we present details about our modeling techniques and the verification method. In Section 7. 4, we give the implementation of the verification procedure in CLP(BNR) Prolog [66]. In Section 7. 5, we discuss how to improve the efficiency of the verification procedure and we compare the results of our procedure with one that uses a set of rule-based Prolog clauses to define the executions of the machine. In Section 7. 6, we present experimental results of the two procedures for an example and present conclusions.

7. 2 Problem Definition

In this section, we first introduce a pseudo-synchronous controller and its TD specification, then we state the verification problem.

7.2.1 A pseudo-synchronous controller

A pseudo-synchronous controller is usually implemented as a synchronous FSM (sequential circuit) as shown in Figure 7.1. It consists of a combinational logic block and registers holding the values of state variables. The combinational logic is fed by the sampled (synchronized) primary inputs (PI) and by the outputs of the state registers. We assume that there is a register on every output signal which filters possible glitches on the outputs of the combinational circuit (i.e., it can be viewed as a Moore machine). We further assume that all flip-flops are driven by the same clock, and that the clock frequency and the delays in the circuit are such that set-up and hold times are satisfied on all flip-flops (except of course on the flip-flops that sample the asynchronous inputs).

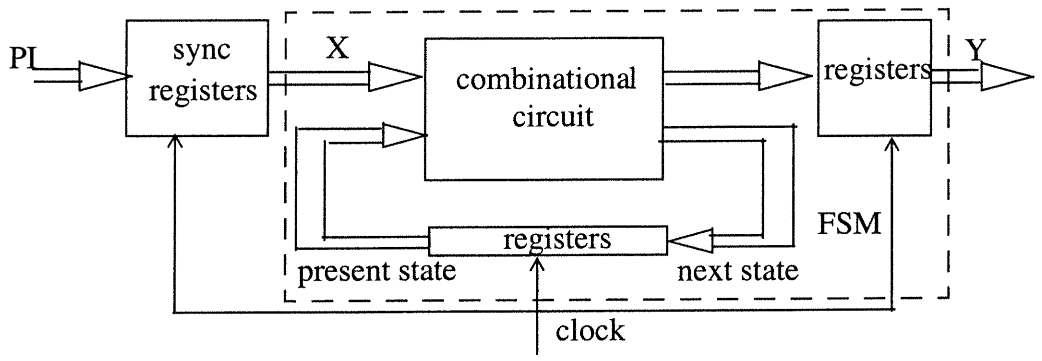


Figure 7.1 An implementation of a pseudo-synchronous controller.

We will use the Moore FSM in Figure 7.2 as an example throughout the Chapter, and adapt the model to a more realistic situation in Chapter 8.

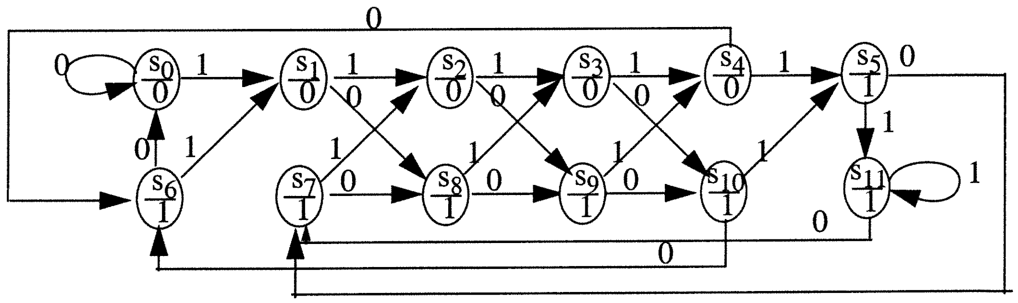


Figure 7.2 Example FSM.

7.2.2 Timing Diagrams

Timing Diagrams (TD) are as defined in Definition 2.1 on page 20. In this chapter, we denote a timing constraint from a source event e_i to a sink event e_j as $c_{ij} = (e_i, e_j, [T_{ijmin}, T_{ijmax}]) \in A \cup C$, $T_{ijmin} \le T_{ijmax}$, $T_{ijmin}, T_{ijmax} \in \Re$. Then corresponding to Equation (2.1), the timing constraint c_{ij} delimits the relationship between occurrence times t_i and t_j of e_i and e_j.

$$T_{ijmin} \leq t_j - t_i \leq T_{ijmax} \tag{7.1}$$

In a TD specification, when more than one constraints sink at the same event e_j, the occurrence time t_j is determined by a combination of these constraints. As defined in Definition 2.2 to Definition 2.4 on page 21, there are three types of combinations in a leaf HAAD specification. In this Chapter, we only consider one of them, conjunctive combination, i.e., all of the constraints must be satisfied simultaneously.

To illustrate our method, we shall use the specification TD as shown in Figure 7.3 for the design in Figure 7.2. It is composed of two waveforms {X, Y}, four events $\{e_{X1}, e_{X2}, e_{Y1}, e_{Y2}\}$, two assume constraints ((*origin*, e_{X1}, [1, 16]), (e_{Y1}, e_{X2}, [61, 78])), two commit constraints ((e_{X1}, e_{Y1}, [70, 80]), (e_{X2}, e_{Y2}, [70, 80])). *Origin* indicates the initial state (values of the signals) at the time origin. We set arbitrarily $t_{origin} = 0$.

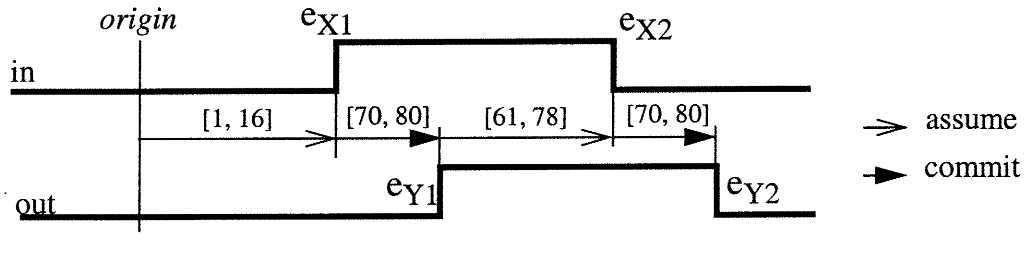


Figure 7.3 A TD specification for the example in Figure 7.2

7.2.3 The Verification Problem

In this section, we define models of the TD specification and the implementation FSM, and then we state the verification problem. The relatively complex model is due to the fact that TDs are event based, while the pseudo-synchronous sequential circuits operate on signal levels. Therefore, much of the definitions that follow are needed to establish a relation between the two representations - event based and level based.

Definition 7.1: A waveform WV is a vector (wv_0, wv_1, …, wv_m) of values, where $wv_i \in B$, $B = \{0, 1\}$, such that their indices 0, 1, …, m increase monotonically with time and the change from wv_i to wv_{i+1} denotes the i-th transition (event) of the waveform, wv_0 is the initial value of the signal at *origin*.

Definition 7.2: A waveform transition automaton WTA = (V, E, G, v_0, v_f) associated with a waveform WV is a 5-tuple where

- $V = \{v_i\} = \{0, 1, \dots, m\}$ is the set of states represented by the set of indexes of the waveform vector WV;
- $E = \{e_1, e_2, \dots, e_m\}$ is the set of events on the waveform;
- $G: V \times E \rightarrow V$ is the state transition function defined as $G(i, e_{i+1}) = i+1$, $0 \leq i < m$;
- $v_0 = 0 \in V$ is the initial state;
- $v_f = m \in V$ is the final state.

Definition 7.3: With each occurrence of an event $e_i \in E$ in a waveform transition automaton WTA, we associate an occurrence time t_i, $t_i \in \Re$.

When WTA is in state i, the value of the waveform is wv_i.

Definition 7.4: A TD transition automaton, TDTA = (V, E, G, v_0, v_f) is defined as the Cartesian product of the WTAs of the waveforms in the TD. Without loss of generality, consider a TD with two waveforms WTA_1 and WTA_2. $TDTA = WTA_1 \times WTA_2$ is defined as:

- $V = V_1 \times V_2$;

- $E = E_1 \cup E_2$ is the set of events in the TD;

- $G: V \times E \rightarrow V$ is the next state transition function defined as

$$G((i, j), e_{1,i+1}) = (i+1, j) \text{ iff } G_1(i, e_{1,i+1}) = i+1 \text{ in } WTA_1,$$

$$G((i, j), e_{2,j+1}) = (i, j+1) \text{ iff } G_2(j, e_{2,j+1}) = j+1 \text{ in } WTA_2;$$

- $v_0 = (0, 0) \in V$ is the initial state,

- $v_f = (m_1, m_2) \in V$ is the final state.

The occurrence time of a transition (event) in TDTA is the occurrence time of the event in the respective WTA as defined in Definition 7.3.

The set of events E can be partitioned into E_{IN} and E_{OUT} such that events in E_{IN} are input events while events in E_{OUT} are output events, $E_{IN} \cup E_{OUT} = E$ and $E_{IN} \cap E_{OUT} = \varnothing$. V can be partitioned into V_{IN} and V_{OUT} such that the waveform values in WV corresponding to V_{IN} are from the input waveforms, and those corresponding to V_{OUT} are from the output waveforms, $V = V_{IN} \times V_{OUT}$.

For example, in the case of the timing diagram in Figure 7.3 we get:

Waveform X: $WV_X = (0, 1, 0)$; $V_X = \{0, 1, 2\}$; $v_{X0} = 0$; $v_{Xf} = 2$; $G = ((0, e_{X1}) \text{->} 1, (1, e_{X2}) \text{->} 2$. Similarly for waveform Y. The product of these two automata forms the TDTA, however, we shall never construct this automaton explicitly, but rather describe its behavior by the concurrent execution of its constituent WTAs, synchronized through the timing constraints.

Definition 7.5: A timed trace of a TDTA, $tr_{TD} = ((e_1, t_1), (e_2, t_2), \ldots, (e_n, t_n))$ is a sequence of pairs (event e_i, occurrence time of e_i), $0 \leq i \leq n$, such that

$v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \rightarrow \ldots \xrightarrow{e_n} v_n$, $v_i \in V$, $G(v_i, e_{i+1}) = v_{i+1}$, $t_{i+1} \geq t_i$, $0 \leq i < n$, $v_n = v_f$, and the occurrence times $t_1, \ldots, t_n$ satisfy the timing constraints in $A \cup C$. Let $TR_{TD} = \{tr_{TD}\}$ be the set of all possible[1] timed traces of a TDTA.

For example, a possible timed trace of the TDTA in our example is as follows ($t_{origin} = 0$): tr = ((e_{X1}, 10),(e_{Y1}, 80), (e_{X2}, 150), (e_{Y2}, 220)). It corresponds to the following run of the TDTA: (0, 0) -e_{X1}-> (1, 0) -e_{Y1}-> (1, 1) -e_{X2}-> (2, 1) -e_{Y2}-> (2, 2). Depending on the occurrence time assignment to the events (satisfying the TD timing constraints), there may be infinitely many different timed traces of the same (untimed) run (e_{X1}-> e_{Y1}-> e_{X2}-> e_{Y2}) of the TDTA.

We shall relate the occurrence times of events t_i, i = 1, ..., with the i-th change of the state of the WTA, thus obtaining the related waveform values needed by the controller, while constraining the occurrence time by the timing constraints on t_i. Next we shall define the controller FSM and its execution in time given the clock period P and its Clk-th occurrence in time.

Definition 7.6: A sampled input/output Moore state machine controller implementing the protocol specified by a timing diagram is defined as M = (S, WV_{IN}, N, O, s_0), where

- S is the set of states;
- $WV_{IN} = (wv_{IN}(j\_in))$, $1 \leq j\_in \leq n\_input$, is a vector of the input waveform values of the related TDTA, where $wv_{IN}(j\_in) \in WV_{IN}(j\_in)$, $WV_{IN}(j\_in)$ is the correspondent input waveform defined in Definition 7.1 and n_input is the number of input waveforms in the TD

[1]: In a trace tr_{TD}, it is possible that there are sets of events $\{e_i\}$ such that all have the same occurrence time t; to simplify the presentation, we shall assume that in TR_{TD} there are traces behaviorally equivalent to tr_{TD} consisting of all the possible orders of the events in $\{e_i\}$ that satisfy $t_{j+1} \geq t_j$

specification;

- N: $S \times WV_{IN} \rightarrow S$ is the state transition function;

- O: $S \rightarrow WV_{OUT} = (wv_{OUT}(j\_out))$, $1 \leq j\_out \leq n\_output$, is the output function, where $wv_{OUT}(j\_out) \in WV_{OUT}(j\_out)$, $WV_{OUT}(j\_out)$ is the correspondent output waveform defined in Definition 7.1 and n_output is the number of output waveforms in the TD specification;

- $s_0 \in S$ is the initial state.

In the example shown in Figure 7.2, n_input = 1, $S = \{s_0, \ldots, s_{11}\}$, $WV_{IN} = \{0, 1, 0\}$, N is as in the following table 12,

Table 7.1: Next State Function of the FSM in Figure 7.2

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}
$wv_{IN} = 0$	s_0	s_8	s_9	s_{10}	s_6	s_7	s_0	s_8	s_9	s_{10}	s_6	s_7
$wv_{IN} = 1$	s_1	s_2	s_3	s_4	s_5	s_{11}	s_1	s_2	s_3	s_4	s_5	s_{11}

and the output function O is as follows: the output value is 0 if the current state $s \in \{s_0, \ldots, s_4\}$ and is 1 if the current state $s \in \{s_5, \ldots, s_{11}\}$. s_0 is the initial state.

Definition 7.7: An input/output sequence of machine M of length k, seq = $(((wv_{0IN}(j\_in)), (wv_{0OUT}(j\_out))), (((wv_{1IN}(j\_in)), (wv_{1OUT}(j\_out))), \ldots, (((wv_{kIN}(j\_in)), (wv_{kOUT}(j\_out)))$, where $(wv_{iIN}(j\_in)) \in WV_{IN}$, $(wv_{iOUT}(j\_out)) \in WV_{OUT}$, $1 \leq j\_in \leq n\_input$, $1 \leq j\_out \leq n\_output$, $0 \leq i \leq k$, is a sequence of pairs of input output values of M from the initial state s_0 such that $s_0 \xrightarrow{(wv_{0IN}(j\_in))} s_1 \xrightarrow{(wv_{1IN}(j\_in))} s_2 \xrightarrow{(wv_{kIN}(j\_in))} \cdots \rightarrow s_{k+1}$, $s_{i+1} = N(s_i, (wv_{iIN}(j\_in)))$, $i \geq 0$; $(wv_{iOUT}(j\_out)) = O(s_i)$.

