Computer Science Department
University of Crete

An EDA Tool for the Timing Analysis, Optimization and Timing Validation of Asynchronous Circuits

Master's Thesis

Kasapaki Evangelia

April 2008
Heraklion, Greece

An EDA Tool for the Timing Analysis, Optimization and Timing Validation of Asynchronous Circuits

by

Evangelia Kasapaki

Master's Thesis

Department of Computer Science
University of Crete

Abstract

Synchronous circuits have enjoyed, since the mid-80's, a constantly maturing EDA tool/flow framework, which enabled the implementation of multi-million transistor chips, and sustains the pace of the electronics industry. The cornerstones of EDA, which triggered its wide adoption are twofold, i.e. Timing Analysis and Timing Analysis-Driven Optimization. Unfortunately, conventional Static Timing Analysis cannot be directly applied to asynchronous circuits, as the latter are closed-loop systems.

The primary reason why Asynchronous Design approaches are not attempted today is the lack of any viable and complete EDA flow. This work presents a complete Asynchronous Timing Analysis algorithm implementation, suitable for EDA, which is capable of analyzing the timing of any asynchronous circuit. This work also demonstrates closed-loop Timing Analysis-Driven optimization for asynchronous circuits.

The TA algorithm has its foundations in prior theoretical work on algorithms for deriving accurate bounds for the separation time between events of concurrent systems. Several insufficient and incomplete aspects of that work were clarified, completed and improved and a complete and efficient implementation has been achieved.

Results on several asynchronous circuits demonstrate the viability of the implemented algorithm, and the capability to automatically optimize selectively the timing-critical subparts of an asynchronous circuit for timing and the other non-timing critical subparts for area.

Thesis Supervisor: Manolis Katevenis, Professor

Thesis Vice-Supervisor: Christos P. Sotiriou, Collaborating Researcher ICS-FORTH

Εργαλείο ΕΔΑ για Χρονική Ανάλυση, Βελτιστοποίηση και Χρονικής Επικύρωσης Ασύγχρονων Κυκλωμάτων

Ευαγγελία Κασαπάκη

Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Περίληψη

Τα σύγχρονα κυκλώματα, από τα μέσα της δεκαετίας του ′80, απολαμβάνουν μια διαρκώς ωριμάζουσα ροή εργαλείων Ηλεκτρονικού Σχεδιαστικού Αυτοματισμού (EDA), η οποία αρχικά κατέστησε δυνατή την υλοποίηση ολοκληρωμένων κυκλωμάτων με πολλά εκατομμύρια τρανζίστορ, ενώ σήμερα διατηρεί τον ρυθμό ανάπτυξης της ηλεκτρονικής βιομηχανίας. Οι δύο ¨ακρογωνιαίοι λίθοι' του EDA είναι η Χρονική Ανάλυση καί η Βελτιστοποίηση των κυκλωμάτων βάση αυτής. Δυστυχώς, η συμβατική Στατική Χρονική Ανάλυση, η καθιερωμένη δηλαδή διαδικασία, δεν μπορεί να εφαρμοστεί άμεσα σε ασύγχρονα κυκλώματα, καθώς τα τελευταία περιέχουν πάντα ανάδραση και έτσι είναι κυκλικά, δηλαδή κλειστού βρόχου.

Ο κύριος λόγος για τον οποίο δεν χρησιμοποιούνται προσεγγίσεις Ασύγχρονης Σχεδίασης είναι η έλλειψη ολοκληρωμένων και βιώσιμων αυτοματοποιημένων ροών. Η παρούσα εργασία παρουσιάζει έναν ολοκληρωμένο αλγόριθμο για Ασύγχρονη Χρονική Ανάλυση, κατάλληλο για EDA, που μπορεί να εφαρμοστεί για την ανάλυση του χρονισμού οποιουδήποτε ασύγχρονου κυκλώματος. Επιπλέον, παρουσιάζεται διαδικασία βετιστοποίησης, η οποία εκτελείται σε κλειστό βρόγχο και καθοδηγείται από τη χρονική ανάλυση.

Ο συγκεκριμένος αλγόριθμος χρονικής ανάλυσης έχει τα θεμέλιά του σε προηγούμενη θεωρητική εργασία, η οποία αφορούσε ανάπτυξη αλγορίθμων υπολογισμού ορίων ως προς τη χρονική απόκλιση συμβάντων παράλληλων συστημάτων. Σημαντικός αριθμός από ελλείψεις και ανεπάρκειες της προηγούμενης εργασίας διασαφηνίστηκαν, ο ορισμός και τα χαρακτηριστικά του αλγορίθμου ολοκληρώθηκαν αλλά και βελτιώθηκαν, και επιτεύχθηκε τελικώς μία αποδοτική υλοποίηση.

Αποτελέσματα από την ανάλυση διάφορων ασύγχρονων κυκλωμάτων επιδεικνύουν την αποδοτικότητα του υλοποιημένου αλγορίθμου και την ικανότητά του να βελτιώνει αυτόματα και επιλεκτικά, τα μέν χρονικώς κρίσιμα τμήματα ενός ασύγχρονου κυκλώματος ως προς την καθυστέρηση,

τα δέ υπόλοιπα τμήματα ως προς το εμβαδό.

Επόπτης Μεταπτυχιακής Εργασίας: Μανόλης Κατεβαίνης, Καθηγητής
Επιβλέπων Μεταπτυχιακής Εργασίας: Χρήστος Σωτηρίου, Συνεργαζόμενος Ερευνητής ΙΠ-ΙΤΕ

Acknowledgments

Firstly, I would like to thank my parents for their support all these years. If it was not for their help I would have never made it to here.

I would like to recognize the contribution of my supervisor, Dr. Christos Sotiriou and thank him for his guidance and support, throughout this work. Additionally, i would like to thank all the people with whom I have worked during this thesis for their constructive remarks and sharing of ideas.

This work was conducted in collaboration with the ICS-FORTH and funded by it.

Last but not least, I would like to thank my friends for their support, through difficult times.