Definition 7.8: The arrival time of machine M to state s_j, $j > 0$, is the moment when input signal values change from $(wv_{(j-2)IN}(j\_in))$ to $(wv_{(j-1)IN}(j\_in))$ and the machine M is experiencing a state transition from s_{j-1} to s_j.

Given a period P of the clock of machine M, the arrival time of M into state s_j is given by $\tau_j = j * P + \delta$, $j > 0$, where $0 \leq \delta < P$ is the phase shift delay of the clock generator relative to *origin*. At time t_{origin} (= 0), machine M is in state s_0.

Definition 7.9: The controller machine M samples input signal values in each clock cycle. In a specific clock cycle, if the sampled value on an input waveform is different from that in the previous clock cycle, then an input event is detected at the beginning of that clock cycle by machine M on the corresponding input port.

An input event $e_i \in E_{IN}$ is detected by machine M in the transition $s_j \xrightarrow{(wv_{jIN}(j\_in))} s_{j+1}$ if $(wv_{j-1IN}(j\_in)) \neq (wv_{jIN}(j\_in))$, and the difference in values of the signal corresponds to e_i. The observed occurrence time of an input event e_i is thus $\tau_i = (j + 1) * P + \delta$, rather than t_i.

Definition 7.10: The controller machine M generates outputs in each clock cycle. In a specific clock cycle, if the signal value on an output generated is different with that in the previous clock cycle, then an output event is produced at the beginning of that clock cycle by machine M on the corresponding output port.

An output event $e_o \in E_{OUT}$ is produced in state s_{j+1} if $O(s_j) \neq O(s_{j+1})$, the difference in signal values on the corresponding output port identifies the output event e_o. The produced occurrence time of an output event e_o is $\tau_o = (j + 1) * P + \delta$. It is equal to the actual occurrence time t_o if for now we ignore output delays in combinational circuitry and interconnects. To simplify the notation in what follows, we assume δ

= 0.

Definition 7.11: A controller M accepts events of a TD automaton if the controller M detected all the input events E_{IN} and observed all the output events E_{OUT} of the TDTA.

Definition 7.12: A timed trace tr = $((e_1, \xi_1), (e_2, \xi_2), \ldots, (e_n, \xi_n))$, of a controller M that accepts events of a TD automaton is a sequence of pairs (e_i, ξ_i), $\xi_{i+1} \geq \xi_i$, $i > 0$, such that if e_i is an input event, then $\xi_i = t_i$; else if e_i is an output event, then $\xi_i = \tau_i$, i.e., the time when e_i is produced by machine M under the observed inputs. Let TR = {tr} be the set of all possible timed traces tr.

TR is different from TR_{TD} in that the occurrence times of the output events in TR are those produced by the implementation FSM under the sampled inputs of the input signals in the TD.

The verification problem we wish to solve can now be stated as follows:

Given a TD specification (a leaf HAAD specification with linear-only timing constraints) and a sampled input/output FSM implementation, does the protocol (the occurrence of events that matches what is expected) and timing of the output events in TR satisfy the commit constraints C in the TD, i.e., given the assumptions on input timing, is the set of traces produced by the controller included in the set of traces specified by the Timing Diagram, $TR \subseteq TR_{TD}$?

7. 3 The Verification Method

In this section, we present our modeling and verification method using a system of constraints. Each set of traces TR and TR_{TD} is characterized by a set of constraints derived from the next-state functions, event identification as state changes, and the timing constraints from the TD specification. The two sets share the assume constraints from the TD specification. Therefore, to verify that the trace produced by the controller is within TR_{TD}, we verify that the timing of the produced output events is not outside that permitted by the commit constraints in the TD specification. That is, we check that the system of constraints characterizing TR and the negation of the commit constraints form an inconsistent system of constraints.

7.3.1 The Set of Timed Traces of TDTA

From Definition 7.5, the set TR_{TD} of timed traces of the timing diagram is produced by generating all events in the TD transition automata and imposing assume and commit constraints on the occurrence times of the generated events. To describe the generation of all the events in the TD, we define executions of a WTA and a TDTA.

A sampled execution of a waveform transition automaton WTA starts at time $t = 0$, and at that time the WTA is at its initial state 0. The value of time increases continuously. In a clock cycle Clk, i.e., the time interval $[Clk * P, (Clk+1)*P)$, the WTA makes a transition from v_{i-1} to v_i if e_i occurs, i.e., $t_i \in [Clk * P, (Clk+1)*P)$. Otherwise, the WTA stays at v_{i-1}. An execution $\sigma = v_0v_1...v_k$ of the WTA in k clock cycles is accepted by the WTA if a sequence of events $e_0, e_1, ..., e_m, e_i \in E, 0 \leq i \leq m$ occurred in the k clock cycle, in that case, $v_k = m$. In this case, the state sequence of WTA in the execution is 0, ..., m, and the corresponding sequence of waveform values is $wv_0, ..., wv_m$, which is exactly the same as the value sequence on the waveform with which the WTA is associated. At the end of an accepted execution, all the events on the waveform have occurred. We give formal definitions of an execution and an accepted execution of a WTA in k clock cycles as follows:

Definition 7.13: An execution $\sigma = v_0v_1...v_k$ of a WTA in k clock cycles is a sequence of k state values of the WTA, where $v_0 = 0$, and v_{Clk}, $Clk > 0$ is

$$v_{Clk} = \begin{cases} v_{Clk-1} & \text{if } \forall e_j \in E, t_j \notin [Clk \times P, (Clk+1) \times P] \\ v_{Clk-1} + 1 & \text{if } \exists e_j \in E, t_j \in [Clk \times P, (Clk+1) \times P] \end{cases} \qquad (7.2)$$

Definition 7.14: An execution $\sigma = v_0 v_1 \ldots v_k$ of a WTA in k clock cycles is accepted iff $v_k = m$, where m is the number of events in E of the WTA.

A sampled execution of a TDTA modeling all the waveforms in the TD specification starts at time $t_{origin} = 0$, at that time all the automata are in the initial states. An execution of the TDTA is accepted in k clock cycles if the execution of each WTA is accepted in k clock cycles.

It follows that at the end of an accepted execution of a TDTA, all the WTAs are in their final states and all the events in the TD have occurred. We thus can extract a timed trace $tr_{TD} = \{(e_1, t_1), \ldots, (e_n, t_n)\}$ from an accepted execution of the TDTA such that

$$v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \rightarrow \ldots \xrightarrow{e_n} v_n, v_i \in V, G(v_i, e_{i+1}) = v_{i+1}, t_{i+1} \geq t_i, 0 \leq i < n, v_n = v_f.$$

The execution of a WTA in a clock cycle Clk can be represented by the following constraints:

$$[((t_i < (Clk+1) * P) \text{ and } (t_i \geq Clk * P) \text{ and } (vp = i - 1)) \rightarrow (vn = i)] \text{ and}$$

$$[(\text{not}((t_i < (Clk+1) * P) \text{ and } (t_i \geq Clk * P)) \text{ and } (vp = i - 1)) \rightarrow (vn = i - 1)] \quad (7.3)$$

where vp and vn are the current state and the next state variables of the WTA, respectively, and '→' is an implication.

The constraints whose solutions are the traces in TR_{TD} in k clock cycle (we will discuss the determination of k at the end of Section 7.3.2) can thus be described as follows:

1. Initialize $t_{origin} = 0$, $Clk = 0$, $vp_1 = 0, \ldots, vp_w = 0$, where vp_i, $1 \le i \le w$, is the current state of the WTA modeling waveform i, and w is the number of waveforms in the TD

2. In a clock cycle Clk do

   2.1 If $Clk < k$, then include a constraint for each event in the TDTA in the form of (7.3).

   2.2 If $Clk = k$, then $vp_1 = m_1, \ldots, vp_w = m_w$, where $m_1, \ldots, m_w$ are the final states (the numbers of events on the w waveforms), go to Step 4.

3. Increment Clk by 1, assign the current state values of the WTAs as the next state from the last cycle and goto Step 2.

4. Include timing constraints for $A \cup C$ in the form of relation (7.1).

7.3.2 The Set of Timed traces TR

From timed trace of the TD specification TR_{TD} characterized by the constraint system described in Section 7.3.1, we can get the sampled input sequences observed by machine M as follows:

$$\forall e_i \in E_{IN}, (t_i < Clk * P) \text{ and } (t_i \ge (Clk - 1) * P) \rightarrow (\tau_i = Clk * P) \qquad (7.4)$$

We feed the sampled inputs (observed at τ_i) to the FSM to generate the outputs of machine M. To model the execution of machine M in k clock cycles, we model the next state and the output functions of the FSM using constraints. For example, the next state function of the Moore machine in Figure 7.2 can be represented as constraints between the current state (SP), the next state (SN) and the input values of the FSM as shown in Figure 7.4.

$$[((SP = 0) \text{ and } (input\_value = 0)) \rightarrow (SN = 0)] \text{ and}$$
$$[((SP = 0) \text{ and } (input\_value = 1)) \rightarrow (SN = 1)] \text{ and}$$
$$\dots$$
$$[((SP = 11) \text{ and } (input\_value = 0)) \rightarrow (SN = 7)] \text{ and}$$
$$[((SP = 11) \text{ and } (input\_value = 1)) \rightarrow (SN = 11)]$$

Figure 7.4 Constraints modeling the state transition of the FSM.

The input waveform values are in one to one correspondence with the states in the WTA model of the input waveform. In our example, if WTA model of X is at state 0 or 2 then the input value is 0, otherwise, the input value is 1. This can be expressed as the following constraint (XP is the current state of the WTA):

$$[((XP \neq 1) \rightarrow (input\_value = 0)] \text{ and}$$
$$(XP = 1)) \rightarrow (input\_value = 1) \tag{7.5}$$

The output of the Moore machine is determined by the current state of the FSM. In the example, the constraints characterizing the output values are:

$$[((SP \geq 0) \text{ and } (SP \leq 5) \rightarrow (output\_value = 0)] \text{ and}$$
$$[((SP \geq 6) \text{ and } (SP \leq 11)) \rightarrow (output\_value = 1)] \tag{7.6}$$

An output event is produced if the current output value of the FSM is different from the value in the last cycle. It can be detected by using the current and the next states of the WTA modeling the output waveform. If an output event can be produced in clock cycle Clk, the occurrence time of the event is $\tau_i = (Clk + 1) * P$. In the example shown in Figure 7.3, event e_{Y1} (respectively e_{Y2}) is produced if the current state YP of WTA is 0 (1) and the next state YN of WTA is 1 (respectively 2). This can be expressed as:

$$[((YP = 0) \text{ and } (YN = 1)) \rightarrow (\tau_1 = (Clk + 1) * P)] \text{ and}$$

$$[((YP = 1) \text{ and } (YN = 2)) \rightarrow (\tau_2 = (Clk + 1) * P)] \qquad (7.7)$$

where YN can be determined from XP and the output value by the constraints shown in Figure 7.5

$$[((YP = 0) \text{ and } (output\_value = 0)) \rightarrow (YN = 0)] \text{ and}$$

$$[((YP = 0) \text{ and } (output\_value = 1)) \rightarrow (YN = 1)] \text{ and}$$

$$\ldots$$

$$[((YP = 2) \text{ and } (output\_value = 0)) \rightarrow (YN = 2)]$$

Figure 7.5 Constraints modeling the state transitions of WTA Y.

In general, there are more inputs/outputs of the FSM and TD, constraints modeling the state transition of the FSM, input signal values based on input WTA states, output function of the FSM, detection of an output events should be similar to what we have in the above example. For instance, if there are more inputs and states of the FSM, then the constraints modeling the state transition cab be obtained by enumerating all SP values under all possible input values such as "(input1_value =0) and (input2_value = 0) and …". the result constraint is similar to Figure 7.4.

The constraints that characterize TR in our example can thus be generated as follows:

1. Initialize $t_{origin} = 0$, Clk = 0, XP = 0, YP = 0, SP = 0, input_value = 0

2. In a clock cycle Clk

    2.1 If Clk < k, then add the constraints modeling the input value from Relation (7.5), the constraints modeling the next state functions of the FSM and WTAs (Figure 7.4, Equation (7.3) and in Figure 7.5 in the example), the constraints modeling the output function (Relation (7.6) in the example), and the constraints modeling the detection and occurrence time assignment of the detected output events (Relation (7.7) in the example).

    2.2 If Clk = k, then all the WTAs have reached their final states, go to Step 4.

3. Increment Clk by 1, assign the current state values of the WTAs modeling the TD and the FSM as the next state calculated in Step 2.1 and repeat Step 2.

4. Include assume constraints A (Relation (7.1) in the example).

We must verify the traces over a large enough number of clock cycles k (denoted as *Maxcycle*) within which all events in the TD will have been observed (IN events) and produced (OUT events) by the controller FSM for any possible timing of input events as specified by the TD. *Maxcycle* can be calculated by the maximum time separation from the *origin* to the last event in the TD (see e.g., [37] for solution techniques). The set of timed traces TR is thus characterized by constructing a constraint system CS using the above four steps, where k is replaced by *Maxcycle.*

7.3.3 Putting It All Together - The Verification Procedure

We verify $TR \subseteq TR_{TD}$ in the following way: For each commit constraint in C, we include the complement of the commit constraint in CS characterizing the timed trace set TR and then check the consistency of the constraint system by unifying τ_i with t_i (adding a constraint $\tau_i = t_i$ in the constraint system) when e_i is an output event (i.e., setting the occurrence time of e_i to that produced by the FSM). For the commit constraint to the last output event in the TD, the constraint system characterizing the TR is described in Section 7.3.2 where all the assume constraints in TD are included. For all the other commit constraints in CS, we include the complement of the commit constraint and only the assume constraints related to the generation of the particular output event of the commit constraint. These are determined based on the event block order [50]. This reduced CS is obtained by applying the four steps of producing TR in Section 7.3.2 with $k = Maxcycle$ as determined by the maximum time separation from *origin* to the output event, and in Step 4 only the related assume constraints are included.

In the example as shown in Figure 7.3, to verify the commit constraint to e_{Y1}, the assume constraint from *origin* to e_{X1} is included; to verify the commit constraint to e_{Y2}, the assume constraints to e_{X1} and e_{X2} are included.

Our verification procedure is thus as follows for each commit constraint in TD:

Step 1: Pre-calculate *Maxcycles*.

Step 2: Include the complement of the commit constraints, and the related assume constraints.

Step 3: Include the constraints characterizing TR for *Maxcycle* clock cycles.

Step 4: Check if the constraint system is consistent.

We conclude the above discussion to the following theorem.

Theorem 7.1 If the constraint system is inconsistent then the controller satisfies that commitment, else output a solution of the problem variables (assignments to event occurrence times and state values) from which a counter-example can be constructed.

If, for all the commit constraints, the above procedure yields an inconsistent system, then $TR \subseteq TR_{TD}$.

7. 4 Implementation in CLP(BNR) Prolog

We used CLP (BNR) Prolog which is a constraint logic programming environment based on relational interval arithmetic (RIA).

Constraints are expressed over interval-valued variables. Consistency checking is solved by domain narrowing and interval propagation as follows: one or more variables are originally constrained by finite domains (t_{origin} is set to [0, 0]). All other interval-valued variables are initialized with the value $[-\infty, \infty]$. An event-driven mechanism repeatedly selects primitive constraints and updates the value of the variables involved in these constraints until a fixed point is reached.

In a clock cycle Clk, if the occurrence time t_i of an input event e_i satisfies $t_i \in$ [(Clk-1) * P, Clk * P) then we have to enumerate all the possibilities for the event to occur in sub-regions of the interval t_i to achieve global consistency of the solution in CLP (BNR). The interested reader is referred to for details about partial and global consistency in CLP (BNR) Prolog. We use a Boolean variable $C_{i,Clk}$ in each clock cycle to model the possible occurrence of an input event $e_i \in E_{IN}$ as follows:

$$C_{i,Clk} = (t_i < (Clk * P)) \text{ and } (t_i >= (Clk - 1) * P) \tag{7.8}$$

The variable becomes true if and only if the event occurs within the Clk-th cycle.

A value assignment to the Boolean variables models a possible input sequence, their enumeration thus ranges over all such sequences. CLP (BNR) treats Boolean variables as intervals of integers ranging from 0 to 1. The Boolean variables remain unbounded until they are enumerated or reduced to punctual values by interval propagation and narrowing through constraints. In most cases, inconsistency is detected by the interval narrowing mechanism long before any enumeration takes place or an assignment of point values to only a subset of the variables may be sufficient to detect it, regardless the value of the remaining variables (that still carries the interval [0, 1]). Following is a small example showing how the interval narrowing mechanism works and how inconsistency is detected before a Boolean variable is enumerated.

Example: There are 3 events X, Y, Z in the system, between these events, there are following three constraints of between occurrence times (all in real) of these three events:

$Z = X + D_{xz}$, $Y = Z + D_{zy}$, and $Y = c*D_{xy1} + \sim c* D_{xy2}$, where $D_{xz} = [1, 6]$, $D_{zy} = [3, 6]$, $D_{xy1} = [0, 1]$, $D_{xy2} = [1, 2]$, and c is a boolean variable with its value to be either 0 or 1.

The CLP(BNR) Prolog code to check the consistency of the above constraint system would be:

```
check_consistency_example:-
[X, Y, Z]: real,
C: boolean,
D_xz: real [1 6], D_zy: real [3 6],
D_xy1: real [0 1], D_xy2: real [1 2],
{X ==0},
{Z == X + D_xz}, {Y == Z + D_zy},
{Y == C*D_xy1 + ~C*D_xy2},
```

The interval narrowing goes as follows: originally, $X = Y = Z = (-\infty, \infty)$. When the procedure arrives at line 5, X changes to 0. At line 6, X = 0, Z = [1 6] and Y = [1 6]

+ [3 6] = [4 12]. The procedure then arrives at line 7, this line says Y = ([0 1] * [0 1] + [0 1] * [1 2]) = [0 3] while the previous line asks Y = [4 12]. They have to be satisfied simultaneously, Therefore Y = [0 3] $\cap$ [4 12] = $\varnothing$. The procedure thus returns "no" indicating inconsistency of the system without enumerating the Boolean variable c.

Of course, the Boolean variable will be enumerated if no inconsistency is detected in the original constraint propagation where Boolean variables are treated as intervals [0 1].

By taking the advantage of the automatic backtracking mechanism and the built-in consistency checking techniques, the implementation of the verification procedure in CLP(BNR) is straightforward.

We first calculate *Maxcycles* using the algorithms in (in this example, *Maxcycle* = 28), then we create the following clause (*time_checker*) to check the consistency of the system consisting of the complement of the commit constraints and the constraints modeling TR. In the following, we show how the verification procedure is coded in CLP(BNR) Prolog for the example (T1, T2, T3, T4 are occurrence times of e_{Y1}, e_{Y2}, e_{X1}, e_{X2}, respectively). We discuss the general procedure afterwards.

```
time_checker:-
[T_origin,T1,T2,T3,T4]: real,
               /*declare timing variables as real*/
{T_origin==0},       /* set the reference point*/
assume_complement_commit_constraints(T_origin,T1,T2,T3,
T4,Maxcycle, XF,YF),
create_constraint_system(Maxcycle,0,0,0,0,T1,T2,T3,T4,
[]).
```

The clause *assume_ complement_commit_constraints* constructs the complement of the commit constraints and the related assume constraints. The assume constraints

$A_j = (e_i, e_j, [l_{ij}, u_{ij}])$, $j = 1, \ldots, p$, can be represented in CLP(BNR) Prolog as:

```
Dij: real [lij, uij],
{Tj == Ti + Dij},
```

where T_i and T_j are timing variables associated with e_i and e_j (In CLP(BNR), all constraints are enclosed in '{}', and an equality constraints are represented by '==').

The complement of a commit constraint $C_z = (e_v, e_z, [l_{vz}, u_{vz}])$, $z = 1, \ldots, q$, can be expressed as:

```
{Tz == (Tv + [uvz, _]; Tv + [_,lvz])},
```

where T_v and T_z are the timing variables associated with e_v and e_z, and ';' represents interval span (union) operation in which interval narrowing propagating from T_z to T_v may split into one of the two cases.

The construction of the constraints modeling the unfolding of the controller FSM and the traces of TDTA are implemented by two parallel clauses, one adds a copy of the FSM constraints one clock cycle after the other, while the other clause generates the constraints on the states of the TDTA and the enumeration list of the Boolean variables relating the input sequences.

```
create_constraint_system(Maxcycle,XF,YF,Clk,SP,YP,XP,
T1,T2,T3,T4,C):-
XN: integer(0,2),
[C3,C4, Input, Output]: boolean,
{Clk < Maxcycle},
generate_input_event(XP,Input),
next_state(SP,Input,SN),
output(SP,Input,Output),
next_y_state(YP,Output,YN),
time_assignment_T1T2(Clk,YP,YN,T1,T2),
input_boolean_variable(Clk,C3,C4,T3,T4),
next_x_state(XP,C3,C4,XN),
insert(C3,C,CC),
insert(C4,CC,New_C),
Clk1 is Clk+1,
create_constraint_system(Maxcycle,XF,YF,Clk1,SN,YN,XN,
T1,T2,T3,T4,New_C).

create_constraint_system(Maxcycle,Clk,SP,YP,XP,
T1,T2,T3,T4,C):-
{Clk==Maxcycle},
{XP==XF}, {YP==YF},
reverse(C,R_C), enumerate(R_C),
!.
```

The first clause is recursive. It represents Step 2.1 of producing TR. First, the current input value is determined as in Relation (7.5). Then, the next state and the output values of the FSM are calculated by adding the constraints as stated in Figure 7.4 and Equation (7.6). After that, constraints representing the next Y state are added as in Figure 7.5. The next clause includes constraints to detect the output events e_{Y1}, e_{Y2}, and to assign occurrence times to them in the form of Relation (7.7). Then, constraints modeling the possible input events in the form of Relation (7.8) are added. Following that, the next state of the WTA modeling the waveform X is computed as in Relation (7.3), where (t_i < Clk * P) and (t_i >= (Clk -1)* P) is substituted by $C_{i,Clk}$ from Equation (7.8). Next, the Boolean variables related to the input events in the clock cycle are inserted into a list that will be enumerated at the end of the construction of the constraint system. The clause ends with a recursive call to itself after incrementing Clk.

The process of constructing the constraint system corresponding to the recursive call of the first clause can be illustrated in Figure 7.6

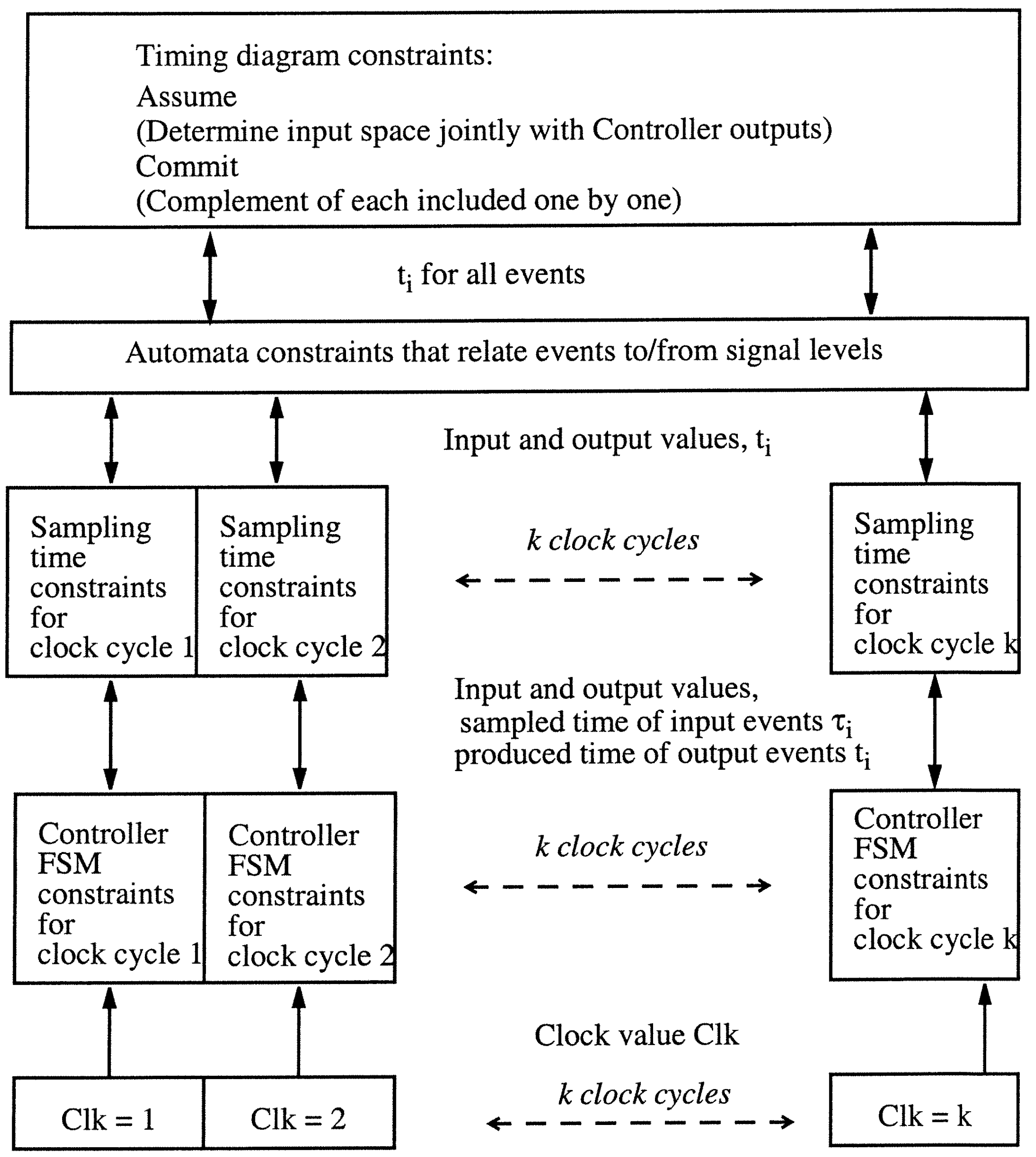

Timing diagram constraints:
Assume
(Determine input space jointly with Controller outputs)
Commit
(Complement of each included one by one)
t_i for all events
Automata constraints that relate events to/from signal levels
Input and output values, t_i
Sampling time constraints for clock cycle 1
Sampling time constraints for clock cycle 2
k clock cycles
Sampling time constraints for clock cycle k
Input and output values, sampled time of input events τ_i produced time of output events t_i
Controller FSM constraints for clock cycle 1
Controller FSM constraints for clock cycle 2
k clock cycles
Controller FSM constraints for clock cycle k
Clock value Clk
Clk = 1
Clk = 2
k clock cycles
Clk = k

Figure 7.6 Constructing the constraint system

The second clause defining *create_constraint_system* represents Step 2.2 of producing TR in Section 7.3.2. It checks the status of all the WTAs and enumerates the list of $C_{i,Clk}$ variables that determine all the valid input sequences. If the constraint system is consistent for an enumeration of the Boolean variables, the procedure returns a 'yes' to the main clause, representing a violation of the specification by the implementation. In that case, we can print the occurrence times of all the events in the system which make the constraint system be consistent. From there we can derive a counter example. Otherwise, the constraint system is inconsistent under all possible input sequences. In that case the main procedure returns a 'no' to the query *time_checker*, and we can conclude that the controller with the given clock period satisfies the TD specification.