Contents

1 Introduction **1**

2 Timing Analysis of Asynchronous Circuits **3**

2.1 Electronic Design Automation (EDA) and Hardware Design 3

2.2 Timing Analysis Methods 4

2.3 Specification Models for Asynchronous Timing Behavior 5

2.3.1 Petri-Nets 5

2.3.2 STGs 6

2.3.3 Event Rule (ER) System 7

2.4 Timing Separation of Events 7

3 Timing Separation of Events in Concurrent Systems **11**

3.1 Specification Model 11

3.1.1 Execution Modeling 12

3.1.2 Problem Definition 13

3.2 Acyclic TSE Algorithm 14

3.2.1 m-values and M-values 14

3.2.2 Acyclic TSE Algorithm 15

3.3 TSE Algorithm 16

3.3.1 Backwards Unfolding and Functions 16

3.3.2 Bounding the Maximum Separation Time 18

3.3.3 Repetition Parameters 19

3.3.4 Function Matrices 20

3.4 Open issues 21

4 Analysis, Clarifications and Improvements on the Timing Separation of Events (TSE) Algorithm **23**

4.1 Specification Model . . . 24
4.2 Critical Cycles and Repetition Parameters . . . 24
4.2.1 Ratio'ed Cycles and Maximum Ratio Cycles . . . 25
4.2.2 Calculation of ϵ^* . . . 26
4.2.3 Calculation of k^* . . . 27
4.2.4 Examples . . . 28
4.3 Cutsets . . . 29
4.4 Closure . . . 30
4.4.1 Closure in Closed Semirings . . . 30
4.5 Relation Matrices . . . 31
4.6 Minimum Separation Analysis . . . 32
4.6.1 m-values and M-values . . . 32
4.6.2 Cycles and Repetition Parameters . . . 34
4.6.3 Functions . . . 35
4.7 Implementation and Complexity . . . 36
4.7.1 Acyclic TSE Version . . . 36
4.7.2 Complete TSE Version . . . 37
4.7.3 Minimun Analysis . . . 41
4.7.4 Floating Point Arithmetic . . . 41
4.8 TSE Analysis vs. worst-case analysis . . . 41

5 Application of TSE tool **43**

5.1 Optimization . . . 43
5.1.1 Circuit Specification . . . 45
5.1.2 Asynchronous Timing Analysis (ATA) and Optimization . . . 48
5.2 Relative Timing Constraints (RTC) Validation . . . 49
5.3 An automated tool for optimization . . . 49
5.3.1 Technology Library . . . 49
5.3.2 Input Netlist . . . 50
5.3.3 Input Signal Transition Graph (STG) . . . 51
5.3.4 Optimization Process . . . 51

6 Results **55**

6.1 Experimental Procedure 55

6.2 Two-phase Overlapping Desynchronization Controller 57

6.2.1 Scale of 1 ring controller 58

6.2.2 Scale of 3 ring controllers 59

6.2.3 Scale of 3 ring controllers including wire delays 60

6.2.4 Fork-join Pipeline Structure 60

6.3 Relative-Timed Burst Mode (RTBM) Controller 61

6.4 C-Muller Pipeline 67

6.5 VME Bus controller 67

7 Conclusions **71**

References **72**

List of Figures

2.1 Transition firing in a Petri-net 6
2.2 xyz STG Model 7
2.3 An ER System 8

3.1 An example Process Graph. 12
3.2 Unfolded graph of Figure 3.1 with upper delay bound D assignment to all events. . . 13
3.3 Unfolded graph of Figure 3.1 with m- and M-value annotation. 16
3.4 Portion of the backwards unfolded process graph for the process graph 17
3.5 Bounding TSE 18

4.1 Portion of the backwards unfolded graph for the Process Graph of Figure 3.1 labeled with m-values for $s_{(\beta)}$=$a_{(1)}$ 25
4.2 Process graph G^* of maximum ratio cycles on two different strongly connected componets. The edges in dotted lines are the edges of the initial process graph G' that are not part of a maximum ratio cycle. 27
4.3 Process graph G^* of maximum ratio cycles with two cycles having a common event. 27
4.4 Process graph G_2 29
4.5 Unfolded process graph labeled with m-values for $s_{(\beta)}$=$a_{(1)}$ 29
4.6 Unfolded process graph labeled with m-values for $s_{(\beta)}$=$b_{(1)}$ 29
4.7 Time axis for maximum separation analysis. 33
4.8 Time axis for minimum separation analysis. 34
4.9 Evaluation of m- and M-values in maximum and minimum separation analysis. . . 35
4.10 Unfolded process graph with worst-case timing analysis annotation 42
4.11 Unfolded process graph with maximum timing separation analysis between events a_5 and a_6 42

5.1 Optimization flow using ATA. 44
5.2 Optimization flow using ATA. 45
5.3 controller netlist 46
5.4 State Graph Analysis of the controller 46
5.5 Signal Transition Graph of RTBM controller 47
5.6 Validation flow using ATA. 50
5.7 Optimization tool process 52

6.1 Experimental Procedure Flow. 56
6.2 Desynchronization controller STG 58
6.3 Desynchronization controller netlist 58
6.4 Two-phase desynchronization controller netlist in 1-scale ring 59
6.5 Two-phase desynchronization controller netlist in 3-scale ring 59
6.6 A fork-join desynchronization controllers' structure 62
6.7 Time/area results of desynchronization controller in 1-scale ring. 63
6.8 Time/area results of desynchronization controller in 3-scale ring. 63
6.9 Time/area results of desynchronization controller in 3-scale ring, with wire delay. . . 64
6.10 Time/area results of desynchronization controller in fork-join pipeline. 64
6.11 RTBM controller netlist in 1-scale ring topology 65
6.12 Signal Transition Graph of RTBM controller 65
6.13 Time/area results of RTBM controller in 1-scale ring. 66
6.14 Time/area results of RTBM controller in 4-scale ring. 67
6.15 C-muller gate implementation 68
6.16 C-muller pipeline 68
6.17 Time/area results of a 4-scale c-Muller pipeline. 69
6.18 STG of a read operation in a vme bus controller 69
6.19 Time/area results of a VME Bus Controller. 70

List of Tables

6.1 Desynchronization controller in 1-scale, 3-scale ring and 3-scale ring including wire delays results. 61
6.2 Desynchronization controller in fork-join pipeline results. 62
6.3 RTBM controller in 1-scale and 4-scale ring results. 66
6.4 C-Muller pipeline results 68
6.5 VME Bus controller results 70

1 Introduction

The defacto methodology for digital circuit design and implementation is Synchronous Timing, which requires the existence of the clock signal. Clock signals enable the circuits' synchronization and the clock cycle is the minimum time unit. The performance of any synchronous circuit is, therefore, determined by it. Based on the notion of clock cycle, a well-defined and highly automated procedure is implanted in contemporary Electronic Design Automation (EDA). This constitutes synchronous design as highly appealing methodology.

On the other hand, asynchronous design, while theoretically present throughout the years, still has not been concretely defined and developed in an EDA flow.In asynchronous design, the notion of the clock is absent and the synchronization is local based on handshakes.This indicates a dynamic functionality and a more fine-grained control of the circuit, resulting in a highly demanding design procedure, but also some key advantages over the synchronous one [1]. These are in terms of

- performance: their dynamic behaviour takes advantage of the average case of operation.
- variability tolerance: their timing can adapt through variations in manufacturing process, temperature or voltage
- reduced electromagnetic emissions.

The complex process required for asynchronous design imposed limits on their wide acceptance. The main obstacle in the use of asynchronous design is the absence of EDA tools used in the design and implementation process. In contrast, there is a variety of such tools developed for synchronous design. Using the existing EDA tools developed for synchronous design, to implement asynchronous circuits is challenging. The latter were developed focusing on specific aspects of synchronous design, therefore either providing inadequate information or being non-applicable alltogether. This demonstrates the need for developing EDA tools specifically targeted to asynchronous circuits.

The aim of this master's thesis has been the development and implementation of a Timing Analysis algorithm in the field of EDA tools for asynchronous circuits. A main requirement of this algorithm has been the ability to handle cyclic circuits, a characteristic of asynchronous circuits. An algorithm for calculating exact bound on the timing separation of events that was identified. This work analysed, improved and extended unexplored areas, leading from theoretical analysis to the implementation of a tool for the timing analysis of asynchronous circuits. This tool for the timing analysis was incorporated in a simple mapping tool, also implemented in this work, performing optimizations through gate resizing. The technique was applied to some asynchronous controllers and the results were compared to the ones that came from existing tools for timing analysis, synthesis and optimization of synchronous circuits.

This master's thesis is organized as follows: In Chapter 2, a review on timing analysis is presented and how it is applied for synchronous and asynchronous circuits. Additionally, specification models for the timing behavior of asynchronous circuits are examined, as well as performance metrics for their performance evaluation. In Chapter 3, previous work on an algorithm for the timing separation of events in concurrents systems is presented, with the issues that are left incomplete or unclear. In Chapter 4, these issues are addressed, as they were studied in this thesis, leading to a complete picture of the algorithm. An implementation of an EDA Timing Analysis (TA) tool for asynchronous circuits is, also, presented, which is based on the particular algorithm, with its complexity estimation. In Chapter 5, applications of this TA tool are examined, and flows are proposed for the optimization of asynchronous circuits and validation of Relative Timing Constraints (RTC). The implementation of an optimization tool is also described. Optimization of several asynchronous circuits was performed, using the implemented optimization tool in the proposed flow and the results, as well as the experimental procedure are presented in Chapter 6. Finally, the conclusions drawn from the described work with some suggestions for future work are included in Chapter 7.

2

Timing Analysis of Asynchronous Circuits

This chapter covers the subjects of Timing Analysis and Performance Evaluation for Asynchronous circuits. Methods for Timing Analysis are presented and alternative approaches suitable for asynchronous design are examined. Due to the particular characteristics of asynchronous design, different specification models and different performance metrics are required, some of which are presented in this chapter.

2.1 EDA and Hardware Design

The development of hardware design in recent years owes its improving to the use of EDA tools. The success of EDA tools is due to the automated procedure that provides in the hardware design process. The designer is provided with the means to reach an implementation without being concerned with every specific aspect of the design implementation, whereas they can focus on architectural and designing issues. A description of the circuit is required in a Hardware Description Language (HDL) form and by specifying some parameters and using a technology library a circuit can be automatically synthesized by an EDA tool. In other words, the specification is made independent of the implementation but the implementation is conformed by it.

Moreover, various implementations can be derived from the same system specification, with dif-

ferent characteristics, *i.e.* minimum area, performance and power. The space of all possible implementations is explored using an EDA tool, resulting to the most suitable for each case, *i.e.* EDA is the means to move along the Pareto curve through different implementations of the same design. Additionally, the specification is independent from the implementing technology, due to the use of technology libraries in synthesizing. Thus, changes in technology are easily imported in designs, as they don't require re-designing of every system for the new technology, only by re-synthesizing the system using the new library.

It is apparent that an EDA tool is not merely a translator of a circuit specification to the implementation. It is a tool that performs various complex procedures, during implementation. Apart from mapping functionality to implementation units, it performs analysis on the system for various characteristics, such as for timing, area or power, exploring trade-offs to reach an optimized implementation, for a given set of constraints. For the development of an EDA tool numerous algorithms need to be examined and implemented.

Contemporary EDA tools focus on synchronous hardware design. Various EDA tools have been developed focusing on specific aspects of synchronous design. This is not the case with asynchronous hardware design. Asynchronous design is not used today, due to the fact that a complete and mature design flow does not exist. Hence, designing of asynchronous circuits requires manual work on implementation and trade-off exploration.

The most important factor both in the design and in the implementation process, is the circuit's timing. Apart from providing information for the attainable performance of a specific implementation, it provides guidelines for timing-drive optimizations. Thus, a TA engine for asynchronous circuits is a cornerstone for any viable asynchronous EDA flow.