7. 5 Efficiency Considerations and Experimental Results

7.5.1 Encoding state-transition relations

The efficiency of the verification procedure depends heavily on the number of constraints in the system. We can use some heuristics to reduce the number of constraints. For instance, the state transition of the FSM can be implemented as the following five constraints shown in Figure 7.7 instead of the original 22 shown in Figure 7.4

[((SP = 0) or (SP==6)) → (SN = input_value)] and

[(SP >= 1) and (SP =< 3) → (SN = SP + 1+ (~input_value) * 6)] and

[((SP = 4) or (SP= 10)) → (SN = 5 + (~input_value))] and

[((SP = 5) or (SP = 11)) → (SN = 7 + input_value * 4)] and

[((SP >= 7) and (SP =< 9)) → (SN = SP - 5 + 6 * (~input_value))]

Figure 7.7 Constraints modeling the next state of the FSM after optimization (~ is a negation).

A similar reduction can be achieved in the transition relations of WTAs modeling the X and Y waveforms.

A considerable improvement in the speed of determining the consistency of the constraint system was achieved by adding redundant constraints.

7.5.2 Redundant constraints

The commit constraints are verified in an order that follows increasing time (in a total order derived from the partial order of event "blocks" [50]), starting from *origin*. As soon as a commit constraint is verified, it is added to the constraint system. Since it holds, it is as such redundant. However, as it usually spans a number of clock cycles, it allows much faster propagation of interval narrowing through the constraint network than it would be the case through the FSM constraints, one clock cycle after another. Adding redundant constraints is a well known technique for speeding up the narrowing procedure inside a CLP system based on Relational Interval Arithmetic [66]. Here it pays off by a nearly order of magnitude speed up, especially for the commit constraints that are near the end of the timing diagram.

7.5.3 Comparison with rule-based Prolog clauses

The state transitions of TDTA and of the implementation FSM can also be implemented as Prolog rule-based clauses, which is similar to the sequential simulation by direct execution [26]. For instance, the state transition function of the FSM can be implemented in CLP(BNR) as follows:

```
next_state(0,0,0).
next_state(0,1,1).
     ...
next_state(5,0,6).
next_state(5,1,5).
```

Where next_state(present_state, input, next_state) is a prolog clause modeling the transition from the FSM state "present_state" to state "next_state" under input "input".

If during an unfolding of the controller FSM in a clock cycle the Boolean variable related to an input event is undetermined, then there may be more than one unit clauses that can be executed. For instance, if the current X state is 0 and the Boolean variable related to the input event e_{X1} is undetermined, then the next X state will be either 0 or 1. The correspondence can be described as follows: during the next clock cycle, if the current state of the FSM is 0 then both the clauses *next_state(0,0,0)* and *next_state(0,1,1)* are true. In an execution of the Prolog program, one of them will be selected nondeterministically, if it fails the next one is chosen and so on. In other words, we cannot include all timing information in the constraint system in one execution of the program as in the constraint-based method. The enumeration leads to many constraint systems being constructed during the execution of the procedure. When the length of the intervals in the assume constraints increases, the number of constraint systems so constructed increases exponentially.

The Prolog code of the simulation procedure is given in Appendix A. The use of rule based unit clauses is illustrated by the next state function of the FSM in Section 7. 3.

7. 6 Example Results

Given the TD in Figure 7.3, the FSM in Figure 7.2, and the clock period P = 10, we executed the prototype implementations in CLP(BNR) Prolog on a SPARC Station 5 under different values of the assume constraints. The Prolog code for this example is also shown in Appendix A (A1 is the implementation of our verification procedure and A2 is that of rule_based prolog clauses). Table 1 shows the CPU times and the memory used.

Table 7.2: Experimental Results

$[l_{o3}, u_{o3}]$	$[l_{14}, u_{14}]$	CLP verification CPU time (seconds)	Prolog simulation CPU time (seconds)	CLP verification memory (Kbytes)	Prolog simulation memory (Kbytes)
[1, 16]	[51, 78]	15	4	441	85
[1, 26]	[61, 88]	16	7	472	91
[1,36]	[61, 98]	17	13	504	98
[1,46]	[61, 108]	19	9	536	105
[1,56]	[61, 118]	20	17	567	115
[1,66]	[61, 128]	21	22	599	118
[1,76]	[61, 138]	22	27	631	125
[1,86]	[61, 148]	23	33	663	132
[1,96]	[61, 158]	24	40	694	139
[1,106]	[61, 168]	26	46	726	145
[1,116]	[61, 178]	27	53	758	152

The experiments show that the execution time of the constraint-based verification procedure increases linearly with the increase of the length of interval in the assume constrains, while that of the simulation increases exponentially as expected. The constraint-based form consumes more memory than the rule-based form, but in both cases, it increases linearly with the intervals in the constraints.

We can easily extend the verification procedure to deal with situation where clock frequency is changing with an range by representing P with an interval.

In the next Chapter, we apply the solution technique developed here to a practical example.

Chapter 8

Verifying Two Continuous Write Cycles of an Interface Controller

In this Chapter, we verify that the implementation of an interface controller will produce output on time (i.e., within the time specified in the TD specification) under the condition that all the inputs fed to the controller meet the assumptions as stated in the specification.

To apply our verification method, we need the specification in the form of a Timing Diagram (TD) and the implementation in the form of a finite state machine (FSM). In this experiment, the TD specification of the interface controller is derived from the general specification (GS) of the controller, while the FSM implementation is extracted from the RTL Verilog code of the design. The implementation of the controller is more complicated than as illustrated in due to the fact that some outputs are synchronous (outputs of flip-flops) and some are asynchronous (outputs of combinational gates).

To deal with the asynchronous outputs in the implementation, we modify the verification procedure in Chapter 7. In this experiment, we consider only the specification in two consecutive write cycles. We verified that all the commit constraints in the TD specification are satisfied. In other words, we verified that the implementation of the interface controller is correct with respect to its specification in two consecutive write cycles.

This chapter consists of five sections. In Section 8.1, we briefly introduce the interface controller and give the TD specification of the interface controller of two write cycles. In Section 8.2, we show the FSM implementation extracted from RTL Verilog code. In Section 8.3, we give the modified verification procedure to accommodate the asynchronous outputs. Finally in Section 8.5, we show the experimental results, and concluding that the implementation is correct as far as the write cycles are concerned.

8.1 Specification of the Interface Controller

The interface controller handles the handshaking between an external microprocessor and internal blocks of the ASIC, as well as the internal handshaking between the interface block and other internal blocks of the ASIC which have up accessible registors and/or memories. The simplified block diagram of the controller is shown in Figure 8.1.

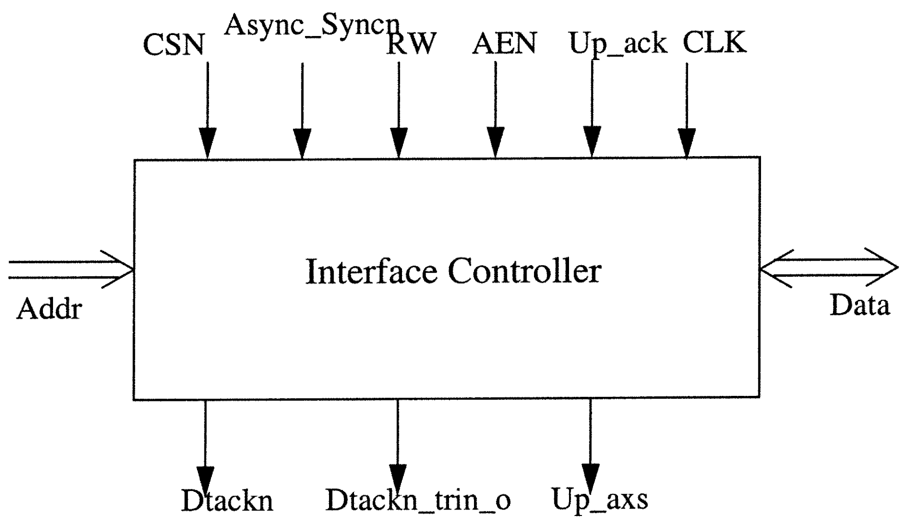


Figure 8.1 Interface controller block diagram

The controller is activated by setting the chip select pin (CSN) low. It can operate in asynchronous or synchronous mode by selecting the Async_Syncn pin to be high or low. In this experiment, the controller is verified in asynchronous operation mode. The basic operations of the controller in asynchronous mode are as follows.

In asynchronous mode, on the external microprocessor side, the controller initiates a data transfer by driving the read/write access enable pin (AEN) low, indicating that the address (Addr) on the bus is valid. A high read/write pin (RW) indicates that a read access is requested, while a low RW pin indicates that a write access is requested. When no data transfer is in progress, the device tri-states the data acknowledge pin (Dtackn). Immediately upon AEN being driven low, the device drives the Dtackn_trin_o pin high and latches the address that is on the bus. During a write access, the device pulls the Dtackn pin low to indicate to the controller that the requested data transfer can proceed. The controller then drives the AEN pin high during a write access to indicate that the data on the data[7:0] bus (Data) are valid. When the AEN is driven high, the device tri-states the Dtackn pin, indicating that no data transfer is in progress.

On the internal block side, read/write up data is carried out by an internal daisy-chained microprocessor bus. In Asynchronous mode, the controller sends out a one clock cycle pulse on Up_axs with every new read/write microprocessor access to the internal blocks and expects to receive an acknowledge signal (a high signal on Up_ack) upon completion of the access.

From the GS of the controller, we derived the timing diagram specification of two asynchronous write cycles of the controller shown in Figure 8.2.

There are 9 signals Addr, Data, CSN, AEN, RW, Up_axs, Upack, Dtackn and Dtackn_trin_o.

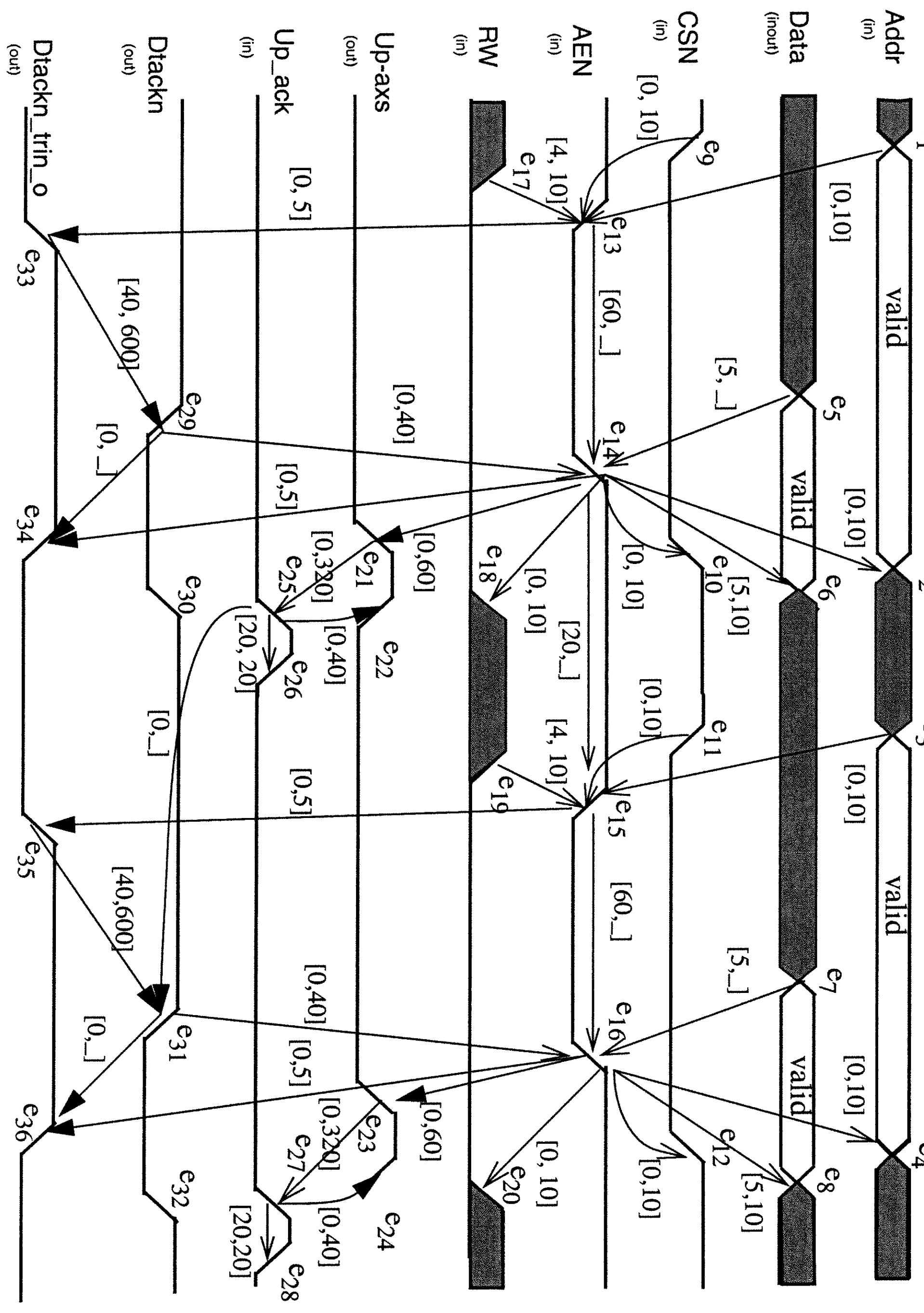


Figure 8.2 The timing diagram specification of two asynchronous write cycles

8.2 Implementation of the Controller

The controller is implemented as a pseudo-synchronous FSM as shown in Figure 8.3. The primary inputs (PI) of the controller, a combination of FSM outputs with PIs are sampled by synchronous registers and the outputs of the synchronous registers are fed to the FSM and the combinational logic producing the outputs of the controller. The primary outputs (PO) of the controller are generated from PI, the outputs of the FSM, and an output of a synchronous register through a combinational circuit. The FSM itself consists of a combinational logic block and registers holding the values of the state variables. The combinational logic block of the FSM is fed by the sampled inputs and by the outputs of the state registers.

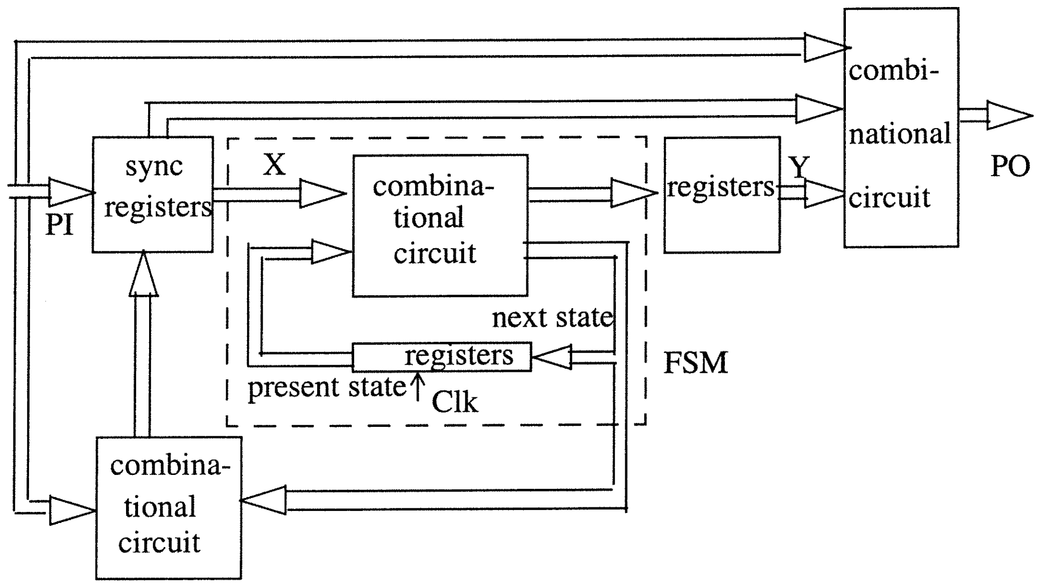


Figure 8.3 The implementation of the interface controller

In this section, we give functional information of each component in the controller implementation. In section 8.2.1, we describe the synchronous registers. In section 8.2.2, we describe the output functions of the controller. In section 8.2.3, we give the FSM diagram, the input function, the transition function and the output function of the FSM.

8.2.1 Synchronous Registers

The controller inputs, CSN and AEN, are sampled by a cascade of two registers (the registers sampling CSN is shown in Figure 8.4, the ones sampling AEN is similar) before entering the FSM. The inputs of these registers are CSN, CSN_M1, AEN, AEN_M1, respectively, and the outputs of these registers are CSN_M1, CSN_M2, AEN_M1, AEN_M2, respectively.

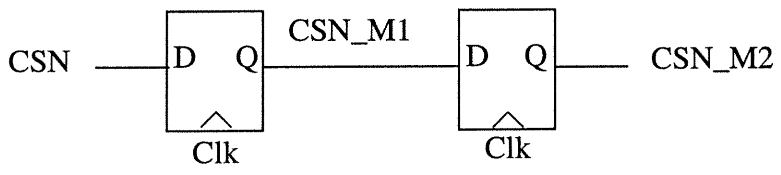


Figure 8.4 Cascade registers sampling CSN

There are three other synchronous registers sampling the signals Up_ack, Up_wrnlocal, and Up_axspending, respectively, where Up_ack is a PI,

$$\text{Up\_wrnlocal} = \sim\text{Up\_wrnlocal\_rst and (Up\_arwlatch and } \sim\text{RWN or } \sim\text{Up\_arwlatch and Up\_wrnlocal\_R)}, \tag{8.1}$$

and

$$\text{Up\_axspending} = (\text{Up\_naxspls or Up\_axspending\_R}) \text{ and } \sim\text{Up\_ack}. \tag{8.2}$$

Here, Up_wrnlocal_rst, Up_arwlatch, Up_naxspls are outputs of the FSM, and RW, Up_ack are PIs.

The outputs of these three synchronous registers are Up_ack_R, Up_wrnlocal_r and

Up_axspending_R, respectively.

8.2.2 Outputs of the Controller

There are three outputs of the controller as shown in the controller block diagram (Figure 8.1). They are Dtackn, Dtackn_trin_o and Up_axs. The outputs are determined by primary inputs, the outputs of the FSM and an output of a synchronous register. The output functions are:

$$\text{Dtackn} = \text{CSN or Dtackn\_R}, \tag{8.3}$$

where Dtackn_R = AEN or ~Dtackn_rst.

$$\text{Dtackn\_trin\_o} = \sim(\text{CSN or AEN}) \tag{8.4}$$

$$\text{Up\_axs} = \text{Up\_naxs or Up\_axspending\_R} \tag{8.5}$$

where CSN, AEN are PIs, Dtackn_rst, Up_naxs are FSM outputs, and Up_axspending_R is an output of a synchronous register.

8.2.3 FSM Diagram

We derived the FSM inside the interface controller from the RTL Verilog code. The simplified transition diagram of the FSM relating to asynchronous WRITE mode is shown in Figure 8.3. It is a Mealy finite state machine. There are 4 states, 5 inputs and 4 outputs of the FSM. The 4 states are: idle (IDLE), write wait (WWAIT), write acknowledge (ACKW) and read end (REND). The 5 inputs of the FSM are Up_wrnlocal_R, Up_ack_R, RW, CSN_M2, AEN_M2, where RW is a primary input of the controller and the other 4 are outputs of synchronous registers. the 4 outputs of the FSM are: Up_wrnlocalrst, Up_arwlatch, Dtackn_rst, Up_naxspls.

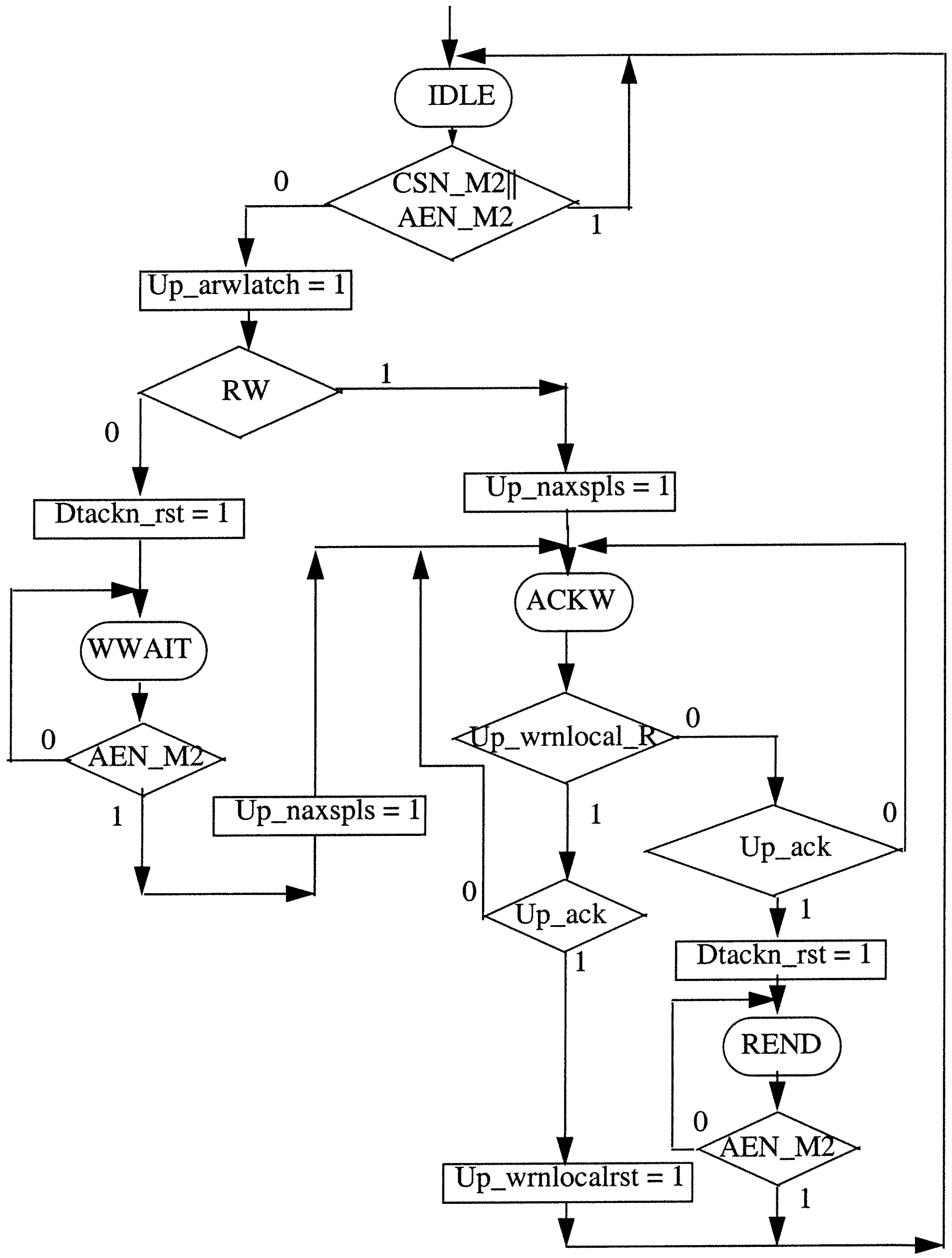


Figure 8.5 Asynchronous interface control FSM

The state transition function is also derived from the Verilog code, which is

((SP = IDLE) and (CSN_M2 or AEN_M2) or (SP = AREND) and
AEN_M2 or (SP = AACKW) and up_wrnlocal_R and up_ack)
$\Rightarrow$ (SN = IDLE)
and
((SP = AWWAIT and ~AEN_M2) or
(SP = IDLE) and ~(CSN_M2 or AEN_M2) and ~RW)
$\Rightarrow$ (SN = AWWAIT)
and
((SP = IDLE) and ~(CSN_M2 or AEN_M2) and RW or
(SP = AACKW) and ~up_ack or (SP = AWWAIT) and AEN_M2)
$\Rightarrow$ (SN = AACKW)
and
((SP = AACKW) and ~up_wrnlocal_R and
up_ack or (SP = AREND) and (~AEN_M2))
$\Rightarrow$ (SN = AREND).

Where SP and SN represents present and next state of the FSM, respectively.

The output functions of the FSM are:

Up_wrnlocal_rst = (SP = AACKW) and Up_wrnlocal_R and Up_ack), (8.6)

Up_arwlatch = ((SP = IDLE) and ~(CSN_M2 or AEN_M2)), (8.7)

Dtackn_rst = (((SP = IDLE) and ~(CSN_M2 or AEN_M2) and ~RWN) or ((SP = AACKW) and ~up_wrnlocal_R and up_ack)), (8.8)

Up_naxspls = (SP = IDLE) and ~(CSN_M2 or AEN_M2) and RWN or (SP = MW-WAIT) and AEN_M2 (8.9)

8.3 Verification Procedure

As explained in Chapter 7, if an output is synchronous, then we can detect the occurrence of the output based on the present and next state of the waveform automaton of the output. The occurrence time of the output event can be assigned as (Clk + 1) * P, where Clk is the clock cycle in which the automaton change from the present state to the next state, P is clock period.

If an output is asynchronous, it is more difficult to determine the occurrence time of an output events. For instance, from Equation (8.4), we know that the rising edges on Dtackn_trin_o (e_{33} and e_{35}) are caused by the falling edges on CSN and AEN (e_9 and e_{13}, e_{11} and e_{15})]. The occurrence time of the events corresponding to a rising edge on Dtackn_trin_o is the latest of the occurrence times of the events corresponding to the falling edges on CSN and AEN (because Dtackn_trin_o is ~CSN and ~AEN). In other words, we can compute the occurrence times of e_{33} and e_{35} as follows:

$$t_{33} = \max(t_9, t_{13}) \tag{8.10}$$

$$t_{35} = \max(t_{11}, t_{15}) \tag{8.11}$$

The occurrence times of the events corresponding to the falling edges on Dtackn_trin_o are the earliest of the occurrence times of the events corresponding to the rising edges on CSN and AEN. We thus have

$$t_{34} = \min(t_{10}, t_{14}) \tag{8.12}$$

$$t_{36} = \min(t_{12}, t_{16}) \tag{8.13}$$

The occurrence times of events on Dtackn are more complicated. By rewrite Equation (8.3), we have

$$\text{Dtackn} = \text{CSN or AEN or} \sim\text{Dtackn\_rst}, \tag{8.14}$$

where Dtackn_rst is an output of the FSM. The output of the FSM is generated synchronously with clocks. Basically, we do not know who causes the signal changes on the output Dtackn. What we did is that we examined the simulation results of the Dtackn_rst and determined that the rising edges on Dtackn are always generated by the rising edges on CSN and AEN, and the falling edges on Dtackn are always generated by the rising edges on Dtackn_rst. We thus have

$$t_{29} = (\text{Clk} + 1) * P \tag{8.15}$$

$$t_{31} = (\text{Clk} + 1) * P \tag{8.16}$$

$$t_{30} = \min(t_{10}, t_{14}) \tag{8.17}$$

$$t_{30} = \min(t_{12}, t_{16}) \tag{8.18}$$

For the output Up_axs, from Equation (8.5), it is a combination of Up_nax and Up_axspending_r, both of them are synchronous signals. Therefore, the occurrence times of all the events on Up_axs (t_{21}, t_{22}, t_{23}, t_{24}) can be assigned to (Clk + 1) * P (we do not consider the delay on the OR gate), where Clk is the clock cycle in which the output automaton changes its related states.

We modify the verification procedure in Chapter 7 using the above occurrence time assignments to output events.

8.4 Redundant constraints to improve efficiency

The commit constraints are verified in an order that follows increasing time (in a total order derived from the partial order of event "blocks" [6]), starting from origin. As soon as a commit constraint is verified, it is added to the constraint system. Since it holds, it is as such redundant.

8.5 Experimental Results and Conclusions

We verified all the thirteen commit constraints in the specification in Figure 8.2 on a Sparc 10 workstation and no violation was found. The Prolog code for this experiment can be found in the Appendix A (A3). The following table shows the CPU time, the memory used, and the redundant commit constraints used when verifying a commit constraint.