2.2 Timing Analysis Methods

In synchronous design timing analysis of circuits is performed by Static Timing Analysis (STA). STA considers all the paths of the circuit netlist and identifies a path of maximum delay. Specifically, the circuit is partitioned into stages of combinational logic separated by registers. Each combinational logic stage is considered and analyzed individually. The performance metric is the maximum clock cycle time, *i.e.* the longest path of any combinational stage, required to complete a computation. STA is typically performed by a Depth-First-Search (DFS) algorithm

Another method for performing timing analysis in synchronous design aiming at multi-corner estimations is Statistical Static Timing Analysis (SSTA). This method follows the same principles as STA but incorporates probability in the delay of gates. STA considers one corner of analysis usually worst-case, while SSTA considers delays with a probability, according to statistical information.

Reseach studies the probability relations when following a path, and the best statistical model.

In asynchronous design, no well-established methodology for timing analysis exists. Commonly, timing analysis is performed by applying point-to-point STA. In asynchronous circuits the processing and forwarding of computational data is controlled locally through handshaking [1], *i.e. event-driven control*. Therefore, the control along stages is determined by the timing of each computational stage and the timing of its neighbors. Moreover, asynchronous timing depends on the interaction with the environment and may exhibit transient behaviour during timing state changes. This implies a large amount of timing dependencies throughout the circuit, and furthermore dependencies are cyclic and closed-loops, as signals often depend back on themselves. The STA model does not account for cyclic dependencies, as it cuts the dependencies whenever they are located throughout a netlist. Whenever, a cyclic dependency is identified, the loop is cut in a non-deterministic way, *i.e.* the point of cutting depends on the sequence the netlist is traversed and not on the netlist itself. In any case, the cutting of a dependency leads to a timing estimation error.

Another way of performing timing analysis in asynchronous circuits is by simulation. However, this serves only performance evaluation and doesn't provide the means for optimizations, moreover to form the basis for an EDA tool.

2.3 Specification Models for Asynchronous Timing Behavior

As asynchronous circuits are essentially hardware implementations of concurrent systems, their behavior and its specification are often modelled by concurrent models of computation, such as Petri-nets [2] or Process Algebras. This work focuses on Petri-net modeling methods, *i.e.* graph modelling.

2.3.1 Petri-Nets

A Petri Net is a formal model for the description and analysis of concurrent systems, such as asynchronous circuits, distributed and parallel systems. A Petri Net defines system behavior as a directed graph with two kinds of nodes, places and transitions, and tokens that represent data values that can move through the arcs of the graph. It is widely used for the specification in asynchronous design.

Specifically, a Petri Net is a directed, weighted graph with two kinds of nodes, places and transitions and an initial marking. Arcs of the graph are either from a place to a transition or from a transition to a place. As it appears in the example Petri-net of Figure 2.1 places are represented as circles, while transitions are represented as bars or boxes. A marking of the Petri-net assigns a number of tokens in places and represents the current state of the system. They are represented as black dots in places.

Tokens can move independently from each other through the arcs of the graph changing the state of

the system. The presence of a token in a place represents that a data value satisfies the corresponding constraint. Each transition has a set of input places, *i.e. pre-conditions*, and a set of output places, *i.e. post-conditions*. A transition is enabled, *i.e.* the corresponding event may occur, only if all the preconditions are satisfied, *i.e.* only if all the input places contain at least one token. The firing of the transition, *i.e.* the actual occurrence of the event, takes place with the removing of one token from all the input places and placing one token in every output place, resulting in a new marking, *i.e.* state [2].

Figure 2.1 shows a Petri-net with an enabled transition on the left. Event or transition a may occur since preconditions $p1, p2$ are satisfied. The transition has fired in the Petri-net on the right.

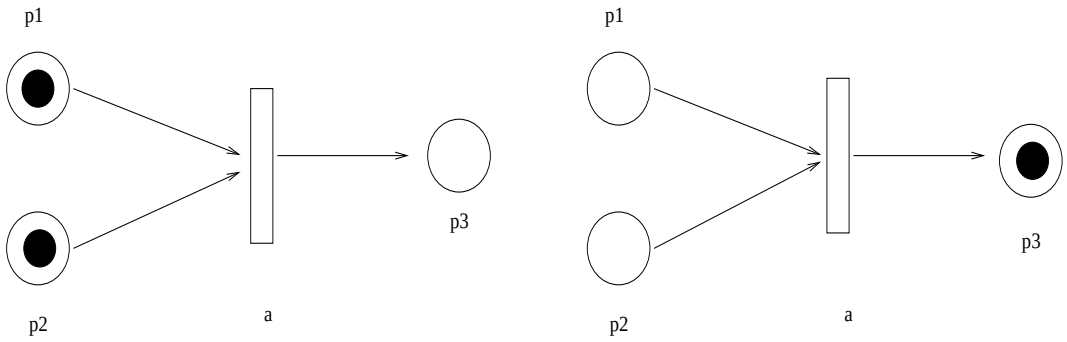

Figure 2.1: Transition firing in a Petri-net

Petri-nets are complex mathematical models to specify any behavior. Usually, subclasses of Petri-nets are used, according to specific applications. Some classes are described here:

State Machines Petri-nets such that each transition has exactly one input place and exactly one output place.

Marked Graphs Petri-nets such that each place has exactly one input transition and exactly one output transition.

2.3.2 STGs

Signal Transition Graphs (STGs) are also graph models able to represent the concurrent behavior and causality in asynchronous circuits, as in other concurrent systems. They are a particular type of Petri-nets, specifically they are Marked Graphs with no choice. Transitions are associated with changes of the values of binary variables.

Since STGs are Marked Graphs, *i.e.* each place has a single input and a single output transition, places can be omitted. Thus, an Signal Transition Graph (STG) is represented by a graph, where the nodes represent binary transitions of signals, while the arcs represent the causality of two events.

Inheriting form the definition of Petri-nets, STGs contain tokens, placed to arcs of the graph. A token placed on an arc r, means that the source transition of r has fired, and the target transition of r is enabled to fire. A transition is enabled to fire when all arcs that reach this transition contain a token. The firing of this transition takes place with the removing of one token from each input arc and the placing of one token in each output arc. The tokens that appear in the specification of the system represent its initial state, *i.e.* its initial marking [3].