Table 8.1: Experimental Results of the Bus Controller

index	verified constraint	# of unfold-ings	index of redundant commit constraints	CPU time (seconds)	memor y used (Kbytes)
1	T33 - T13 ∈ [0, 5]	2		5	191
2	T29 - T33 ∈ [40, 600]	32	1	6	256
3	T34 - T29 ∈ [0, _]	35	1, 2	20	3,049
4	T34 - T14 ∈ [0, 5]	35	1, 2	36	3,051
5	T21 - T14 ∈ [0, 60]	38	1, 2	10	656
6	T22 - T25 ∈ [0, 40]	70	1, 2, 5	1,350	6,243
7	T35 - T15 ∈ [0, 5]	68	1 - 6	19,687	6,094
8	T31 - T25 ∈ [0, _]	88	1 - 7	186	7,875
9	T31 - T35 ∈ [40, 600]	88	1 - 7	12,055	7,988
10	T36 - T31 ∈ [0, _]	90	1 - 9	1408	8,007
11	T36 - T16 ∈ [0, 5]	90	1 - 10	11,044	8,198
12	T23 - T16 ∈ [0, 60]	90	1 - 10	6,497	8,084
13	T24 - T27 ∈ [0, 40]	90	1 - 12	107,699	8,210

From the results in Table 8.1, we can see that the CPU time is not always increasing to verify constraints towards the end of the TD specification, which is the case in the previous example in Chapter 7 as illustrated in Table 7.2. This is the effect of including redundant constraints. We tried to verify the last commit constraints (T24 - T27 $\in$ [0, 40]) without the redundant constraints, the run did not finish after 72 hours on the same machine.

We developed a method using constraint logic programming for verifying whether a pseudo-synchronous (sampled input) FSM implementation of a real-time controller satisfies its TD specification scenario. Verification by simulation of such real-time designs is very difficult, since many possible timing situations may exist and the designer must foresee them all in the testbench or be satisfied with incomplete verification. Our method guarantees to be exhaustive on the given finite TD specification.

Our CLP-based method can be easily extended to systems containing earliest (min) and latest (max) constraints, and delay correlation as in [36]. An automated procedure for translating Verilog or VHDL RTL models to a system of constraints is needed.

Chapter 9

Conclusions and Future Work

9.1 Contributions

Interface verification is very important in system design. In this thesis, we have presented a set of algorithms for computing the maximum time separations in constraint graphs transformed from a type of HAAD specification. The algorithms are directly applicable to compatibility verification of interface specifications. They can also be applied to check other safety timing properties on interface specifications. In the second part of the work, we have developed a solution technique to verify the correctness of real time controllers against their leaf HAAD specifications. The main contributions of this dissertation are as follows.

In the area of timing verification of interface specifications, we extended the solution techniques in [17][36] to interfaces specified in Loop over a leaf HAAD language. We achieved this by

1. Giving a way to transform the HAAD specification to a constraint graph;

2. Providing algorithms for computing maximum time separations in the transformed constraint graphs containing linear-only and linear-plus-latest constraints;

3. Formulating timing verification problems in the form of maximum time separation problems.

In the area of verification real time controllers against their specification, we give a complete solution of the problem. The contributions are:

1. Modeling a timing diagram specification as communicating "TD" automata that

accept event traces respecting the timing constraints;

2. The representation of the finite unfolding of the implementation FSM, the finite execution of the TD automata and timing requirements in the form of constraints;

3. Formulating the FSM versus TD timing verification problem as a consistency check of a series of constraint systems;

4. Implementation of the method using Constraint Logic Programming based on Relational Interval Arithmetic environment;

5. Acceleration of the convergence of the implemented algorithm by adding redundant constraints.

In both areas, we applied the solution techniques to real designs. We verified the causality and two safety timing properties of an interface specification modeling repeated microprocessor READ operations (inspired by MC68360) from a memory. It is based on computing the maximum time separation of events in restricted constraint graphs containing linear-plus-latest constraints. We also verified that the FSM model of a real-time controller extracted from its RTL Verilog description satisfies its timing diagram specification of two consecutive write cycles.

9.2 Future Work

In the direction of computing the maximum time separation in constraint graphs, the earliest (min) constraints should also be considered. The general maximum time separation problem in cyclic system containing linear-plus-min-plus-max constraints is complicated since as pointed out in [62], it is an NP-complete problem even in the acyclic systems. But it is possible to find an efficient algorithm if the constraint graph is restricted properly.

In the verification of interface controllers with respect to its specifications, we are developing algorithms for verifying cyclic scenarios with choices, e.g., read or write cycles, and bounding the number of unfoldings needed to cover all possible situations. In general, we should estimate an upper bound on the number of un-foldings of the combined TD plus FSM system required before all the maximum event separations start to repeat.

We observed that the efficiency of our verification procedure depends heavily on the number of constraints in the system. It is interesting to look into the algorithms and their complexity in the area of reducing the number of constraints in encoding state transition relations of FSM and WTAs.

Bibliography

[1] R. Alur, D. Dill. A theory of timed automata. Theoretical Computer Science, p183-235, 1994.

[2] R. Alur, C. Courcoubetis, and D. Dill. Model Checking in Dense Real Time. Information and Computation, p2-34, 1993.

[3] R. Alur and T.A. Henzinger. A Really Temporal Logic. In Proceedings of the 30th Annual Symposium on Foundations of Computer Science, 1989.

[4] R. Alur and T.A. Henzinger. Logics and Models of Real-Time: A Survey. In Proceedings of REX Workshop "Real-Time: Theory in Practice", p74-106, June 1991.

[5] T. Aom. Specification, Simulation, and Verification of Timing Behaviour. PhD Dissertation, University of Washington, 1993

[6] J. Baeten and J. Bergstra. Real-Time Process Algebra. Formal Aspects of Computing, p142-188, 1991.

[7] S. Berezin et al. Model Checking Using Stalmarck's Method. In Proceedings of the Formal Methods in Computer-Aided Design, FMCAD'98, Palo Alto, CA, p369-386, 1998

[8] B. Berkane, S. Gandrabur, E. Cerny. Algebra of Communicating

Timing Charts for Describing and Verifying Hardware Interfaces. In the Proceedings of CHDL'97, Toledo, Spain, April 1997.

[9] B. Berkane, S. Gandrabur, E. Cerny. Timing Diagrams: Semantics and Timing Analysis, in Proceedings of the 3rd Asian-Pacific Conference on Hardware Description Languages, Bangalore, India, 1996.

[10] T. Bolognesi and F. Lucidi. Lotos-Like Process Algebra with Urgent or Timed Interactions. In Proceedings of REX Workshop "Real-Time: Theory in Practice", June 1991.

[11] G. Borriello. A New Interface Specification Methodology and its Application to Transducer Synthesis. Ph.D. Dissertation, University of California at Berkeley, 1991

[12] P. Bremond-Gregoire, J.Y. Choi, and I. Lee. The Soundness and Completeness of ACSR. Technical Report MS-CIS-93-59, University of Pennsylvania, June 1993.

[13] J.A. Brzozowski, T. Gahlinger and F. Mavaddat. Coherence and Satisfiability of Waveforms Timing Specifications, Networks, Vol.21, No. 1, p 97-107, 1991.

[14] J.R. Burch. Trace Algebra for Automatic Verification of Real-Time Concurrent Systems. PhD Dissertation, Carnegie Mellon University, August 1992.

[15] T. M. Burks, K. A. Sakallah. Min-Max Linear Programming and the

Timing Analysis of Digital Circuits. In Proceedings of International Conference on Computer Design ICCD'93, Oct. 1993, U.S.A.

[16] S. M. Burns. Performance Analysis and Optimization of Asynchronous Circuits. PhD dissertation, California Institute of Technology, 1991.

[17] E. Cerny, B. Berkane, P. Girodias, K. Khordoc. Hierarchical Annotated Action Diagrams: An Interface-Oriented Specification and verification Method, Kluwer Academic Publishers, 1998.

[18] E. Cerny, F. Jin. Verification of Interface Controllers Against Timing Diagram Specification Using Constraint Logic Programming. 1999 International Conference on Computer Design ICCD'99, Oct., 1999, U.S.A.

[19] E. Cerny, F. Jin. Verification of Real Time Controllers Against Timing Diagram Specification Using Constraint Logic Programming. European Micro'98, Sept. 1998, Sweden.

[20] E. Cerny, K. Khordoc. Interface Specifications with Conjunctive Timing Constraints: Realisability and Compatibility. 2nd AMAST Workshop on Real-Time Systems, Bordeaux, June 1995.

[21] S. Chakraborty and D. L. Dill. Approximate Algorithms for Time Separation of Events, Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD), p190-194, November 1997.

[22] S. Chakraborty, K. Yun and D.L. Dill. Timing Analysis of Asynchronous Systems Using Time Separation of Events. 1998 IEEE Custom Integrated Circuits Conference, p455-458, May 1998.

[23] T.A. Chu. Synthesis of Self-Timed VLSI Circuits Fro Graph-theoretic Specification. PhD Dissertation, Massachusetts Institute of Technology, 1987.

[24] E. M. Clarke, E.A. Emerson, and A.P. Sosta. Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specification. ACM Transaction on Programming Language and Systems, Vol. 8, no.2, p244-263, 1986.

[25] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurrent Systems. ACM TOPLAS 15, p36-72, 1993.

[26] W.F. Clocksin. Logic Programming and Digital Circuit Analysis, J. Logic Programming, Nov.4, p59-82, 1987.

[27] D. Dill. Trace Theory for Automatic Hierarchical Verification of Speed Independent Circuit. Ph. D Dissertation. Carnegie Mellon University, 1989.

[28] E.A. Emerson, A.K. Mok, A.P. Sistla, and J. Srinivasan. Quantitative Temporal Reasoning. In Proceedings of International Workshop on Automatic Verification Methods for Finite State Systems. LNCS 407, 1989.

[29] M. A. Escalante and N. J. Dimo poulos. A Probabilistic Timing Analysis for Synthesis inMicroprocessor Interface Design. In Proceedings of the IEEE Pacific Rim Conference on Communications, Victoria, B.C., p277-280, 1995.

[30] M. A. Escalante and N. J. Dimo poulos. Modeling Timing Correlation and the Accurate Timing Verification of Digital Interface Circuits. In Proceedings of the IEEE 39th Midwest Symposium on Circuits and Systems. Des Moines, Iowa, 1996.

[31] M. A. Escalante and N. J. Dimo poulos. Timing Analysis for Synthesis of Hardware Interface Designs Using Timing Signal Transition Graphs. In Proceedings of the Sixth International Workshop on Petri Nets and Performance Models. Duke University, Durham, North Carolina, 1995.

[32] M. Felder, D. Mandrioli and A. Morzenti. Proving Properties of Real-Time Systems Through Specifications and Petri Nets Models. IEEE Transactions on Software Engineering, Vol SE-20, p127-141, 1994.

[33] R. W. Floyd. Algorithm 97: Shortest Path, CACM p345, June 1962.

[34] W. Fokkink and S. Klusener. An Axiomation for Real Time ACP, 1995.

[35] T. Gahlinger. Coherence and Satisfiability of Waveform Timing Specifications. PhD Dissertation, University of Waterloo, 1990.

[36] P. Girodias and E. Cerny. Interface Timing Verification with Delay Correlation Using Constraint Logic Programming. In Proceedings of the European Design & Test Conference, ED&TC' 97, Paris, March 1997.

[37] P. Girodias, E. Cerny and W.J.Older. Solving Linear, Min and Max Constraint Systems Using CLP Based on Relational Interval Arithmetic, in Constraint Programming Symp., CP95, 1995 (longer version in Journal of Theoretical Computer Science in 1997).

[38] M. Gondran and M. Minoux. Graphs and Algorithms. A Wiley-Interscience Publication 1984.

[39] A. Gupta. Formal Hardware Verification Methods: a Survey. Formal Methods in System Design, p151-238, 1992.

[40] M.A. Holliday, M.K. Vernon. A Generalized Timed Petri Net Model for Performance Analysis. IEEE Trans. Software Eng, Vol. 13, p1297-1310, 1987.

[41] H. Hulgaard. Timing Analysis and Verification of Timed Asynchronous Circuits. PhD Dissertation, University of Washington, 1995.

[42] H. Hulgaard, S. Burns, T. Amon and G. Borriello. An Algorithm for Exact Bounds on the Time Separation of Events in Concurrent Systems. IEEE Transactions on computers, Vol. 44, p1306-1317, 1995.

[43] H. Hulgaard, S. Burns, T. Amon and G. Borriello. An Algorithm for Exact Bounds on the Time Separation of Events in Concurrent Sys-

tems. Proceedings of International Conference on Computer Design, p166-173, Oct. 1993.

[44] J. Jaffar, M. Maher. Cnstraint Logic Programming: A Survey. Journal of Logic Programming, p503-581, 1994.

[45] F. Jahanian, A.K.L. Mok. A Graph-Theoretic Approach for Timing Analysis and its Implementation. IEEE Transaction on Computers c-36, p961-975, 1987.

[46] F. Jahanian, A.K.L. Mok. Safety Analysis of Timing Properties in Real-Time Systems. IEEE Transactions on Software Engineering, p890-904, 1986.

[47] F. Jahanian, A.K.L. Mok. Modechart: A Specification Language for Real-Time Systems. IEEE Transactions on Software Engineering, Vol 20, No. 12, p933-947, December 1994.

[48] F. Jin, P. Girodias, E. Cerny. Safety Analysis in Real Time System Using Constraint Logic Programming. IEEE 1997 Canadian Conference on Electrical and Computer Engineering, May 1997.

[49] F. Jin, H. Hulgaard, E. Cerny. Maximum Time separation of Events in Cyclic Systems with Linear and Latest Timing Constraints. International Conference on Formal Methods in Computer-aided Design, Nov. 1998, U.S.A.

[50] K. Khordoc, E. Cerny. Semantics and Verification of Timing Diagrams with Linear Timing Constraints, ACM Trans. Design Aut. of

Electronic Systems (TODAES), p21-50, 1998

[51] K. Khordoc, and E. Cerny, M. Dufresne. Modelling and Execution of Timing Diagrams with Optional and Multimatch Events. Proc. 2nd ACM Workshop on Timing Issues in the Specification and Synthesis of Digital System.

[52] K. Khordoc, M. Dufresne, and E. Cerny. Integrating Behaviour and Timing in Executable Specifications. In Proc. of the Int'l Conference on Hardware Description Languages, Ottawa, Canada, 1993.

[53] K. Khordoc, M. Dufresne, and E. Cerny. A Stimulus/Response System Based on Hierarchical Timing Diagrams. IEEE Proc. ICCAD-91, p358-361,1991.