Figure 2.2 shows an example of an STG. This STG specifies that, initially, transition $x+$ will fire. Since, $x+$ has fired, transitions $z+$ and $y+$ are enabled and they may fire concurrently. Moreover, transition $z-$ will be enabled as soon as transitions $x-$ $y+$ fire.

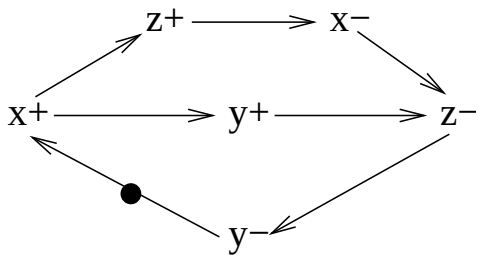

Figure 2.2: xyz STG Model

2.3.3 Event Rule (ER) System

An ER System is a Marked Graph labeled with a delay range on its arcs. As in STGs places can be, also, omitted. Transitions or events, in general, are represented as nodes in the graph. The arcs of the graph represent the causality between events, as well as timing constaints specified by the delay ranges on the arcs. The STG can be translated into an ER System, by switching each transition to a single event, removing tokens and assigning each arc with a delay range, as it appears in Figure 2.3

The practical meaning of an arc, for example $a \rightarrow b$, is that for event b to occur, event a must have occurred first. Delay range $[d1, D2]$ of the arc means that since event a occurs, the necessary processing for event b to occur requires at least $d1$ and at most $D2$ time units. Accordingly, for event e to occur, events c and d must have occurred and at least $max(d3, d5)$ time and at most $max(D3, D5)$ time must have passed.

2.4 Timing Separation of Events

After defining the specification model for a design behavior, a timing metric must be specified for the performance evaluation. One possible performance metric is the total time required for a computation. However, this depends on the characteristics and the complexity of the computation. Moreover, it

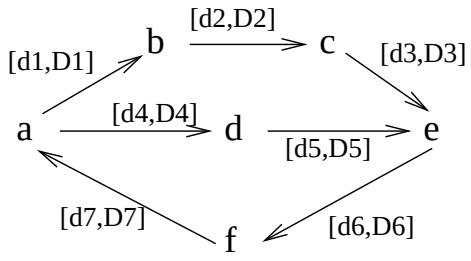

Figure 2.3: An ER System

can't provide an estimation on how much time is consumed in various parts of the design. Another possibility is the time between consequent occurrences of an event or transition, *i.e.* cycle period of the signal, or generally the time between any two events. The cycle period of signals is used as a performance metric for concurrent systems.

Timing separation between events can be a useful tool in asynchronous design process. It can form the basis for a timing analysis tool and moreover, can be used for optimization. Performing timing analysis on a circuit specification provides information and directions for optimization. Optimization functionality can be applied through a closed loop. In every iteration timing analysis is performed and optimizations are applied according to the given directions. Once the optimizations are applied, the Timing Separation of Events (TSE) analysis step confirms the effectiveness of the changes and provides directions for further optimizations.

TSE analysis can also be used for RTC validation. RTC are assumptions about the relative timing of events. Such assumption can lead to simplifications on the specification of designs. A simplified specification with RTC can lead to improved implementation of circuits [4]. The implemented circuit, however, will only operate correctly, if the RTCs are valid, which may not be guaranteed by all stages of the implementation. TSE analysis can be applied after the implementation of the circuit and based on the timing of the specific implementation to prove the validity of the assumed constraints.

An alternative of timing separation of events for the timing analysis would be unfolding the circuit several times and applying DFS algorithms for traversing the circuit. However, this accounts only for a single value in the delay of each component. Moreover, it isn't clear the required number of unfoldings until an accurate estimation is reached. An other alternative is by Petri-net simulation. A Petri-net is simulated starting from initial marking and following each possible state change caused by the moving of each token. The great amount of possible states imposes a high complexity on the solution resulting in an Integer Linear Programming (ILP) problem, which is NP-complete. Thus, attetntion is drawn at TSE.

In [5] an algorithm for the evaluation of bounds on the timing separation of events is presented. It considers ER Systems with concurrent behavior and delay ranges on constraints and evaluates tight bounds on the timing separation of events. This algorithm can handle cyclic graphs and can also take into account infinite execution of the system. This is achieved using algebraic models and structures that will be analyzed in following chapters.

3

Timing Separation of Events in Concurrent Systems

Timing Separation of Events (TSE) appears as important metric for the analysis and evaluation of asynchronous designs. Alternative approach of timing analysis appear unattractive, so an algorithm for the TSE. The algorithm studied, as it is presented in [5], concerns the theoretical analysis behind the timing separation of events in concurrent systems. The specific algorithm handles graphs with cyclic dependencies, a characteristic of concurrent systems such as asynchronous circuits, scheduling protocols, parallel systems and others. Additionally, it considers the theoretical infinite execution of such a system, calculating tight minimum and maximum bounds on the timing separation of any set of events. However, [5] focuses on the evaluation of timing separation as a theoretical problem and was incomplete in certain aspects with respect to providing a complete picture of the algorithm and for leading to an efficient implementation. In this Chapter the algorithm for the timing separation of events is presented as it appears in [5], along with its inefficiencies.

3.1 Specification Model

The specification model describing the concurrent system that is considered in [5] is the Process Graph. The Process Graph is an ER System and it is defined as:

Definition 1 *A PG is is a directed graph $G' = \langle E', R' \rangle$, where:*

- *E' is a finite set of events, the vertices*
- *R' is a set of rule templates, the arcs*

Each arc is labelled with a delay range $[d, D]$ and an occurrence index offset ϵ. The delay range has integer bounds ($0 \leq d \leq D$) and the occurrence index offset $\epsilon \in \mathbb{Z}$, indicates the index difference between the events for a rule template, *i.e.* the rule connects events with ϵ unfoldings difference. For example, an index difference of 1 between two events indicates that the final event depends on the occurrence of the initial event of one unfolding back. The set of events always includes a unique static event, called $root$, which is used to specify the initial state of events of the system. Event $root$ must reach every other event in the Process Graph and no event is allowed to reach $root$. Moreover, the specific analysis considers connected Process Graphs that have $\epsilon(c) > 0$ for all cycles c, *i.e.* the sum of all ϵ values of the arcs of cycle c ($\epsilon(c)$) must be positive.