[54] R. Koymans. Specifying Message-Passing and Time-Critical System with Temporal Loigc. Ph.D Dissertation, Eindhoven University of Technology, The Netherlands, 1989.

[55] R. Kurshan. Computer-Aided Verification of Coordinating Processes. Princeton University Press, 1994.

[56] Lawler and L. Eugene. Combinatorial Optimization: Networks and Matroids. Holt, Kinehart and Winston, 1976.

[57] I. Lee, P. Bremond-Gregoire, and R. Gerber. A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems. Proceedings of the IEEE, p158-171, Jan., 1994.

[58] N. Lopez-Benitez. Dependability Modelling and Analysis of Distributed Programs. IEEE Trans. Software Eng, Vol SE-20, p345-352, 1994.

[59] N.G. Leveson, J.L. Stolzy. Safety Analysis Using Petri Nets. IEEE Transaction on Software Engineering, Vol.13 p386-397, 1987.

[60] O. Maler, S. Yovine. Hardware Timing Verification Using KRONOS. In Proceedings 7th Israeli Conference on Computer Systems and Software Engineering, 1996.

[61] P.M. Merlin, D. J. Farber. Recoverability of Communication Protocols Implications of a Theoretical Study. IEEE Trans. Communication, Vol.24, p1036-1043, 1976.

[62] K. McMillan, D.L.Dill. Algorithms for Interface Timing Verification. In IEEE International Conference on Computer Design, 1992.

[63] O. Millet. Multicycles and RTL Logic Satisfiability. In Proceedings of Formal Techniques in Real-Time and Fault Tolerant Systems, p73-86, 1992.

[64] F. Moller, C. Tofts. A Temporal Calculus of Communicating Processes. In Proceedings of Theories of Concurrency: Unification and Extension), LNCS 458, p401-415, 1990.

[65] X. Nicollin, J. Sifakis. The Algebra of Timed Processes, ATP: Theory and Application. Information and Computation, Vol. 114, p131-178, 1994.

[66] W. Older and F. Benhamou, programming in CLP(BNR), BNR Research Report, 1993. The CLP (BNR) Prolog will be distributed soon by Advanced Logic Systems, Inc., Cambridge, MA. See http://www.als.com/als/clpbnr/clp_info.html for information and user manuals.

[67] J. Ostroff. Automated Verification of Timed Transaction Model. In Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems. LNCS 407, p247-256, 1989.

[68] S. Palnitka. Verilog HDL: A Guide to Digital Design and Synthesis. Prentice HAll, Mar., 1996.

[69] D. Perry. VHDL. McGraw-Hill, Inc. 1991.

[70] C. V. Ramamoorthy. Analysis of Asynchronous Concurrent Systems by Petri Nets. Cambridge, MA: MIT Project MAC, TR-120, 1974.

[71] C. V. Ramamoorthy and G. S. Ho. Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets. IEEE Transactions on Software Engineering Vol. 6, p440-449, 1980.

[72] R.R. Razouk, C.V. Phelps. Performance Analysis Using Time Petri Nets. In Proceedings 4th IFIP Protocol Specification, Testing and Verification, 1985.

[73] G. Reed and A. Roscoe. Metric Spaces as Models for Real-Time Concurrency. In Proceedings of Mathematical Foundations of Com-

puter Science, LNCS, Vol. 298, New York, 1987.

[74] P. Rony. Interface Fundamentals: Timing Diagram Convention. Computer Design, p152-153, 1980.

[75] S. Sjonolm, L. Lindh. VHDL for Designers. Prentice Hall, Dec. 1996.

[76] D. Stuart. Formal Methods for Real-time System. Ph.D Dissertation, University of Texas at Austin, 1996.

[77] J.J.P. Tsai, S. J. Yang and Y. H. Chang. Timing Constraint Petri Nets and Their Application to Scheduability Analysis of Real-Time System Specifications. IEEE Transaction on Software Engineering, Vol. 21, p32-49, 1995.

[78] P. Vanbekbergen, G. Goossens & H.D. Man. Specification and Analysis of Timing Constraints in Signal Transition Graphs. In European Design Automation Conference, March 1992.

[79] P. Vanbekbergen. Synthesis of Timed Asynchronous Circuits. In Proc. of European DAC, 1994.

[80] E. A. Walkup and G. Borriello. Interface Timing Verification with Applications to Synthesis. Proceedings of the Design Automation Conference, 1994.

[81] E. A. Walkup. Optimization of Linear Max-Plus Systems with Application to Timing Analysis, Ph.D Dissertation, University of

Washington, 1995.

[82] Y. Wang. CCS + Time = an Interleaving Model for Real Time Systems. In Proceedings of ICALP'91, Spain, 1991.

[83] T. Y. Yen, A. Ishiii, A. Casavant and W. Wolf. Efficient Algorithms for Interface Timing Verification. In Proceedings of the 1994 European DAC, p34-39, September 1994.

Appendix A

The appendix contains code in BNR Prolog of the three examples in Chapters 7 and 8. The first one (A1) is the procedure using rule-based prolog clauses to simulate the execution of the FSM on the example in Chapter 7. The second one (A2) is constraint-based approach for the same example in Chapter 7. A3 contains the code to verify the interface controller in Chapter 8 using the constraint-based approach. The execution results of the procedures in A1 and A2 are shown in Table 7.2, and the results of procedure in A3 are illustrated in Table 8.1.

A1. Prolog code of rule-based prolog clause approach on the example in Chapter 7

```
% assume and complement constraints and the final states of the automata modeling waveform
%X and Y added to verify the commit constraint from T3 to T1
assume_complement_commit_constraints(T1,T2,T3,T4,XF,YF):-
Assume_root_T3: real(1,46),
{T3==Assume_root_T3},
{T3>=0},
{XF==1},
{YF==1},
{T1<T3+70};
{T1>T3+80}.

% assume and complement constraints and the final states of the automata modeling waveform
%X and Y added to verify the commit constraint from T4 to T2
assume_complement_commit_constraints(T1,T2,T3,T4,XF,YF):-
Assume_root_T3: real(1,46),
Assume_T1_T4: real(61,108),
{T3==Assume_root_T3},
{T4==T1+Assume_T1_T4},
{T4>T3},{T3>=0},
{XF==2},{YF==2},
{T2<T4+70};
```

```
{T2>T4+80}.

%create the input value according to the current state of the X automaton
generate_input_event(0,0).
generate_input_event(1,1).
generate_input_event(2,0).

% the next state of the FSM according to the current state and the input value
next_state(0,0,0). next_state(0,1,1).
next_state(1,0,8). next_state(1,1,2).
next_state(2,0,9). next_state(2,1,3).
next_state(3,0,10). next_state(3,1,4).
next_state(4,0,6). next_state(4,1,5).
next_state(5,0,7). next_state(5,1,11).
next_state(6,0,0). next_state(6,1,1).
next_state(7,0,8). next_state(7,1,2).
next_state(8,0,9). next_state(8,1,3).
next_state(9,0,10). next_state(9,1,4).
next_state(10,0,6). next_state(10,1,5).
next_state(11,0,7). next_state(11,1,11).

% The Output tof the FSM according to the current state
output(0,0). output(1,0).
output(2,0). output(3,0).
output(4,0). output(5,0).
output(6,1). output(7,1).
output(8,1). output(9,1).
output(10,1). output(11,1).

% next state of the X automaton based on the current state and the waveform value on X
next_x_state(0,0,_,0).
next_x_state(0,1,_,1).
next_x_state(1,_,0,1).
next_x_state(1,_,1,2).
next_x_state(2,_,0,2).

% next state of the X automaton based on the current state and the waveform value on X
next_y_state(0,0,0).
next_y_state(0,1,1).
next_y_state(1,0,2).
next_y_state(1,1,1).
next_y_state(2,0,2).

% assign occurrence time on output events according to the current and the next state of
% Y automaton
assign_T1T2(I,0,1,T1,T2):- {T1==(I+1)*10},
assignt_T1T2(I,1,2,T1,T2):- {T2==(I+1)*10},
assign_T1T2(I,0,0,T1,T2).
assign_T1T2(I,1,1,T1,T2).
assign_T1T2(I,2,2,T1,T2).
```

The clause building the constraint system

```
create_constraint_system(XF,YF,I,Current_state,Current_y_state,XP,T1,T2,T3,T4):-
[XP,XN, Current_y_state]: integer(0,2),
[Current_state, Next_state]: integer(0,11),
[C1,C2, Input, Output]: boolean,
{Current_y_state<>YF},
generate_input_event(XP,Input),
next_state(Current_state,Input,Next_state),
output(Current_state,Output),
next_y_state(Current_y_state,Output,Next_y_state),
constraint_T1T2(I,Current_y_state,Next_y_state,T1,T2),
{C1==(T3<(I+1)*10) and (T3>=I*10)},
{C2==(T4<(I+1)*10) and (T4>=I*10)},
next_x_state(XP,C1,C2,XN),
enumerate([C1,C2]),
I1 is I+1,
create_constraint_system(XF,YF,I1,Next_state,Next_y_state,XN,T1,T2,T3,T4).

create_constraint_system(XF,YF,I,Current_state,Current_y_state,XP,T1,T2,T3,T4):-
{Current_y_state==YF},
{XP==XF}.

% the main clause
ex:-
[T1,T2,T3,T4]: real,
assume_complement_commit_constraints(T1,T2,T3,T4,XF,YF),
Initial_state is 0,
create_constraint_system(XF,YF,0,0,0,Initial_state,T1,T2,T3,T4),
write('T1='),print(T1),nl,
write('T2='),print(T2),nl,
write('T3='),print(T3),nl,
write('T4='),print(T4),nl.
```

A2. Prolog code of constraint-based approach on the example in Chapter 7

```
% append a list to another list
append([],L,[L]).
append([X|Xs],L,[X|Zs]):- append(Xs,L,Zs).

% reverse a list
reverse([],[]).
reverse([X,Xs..],Zs):-
reverse([Xs..],[Ys..]), append([Ys..],X,Zs).
```

```
% insert an element to a list
insert(X,[Xs..],[X,Xs..]).
insert(X,[],[X]).

% assume constraints in the specification
assume_constraints(T1,T2,T3,T4):-
Assume_root_T3: integer(1,126),
Assume_T1_T4: integer(51,178),
{T3==Assume_root_T3},
{T4==T1+Assume_T1_T4}.

% complement of the commit constraints, the final states of the automata, and the maxcycle to
% verify the constraint from T3 to T1.
complement_commit_constraints(T1,T2,T3,T4,Maxcycle):-
{T1<T3+39};
{T1>T3+50},
{Maxcycle==19}.

% complement of the commit constraints, the final states of the automata, and the maxcycle to
% verify the constraint from T4 to T2.
complement_commit_constraints(T1,T2,T3,T4,Maxcycle):-
{T2<T4+39};
{T2>T4+50},
{Maxcycle==42}.

% create the constraint system
create_constraint_system(Maxcycle,I,Current_state,YP,XP,T1,T2,T3,T4,C):-
[XN,YN]: integer(0,2),
Next_state: integer(0,5),
[C1,C2,Input,Output]: boolean,
% check stop condition
{I=<Maxcycle},
%  generate the Input of the fsm from present x state XP
{Input==(XP==1)},
% generate the next_state of the fsm
{(Current_state==0)  =< (Next_state==Input)},
{((Current_state>=1) and (Current_state=<3)) =< (Next_state== Current_state + 1)},
{((Current_state>=4) and (Input==0)) =< (Next_state==Current_state -4)},
{((Current_state>=4) and (Input==1)) =< (Next_state==5)},
% generate the Output of the fsm
{(Current_state==0) =< (Output==0)},
{((Current_state>=1) and (Current_state=<4)) =< (Output == ~Input)},
{(Current_state==5) =< (Output==1)},
% generate the next y state form present y state and Output of fsm
{((YP==0) and (Output==0)) =< (YN==0)},
{((YP<>2) and (Output==1)) =< (YN==1)},
{((YP<>0) and (Output==0)) =< (YN==2)},
% assign occurrence times to outputs events
{((YP==0) and (YN==1)) =< (T1==(I-1)*10)},
{((YP==1) and (YN==2)) =< (T2==(I-1)*10)},
% detect an input event
{C1==(T3<(I+1)*10) and (T3>=I*10)},
{C2==(T4<(I+1)*10) and (T4>=I*10)},
% next state function of the X automaton
{(XP==0) =< (XN==C1)},
{(XP==1) =< (XN==XP+C2)},
{((XP==2) and (~C2)) =< (XN==2)},
insert(C1,C,CC),
insert(C2,CC,New_C),
I1 is I+1,
create_constraint_system(Maxcycle,I1,Next_state,YN,XN,T1,T2,T3,T4,New_C).
```

```
% after maxcycle iteration, constraint the final states of the automata and checking the
% consistency of the constraint system while enumerating the boolean variables
create_constraint_system(Maxcycle,I,Current_state,YP,XP,T1,T2,T3,T4,C):-
{I==Maxcycle},
reverse(C,R_C),
enumerate(R_C),
{XP == XF}
{YP == YF}.

% the main clause
ex:-
[T1,T2,T3,T4]: real,
assume_constraints(T1,T2,T3,T4),
complement_commit_constraints(T1,T2,T3,T4,Maxcycle),
write('Maxcycle='),print(Maxcycle),nl,
create_constraint_system(Maxcycle,0,0,0,0,T1,T2,T3,T4,[]),
write('T1='),print(T1),nl,
write('T2='),print(T2),nl,
write('T3='),print(T3),nl,
write('T4='),print(T4),nl.
```