A simple example of a Process Graph appears in Figure 3.1. The vertical lines on the edges represent the number of the occurrence index offset ϵ.

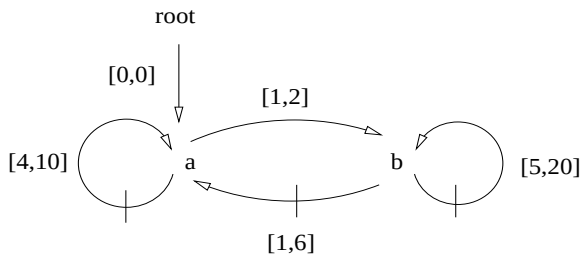

Figure 3.1: An example Process Graph.

The semantics of a process graph modeling a concurrent system dictate that an event v may only occur when all the events that posses a rule leading to the event v have occurred, according to the timing and indexing constraints. For example, for a rule $r : u \rightarrow v$ with $[d, D]$, then the constraints imposed by the rule r require time $t : d \leq t \leq D$, after event u has occurred, before event v may occur.

3.1.1 Execution Modeling

The execution of the system can be modeled by a process graph by unfolding the graph to an acyclic directed graph. This process is called the execution of the Process Graph. Each event of the process graph may occur several times on the execution, so every event in the unfolded graph is labeled with

an occurrence index. The first occurrence of an event v is labeled with occurrence index 0 (v_0), the second with 1 (v_1), *etc.* Additionally, the edges with an occurrence index offset $\epsilon = \beta$ connect events with an index difference β.

Moreover, the occurrence of an event v_k implies a time of occurrence, denoted as $\tau(v_k)$. Therefore, a timing assignment defines an execution instance of a process graph. This timing assignment must be consistent with the timing constraints of the rules. According to the semantics of the execution this timing assignments must satisfy the following equation:

$$max\{\tau(u_{k-\epsilon}) + d | u_{k-\epsilon} \rightarrow u_k \in R\} \leq \tau(v_k) \leq max\{\tau(u_{k-\epsilon}) + D | u_{k-\epsilon} \rightarrow u_k \in R\} \quad (3.1)$$

An execution, of the process graph of Figure 3.1 appears in Figure 3.2. The specific timing assignment is based on the upper delay bound for all the rules of the edges of the graph. By looking at the timing assignment, it is apparent that the difference between same event occurrences, initially, change, as we move towards further unfoldings and, finally, reaches a point where the difference is stabilized.

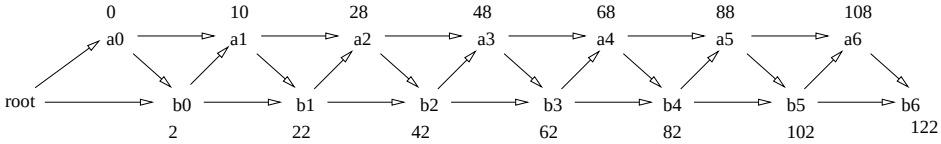

Figure 3.2: Unfolded graph of Figure 3.1 with upper delay bound D assignment to all events.

3.1.2 Problem Definition

The problem of calculating the maximum or minimum timing separation of events on a process graph with delay ranges on the arcs requires identifying the bounds that limit the timing separation in a range. So, considering a source event s and a target event t in E' with an occurrence index separation β, the minimum/maximum timing separation will be the difference $\tau(t_k) - \tau(s_{k-\beta})$, according to delay assignments. Minimum and maximum integers δ and Δ can be determined that bound this difference as

$$\delta \leq \tau(t_k) - \tau(s_{k-\beta}) \leq \Delta \quad (3.2)$$

The problem that is addressed in [5] is the evaluation of the maximum timing separation between two events, *i.e.* evaluating an upper bound Δ on the difference in the above equation. The key idea is to choose a timing assignment that will maximize the difference $\tau(t_k) - \tau(s_{k-\beta})$. The timing assignment to do that is one that forces the source event to occur on its earliest possible time and

(based on this timing assignment to force) the terminal event to occur on its latest possible time, for all possible occurrences.

It is claimed in [5] that minimum separation analysis can be addressed as a maximum separation analysis with a mathematical transformation in the equation, *i.e.* as

$$\tau(s_k) - \tau(t_{k-(-\beta)}) < -\delta$$

However, the formulation presented is incomplete and insufficient to minimum separation analysis. This problem was addressed in this thesis and is presented in the next chapter.

3.2 Acyclic TSE Algorithm

Two approaches are followed for the evaluation of an upper bound on TSE analysis. The first and simplest considers only a portion of an execution instance of a system. Based on this portion, the .. addresses the TSE ... of identifying the upper bound Δ_α for the timing separation of two specific event occurrences, $s_{\alpha-\beta}$ and t_α. For example, considering events a and b of the process graph of Figure 3.2, as source and target, respectively, and an index separation $\beta = 0$, the separation Δ_1 can be evaluated as an upper bound on $\tau(b_1) - \tau(a_1)$. Accordingly, the separation Δ_2 can be evaluated as upper bound on $\tau(b_2) - \tau(a_2)$, the Δ_i as $\tau(b_i) - \tau(a_i)$, *etc*.