A3. Prolog code of verifying the real-time controller in Chapter 8

```
append(L, [], L).
append(L, [X|Xs], [X|Zs]):- append(L, Xs, Zs).

assume_complement_commit_constrants(MaxI,T,SF):-
MaxI:integer,
[T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20,T21,T22,T23,T24,T25,T26,T27,T28,T
29,T30,T31,T32,T33,T34,T35,T36]=T,

[SFcsn,SFseln, SFrwn, SFtx2upack, SFup_axs, SFdtackn, SFtrin_o]: integer,
Assume_T9_T13: integer(0,10),
Assume_T17_T13: integer(4, 10),
Assume_T13_T14: integer(60, _),
Assume_T14_T10: integer(0, 10),
Assume_T14_T18: integer(0,10),
Assume_T21_T25: integer(0, 280),
Assume_T25_T26: integer(20,20),
Assume_T29_T14: integer(0,40),
Assume_T14_T15: integer(40,_),
Assume_T11_T15: integer(0,10),
Assume_T19_T15: integer(4, 10),
Assume_T15_T16: integer(60,_),
Assume_T16_T12: integer(0, 10),
Assume_T16_T20: integer(0,10),
Assume_T23_T27: integer(0, 600),
Assume_T27_T28: integer(20,20),
Assume_T31_T16: integer(0,40),
{T13==T9+Assume_T9_T13},
{T13==T17+Assume_T17_T13},
{T14==T13+Assume_T13_T14},
{T10==T14+Assume_T14_T10},
{T18==T14+Assume_T14_T18},
{T25==T21+Assume_T21_T25},
{T26==T25+Assume_T25_T26},
{T14==T29+Assume_T29_T14},
{T15==T14+Assume_T14_T15},
{T15 ==T11+Assume_T11_T15},
{T15 ==T19+Assume_T19_T15},
{T16 ==T15+Assume_T15_T16},
{T12 ==T16+Assume_T16_T12},
{T20 ==T16+Assume_T16_T20},
{T27 ==T23+Assume_T23_T27},
{T28 ==T27+Assume_T27_T28},
{T16 ==T31+Assume_T31_T16},
% redundant commit constraints
{T33-T13 =<5}, {T33-T13>=0},
{T34-T14 =<5}, {T34-T14>=0},
{T35-T15 =<5}, {T35-T15>=0},
{T29-T33 =<600}, {T29-T33 >= 40},
{T31-T35 =<600}, {T31-T35 >= 40},
{T21-T14 =< 60}, {T21-T14 >= 0},
{T23-T16 =< 60}, {T23-T16 >= 0},
{T22-T26 =< 40}, {T22 - T26 >= 0},
%{T36-T16 =<5}, {T36-T16>=0},
% final states of the automata
{SFcsn==4},
{SFseln==4},
{SFrwn==4},
{SFtx2upack==4},
{SFup_axs==4},
{SFdtackn==4},
{SFtrin_o==4},
SF=[SFcsn,SFseln, SFrwn, SFtx2upack, SFup_axs, SFdtackn, SFtrin_o],
%max cycle
{MaxI == 90},
%complement of the commit constraint
{T36-T16>5};
{T36-T16<0}.


create_constraint(I, MaxI, SF, Csn_M1, Csn_M2, Seln_M1, Seln_M2, Up_ack,
Up_wrnlocal_R, Up_naxspls_R, Up_axspending_R, SP, SPcsn, SPseln, SPrwn,
SPtx2upack,SPup_axs, SPdtackn, SPtrin_o,T,C, States):-
```

```
I1: integer,
%check stop condition
{I<MaxI},
[SP,SN]: integer(0,3),
[SPcsn, SPseln, SPrwn, SNcsn, SNseln, SNrwn]: integer(0,4),
[SPtx2upack, SNtx2upack]: integer(0,4),
[SPup_axs, SPdtackn, SPtrin_o, SNup_axs, SNdtackn, SNtrin_o]: integer(0,4),

[T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20,T21,T22,T23,T24,T25,T26,T27,T28,T
29,T30,T31,T32,T33,T34,T35,T36]=T,


%Input variables of the system and the registers
[Csn, Seln, Rwn, Tx2upack] : boolean,
[Csn_M1, Seln_M1, Up_wrnlocal, Up_axspending]: boolean,
[Csn_M2, Seln_M2, Up_wrnlocal_R, Up_axspending_R, Up_ack, Up_naxspls_R]: boolean,

%Output variables of the FSM
[Up_wrnlocal_rst, Up_arwlatch, Dtackn_rst, Up_naxspls]: boolean,

%Output variables of the system
[Dtackn, Trin_o, Up_axs]: boolean,

%boolean variables modeling possible input events in the clocl cycle
[C9, C10,C11,C12,C13,C14,C15,C16,C17,C18,C19,C20,C25,C26,C27,C28]: boolean,
% input values
{Csn == ~((SPcsn == 1) or (SPcsn ==3))},
{Seln == ~((SPseln == 1) or (SPseln == 3))},
{Rwn == ~((SPrwn == 1) or (SPrwn == 3))},
{Tx2upack == (SPtx2upack == 1) or (SPtx2upack == 3)},

%Next state of the FSM
```

```
{(((SP == 0) and  (Csn_M2 or Seln_M2)) or ((SP == 3) and Seln_M2) or ((SP == 2) and
Up_wrnlocal_R and Up_ack)) =< (SN == 0)},

{((SP == 1) and ~Seln_M2) or ((SP==0) and ~(Csn_M2 or Seln_M2) and ~Rwn) =< (SN ==
1)},

{((SP==0) and ~(Csn_M2 or Seln_M2) and Rwn) or ((SP == 2) and ~Up_ack) or ((SP == 1)
and Seln_M2) =< (SN == 2)},

{(SP == 2) and ~Up_wrnlocal_R and Up_ack or (SP == 3) and ~Seln_M2 =< (SN == 3)},
write('    SN = '), print(SN),nl,

%Outputs of the FSM
{Up_wrnlocal_rst == ((SP == 2) and  Up_wrnlocal_R and Up_ack)},
{Up_arwlatch == ((SP == 0) and ~(Csn_M2 or Seln_M2))},

{Dtackn_rst == (((SP == 0) and ~(Csn_M2 or Seln_M2) and ~Rwn) or ((SP == 2)
and~Up_wrnlocal_R and  Up_ack))},

{Up_naxspls == (((SP == 0) and  ~(Csn_M2 or Seln_M2) and Rwn) or ((SP == 1) and
Seln_M2))},


 %Outputs of the system
{Dtackn_R==Seln or (~Seln and ~Dtackn_rst)},
{Dtackn == Csn or Dtackn_R},
{Trin_o == ~(Csn or Seln)},
{Up_axs == Up_naxspls or Up_axspending_R},

%Inputs of the registers

{Up_wrnlocal == ~Up_wrnlocal_rst and (Up_arwlatch and ~Rwn or ~Up_arwlatch and
Up_wrnlocal_R)},

{Up_axspending == (Up_naxspls or Up_axspending_R) and  ~Tx2upack},

% the next state of the output waveform transition automata
{((SPup_axs==0) or (SPup_axs==2)) =< (SNup_axs ==SPup_axs+Up_axs)},
{((SPup_axs == 1) or (SPup_axs==3))  and Up_axs  =< (SNup_axs == SPup_axs)},
{((SPup_axs == 1) or (SPup_axs==3))  and ~Up_axs  =< (SNup_axs == SPup_axs+1)},
{(SPup_axs == 4) and ~Up_axs =< (SNup_axs==4)},

{((SPdtackn==0) or (SPdtackn==2)) =< (SNdtackn == SPdtackn+(~Dtackn))},
{((SPdtackn == 1) or (SPdtackn==3))   =< (SNdtackn == SPdtackn+Dtackn)},
{(SPdtackn ==4)  and Dtackn  =< (SNdtackn == 4)},

{((SPtrin_o==0) or (SPtrin_o==2)) =< (SNtrin_o ==SPtrin_o+Trin_o)},
{((SPtrin_o == 1) or (SPtrin_o==3))  and Trin_o  =< (SNtrin_o == SPtrin_o)},
{((SPtrin_o == 1) or (SPtrin_o==3))  and ~Trin_o  =< (SNtrin_o == SPtrin_o+1)},
{(SPtrin_o == 4) and ~Trin_o =< (SNtrin_o==4)},

% timing assignment of output events
{(SPup_axs==0) and (SNup_axs==1) =<(T21==I*20)},
{(SPup_axs==1) and (SNup_axs==2) =<(T22==I*20)},
{(SPup_axs==2) and (SNup_axs==3) =<(T23==I*20)},
{(SPup_axs==3) and (SNup_axs==4) =<(T24==I*20)},

{(SPdtackn==0) and (SNdtackn==1) =<(T29==I*20)},
{(SPdtackn==1) and (SNdtackn==2) =<(T30==min(T14,T10))},
{(SPdtackn==2) and (SNdtackn==3) =<(T31==I*20)},
{(SPdtackn==3) and (SNdtackn==4) =<(T32==min(T16,T12))},

{(SPtrin_o==0) and (SNtrin_o==1) =<(T33==max(T13,T9))},
{(SPtrin_o==1) and (SNtrin_o==2) =<(T34==min(T14,T10))},
{(SPtrin_o==2) and (SNtrin_o==3) =<(T35==max(T15,T11))},
{(SPtrin_o==3) and (SNtrin_o==4) =<(T36==min(T16,T12))},

% possible input evetns
{C9==((T9<(I+1)*20) and (T9>=I*20))},
{C10==((T10<(I+1)*20) and (T10>=I*20))},
{C11==((T11<(I+1)*20) and (T11>=I*20))},
{C12==((T12<(I+1)*20) and (T12>=I*20))},
{C13==((T13<(I+1)*20) and (T13>=I*20))},
{C14==((T14<(I+1)*20) and (T14>=I*20))},
{C15==((T15<(I+1)*20) and (T15>=I*20))},
{C16==((T16<(I+1)*20) and (T16>=I*20))},
{C17==((T17<(I+1)*20) and (T17>=I*20))},
{C18==((T18<(I+1)*20) and (T18>=I*20))},
{C19==((T19<(I+1)*20) and (T19>=I*20))},
{C20==((T20<(I+1)*20) and (T20>=I*20))},
{C25==((T25<(I+1)*20) and (T25>=I*20))},
{C26==((T26<(I+1)*20) and (T26>=I*20))},
{C27==((T27<(I+1)*20) and (T27>=I*20))},
{C28==((T28<(I+1)*20) and (T28>=I*20))},

% the next state of the input waveform transition automata
{(SPcsn==0) =< (SNcsn==C9)},
{(SPcsn==1) =< (SNcsn==SPcsn+C10)},
{(SPcsn==2) =< (SNcsn==SPcsn+C11)},
{(SPcsn==3) =< (SNcsn==SPcsn+C12)},
{(SPcsn==4) =< (C12==0)},
{(SPcsn==4) =< (SNcsn==4)},

{(SPseln==0) =< (SNseln==C13)},
{(SPseln==1) =< (SNseln==SPseln+C14)},
{(SPseln==2) =< (SNseln==SPseln+C15)},
{(SPseln==3) =< (SNseln==SPseln+C16)},
{(SPseln==4) =< (C16==0)},
{(SPseln==4) =< (SNseln==4)},

{(SPrwn==0) =< (SNrwn==C17)},
{(SPrwn==1) =< (SNrwn==SPrwn+C18)},
{(SPrwn==2) =< (SNrwn==SPrwn+C19)},
{(SPrwn==3) =< (SNrwn==SPrwn+C20)},
{(SPrwn==4) =< (C20==0)},
{(SPrwn==4) =< (SNrwn==4)},

{(SPtx2upack==0) =< (SNtx2upack==C25)},
{(SPtx2upack==1) =< (SNtx2upack==SPtx2upack+C26)},
{(SPtx2upack==2) =< (SNtx2upack==SPtx2upack+C27)},
{(SPtx2upack==3) =< (SNtx2upack==SPtx2upack+C28)},
{(SPtx2upack==4) =< (C28==0)},
{(SPtx2upack==4) =< (SNtx2upack==4)},

ap-
pend([C9,C10,C11,C12,C13,C14,C15,C16,C17,C18,C19,C20,C25,C26,C27,C28],C,New_C),

append([SPcsn, SPseln, SPrwn, SPtx2upack, SPup_axs, SPdtackn, SPtrin_o], States,
New_States),

I1 is I+1,

create_constraint(I1, MaxI, SF, Csn, Csn_M1, Seln, Seln_M1, Tx2upack, Up_wrnlocal,
Up_naxspls, Up_axspending, SN, SNcsn, SNseln, SNrwn, SNtx2upack,SNup_axs, SNdtackn,
SNtrin_o,T,New_C, New_States).

create_constraint(I, MaxI, SF, Csn_M1, Csn_M2, Seln_M1, Seln_M2, Up_ack,
Up_wrnlocal_R, Up_naxspls_R, Up_axspending_R, SP, SPcsn, SPseln, SPrwn,
SPtx2upack,SPup_axs, SPdtackn, SPtrin_o, T,C, States):-

{I==MaxI},
[SFcsn, SFseln, SFrwn, SFtx2upack, SFup_axs, SFdtackn, SFtrin_o]=SF,
{SPcsn == SFcsn},
{SPseln == SFseln},
{SPrwn == SFrwn},
{SPtx2upack == SFtx2upack},
{SPup_axs == SFup_axs},
{SPdtackn ==SFdtackn},
{SPtrin_o == SFtrin_o},
enumerate(C),
write('The End'),nl.

ex:-
I: integer,

[T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20,T21,T22,T23,T24,T25,T26,T27,T28,T
29,T30,T31,T32,T33,T34,T35,T36]: integer(0,_),

T=[T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20,T21,T22,T23,T24,T25,T26,T27,T2
8,T29,T30,T31,T32,T33,T34,T35,T36],

assume_complement_commit_constrants(MaxI,T,SF),
create_constraint(0, MaxI, SF, 1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,T,[],[]),
write('T9='),print(T9),nl, write('T10='),print(T10),nl,
write('T11='),print(T11),nl, write('T12='),print(T12),nl,
write('T13='),print(T13),nl, write('T14='),print(T14),nl,
write('T15='),print(T15),nl, write('T16='),print(T16),nl,
write('T17='),print(T17),nl, write('T18='),print(T18),nl,
write('T19='),print(T19),nl, write('T20='),print(T20),nl,
write('T21='),print(T21),nl, write('T22='),print(T22),nl,
write('T23='),print(T23),nl, write('T24='),print(T24),nl,
write('T25='),print(T25),nl, write('T26='),print(T26),nl,
write('T27='),print(T27),nl, write('T28='),print(T28),nl,
write('T29='),print(T29),nl, write('T30='),print(T30),nl,
write('T31='),print(T31),nl, write('T32='),print(T32),nl,
write('T33='),print(T33),nl, write('T34='),print(T34),nl,
write('T35='),print(T35),nl, write('T36='),print(T36),nl.
```