3.2.1 m-values and M-values

Intuitively, the algorithm must identify a timing assignment that forces the source event to occur at the earliest possible time and based on this timing assignment to force the terminal event to occur at the latest possible time. In [5] the earliest and latest execution time are modeled through timing variables m- and M-values, in relation to the source and target event, respectively. Specifically, the m-values of an event v_k represent the maximum offset delay between the event in question and the source event, using the lower delay bounds of the edges. So, they are calculated in relation to the source event $s_{\alpha-\beta}$ as follows:

$$m(v_k) = max\left\{d(h) \mid \text{all paths} v_k \xrightarrow{h} s_{\alpha-\beta}\right\} \tag{3.3}$$

The M-values of an event v_k represent the maximum offset delay between the event in question and the target event, using the upper delay bounds of the edges. They are calculated in relation to previous occurrences of other events, through the following equation.

$$M(v_k) = \begin{cases} max\left\{min(0, M(u_j) + D + m(v_k) - m(u_j)) \mid u_j \xrightarrow{[d,D]} v_k\right\} & \text{if } v_k \text{ has path to } s_{\alpha-\beta} \\ max\left\{M(u_j) + D + m(v_k) - m(u_j) \mid u_j \xrightarrow{[d,D]} v_k\right\} & \text{if } v_k \text{ has no path to } s_{\alpha-\beta} \end{cases}$$

3.2.2 Acyclic TSE Algorithm

The acyclic algorithm determines the maximum separation between two specific event occurrences $s_{\alpha-\beta}$ and t_α of a finite portion G_α of the unfolded process graph, where α is the number of unfoldings. The steps of the algorithm are:

1. From $s_{\alpha-\beta}$ to $root$ assign timing values using the lower bound on delays
2. From $root$ to t_α assign timing values using the upper bound on delays
3. Use assignments to estimate the TSE of $s_{\alpha-\beta}$ and t_α

The algorithm in more detail is presented followingly.

ATSE(G_α , $s_{\alpha-\beta}$, t_α **)**

1: **for** u_j in reverse topological order of G_α **do**

2:

$$m(u_j) = \begin{cases} 0 & \text{if } u_j = s_{\alpha-\beta} \\ 0 & \text{if } u_j \text{ has no path to } s_{\alpha-\beta} \\ max\{d + m(v_k) \mid u_j \xrightarrow{[d,D]} v_k, v_i \rightsquigarrow s_{\alpha-\beta}\} & \\ & \text{if } u_j \text{ has path to } s_{\alpha-\beta} \end{cases}$$

3: **end for**

4: $M(root) \leftarrow 0$

5: **for** v_k in normal topological order of G_α **do**

6: **if** v_k has path to $s_{\alpha-\beta}$ **then**

7: $M(v_k) \leftarrow max\left\{min(0, M(u_j) + D - m(u_j) + m(v_k)) \mid u_j \xrightarrow{[d,D]} v_k\right\}$

8: **else**

9: $M(v_k) \leftarrow max\left\{M(u_j) + D - m(u_j) + m(v_k)) \mid u_j \xrightarrow{[d,D]} v_k\right\}$

10: **end if**

11: **end for**

12: **return** $M(t_\alpha) - m(t_\alpha)$

For example, consider the Process Graph of Figure 3.1 and the problem of finding the maximum separation Δ_3 of events a_2 and a_3 as source and target, respectively. m-values are calculated from source to root and M-values are computed from root to target. The unfolded graph with the m- and M-value annotation appears in Figure 3.3. The result is $\Delta = M(a_3) - m(a_2) = 25 - 0 = 25$.

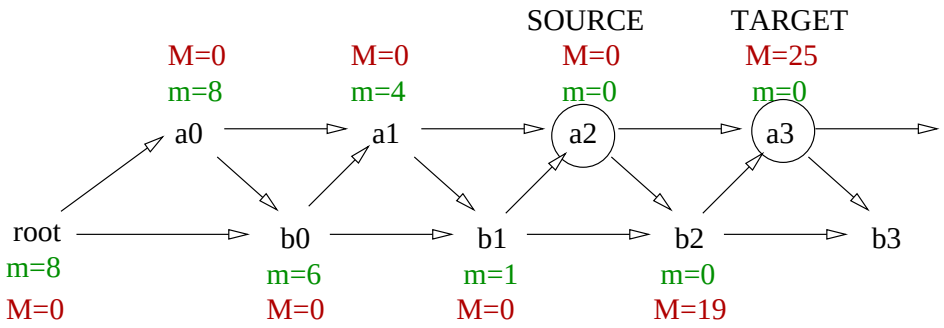

Figure 3.3: Unfolded graph of Figure 3.1 with m- and M-value annotation.

3.3 TSE Algorithm

The approach presented above estimates an upper bound on the timing separation of two specific occurrences of events $s_{\alpha-\beta}$ and t_α of a portion of an unfolded process graph G_α. However, the issue of estimating the TSE of two events of a Process Graph concerns the separation over all possible occurrences rather than two specific ones. The issue must be generalized to take into account infinite execution. The infinite possible occurrences are represented through sequential unfoldings of the initial process graph and the maximum TSE is defined as

$$\Delta = max\{\Delta_k : k \geq max(0, \beta)\} \tag{3.4}$$

which implies infinite number of applications of the previous algorithm. Instead special algebraic structures are used for the modeling and analysis of infinite execution.

3.3.1 Backwards Unfolding and Functions

The infinite different Δ_k analyses imply infinite different source and terminal events for each analysis and consequently different m- and M-values in each unfolding. To avoid recalculation in each unfolding, backward unfolding is used and timing functions are defined for every event.

Backwards unfolding follows the same concepts as normal unfolding but instead of considering root as a steady unique reference point, it considers the target event as the steady unique reference point. This only causes a change in the numbering of event occurrences while keeping the source and target events unique for every unfolding. Instead of numbering events starting from root event and proceeding to the source and terminal events ($s_{\alpha-\beta}$, t_α), events are numbered relative to target event t_α as the reference event. For every event v the relative occurrence index is $\alpha - k$ and is written $v_{(k)}$. The relative occurrence index for t_α is 0 and for $s_{\alpha-\beta}$ is β. For each unfolding k a different $root$ event is considered and is denoted as $root_k$.

Following this method, Figure 4.5 illustrates a portion of the backward unfolded process graph of Figure 3.1.

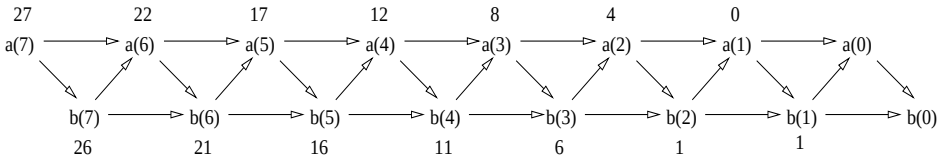

Figure 3.4: Portion of the backwards unfolded process graph for the process graph

Timing functions are defined as a set of pairs $\{\langle l_1, w_1\rangle, ..., \langle l_n, w_n\rangle\}$ and corresponds to the function:

$$f(x) = max\{min(x + l_i, w_i) \mid 1 \leq i \leq n\}$$

Moreover, specific operators are defined over the timing functions. Function composition ($\otimes$) relates the functions along a path of the graph and is defined as $(f \otimes g)(x) = g(f(x))$. Function maximization ($\oplus$) relates functions of two different paths merging in one and is defined as set union, $(f \oplus g)(x) = f \cup g$.

To model the TSE problem with functions, *edge functions* f_r are associated with each edge, based on delays of the rules and whether there is a path to source events or not.

$$f_r = \begin{cases} \{\langle l_r, 0\rangle\} & \text{if } u_k \text{ has path to } s_{\alpha-\beta} \\ \{\langle l_r, \infty\rangle\} & \text{if } u_k \text{ has no path to } s_{\alpha-\beta} \end{cases}$$

where $l_r = D - m(u_{k-\epsilon}) + m(v_k)$

Using the operators of maximization and composition *timing functions* F can be defined for each event, in relation to other events. The main characteristic/attribute of these functions is that when evaluated, they provide the M-value for the specific event, in relation to the other event recursively, through all the paths between them. For example, the M-value of event t_0 can be evaluated relatively to the M-value of $root_k$ with the function:

$$M(t_0) = F_{root_k \to t_0}(M(root)) = F_{root_k \to t_0}(0)$$

Moreover, $\Delta_k = M(t_0) - m(t_0) \Rightarrow \Delta_k = F_{root_k \to t_0}(0) - m(t_0)$. So, TSE can be also evaluated through m-values and timing functions. Combined with backward unfolding, m-values and relative functions are calculated once for each unfolding step and incrementally for each additional unfolding.

3.3.2 Bounding the Maximum Separation Time

A key observation of several examples is that by unfolding several times a process graph the maximum TSE Δ_k of two events will eventually reach a constant value or exhibit periodic behavior. However, when continuously unfolding, a means of determining convergence is required. Therefore, along with unfolding the graph, upper and lower bounds on Δ are computed for each iteration. When these bounds converge then the converging value is the maximum separation over all Δ_k values, as in Figure 3.5.

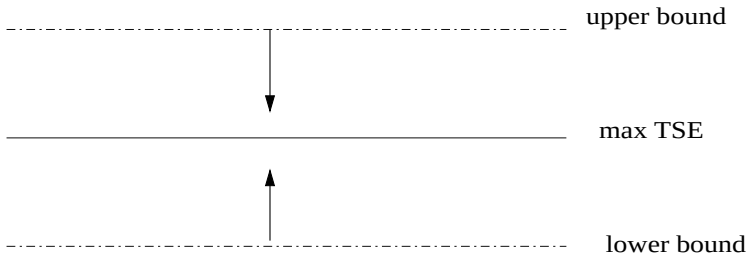

Figure 3.5: Bounding TSE

The pseudocode of an algorithm evaluating the TSE based on relative functions and bounds appears below.

TSE-Unfold(G,s,t,β,k_{max})

1: $k \leftarrow max(0, \beta)$
2: $\Delta^{\perp} \leftarrow -\infty$
3: $\Delta^{\top} \leftarrow +\infty$
4: **while** $\Delta^{\perp} < \Delta^{\perp} \wedge k < k_{max}$ **do**
5: Construct G_k
6: $\Delta_k \leftarrow F_{root_k \rightarrow t_0}(0) - m(t_0)$
7: $\Delta^{\perp} \leftarrow$ new lower bound value
8: $\Delta^{\top} \leftarrow$ new upper bound value
9: $k \leftarrow k + 1$
10: **end while**
11: **return** $(\Delta^{\perp}, \Delta^{\top})$

The lower bound is evaluated as the maximum of Δ_k until the current unfolding. The upper bound is evaluated using recursive functions and cutsets. A cutset n [6] is referred to as a set of relative event occurrences, such that for every further unfolding, G_j, $j \geq k$, every path from $root_j$ to t_0 goes through

an element of the cutset. For a given cutset X for G_k the function $F_{root_k \to t_0}$ is defined as:

$$\bigoplus \{F_{root_k \to u_j} \oplus F_{u_j \to t_0} | u_j \in X\}$$

The upper bound is computed as:

$$\Delta^{\top} = max\{F_{v_k \to t_0}(0) | v_k \in X\} - m(t_0)$$

The choice of cutset in not clarified enough. The definition and the alternative choices for cutset are examined in the next chapter.

3.3.3 Repetition Parameters

The algorithm presented above may exhibit some inefficiencies, in some process graphs. These are that the bounds may not converge or it may require an indefinite number of unfoldings to reach a convergence. Therefore a mathematical theory is developed to give more precise answers. The basis on which this theory works is that unfoldings are determined by a steady repetitive system, which eventually will locks in a repetition with specific period.

Repetition of m-values

As defined in [5] the ration of a cycle c is $\frac{d(c)}{\epsilon(c)}$. A maximum ratio cycle c is a cycle with maximum ratio $\frac{d(c)}{\epsilon(c)}$.

In a strongly-connected graph, all nodes have a path to a maximum ratio cycle, and the maximum ratio cycles have a path to a source node s. This guarantees that the m-values of all nodes eventually are determined repetitively by maximum ratio cycles and thus eventually repeat. For a strongly-connected process graph G' there exist integers k^* and ϵ^* such that:

$$m(v_{k+\epsilon^*}) - m(v_k) = r\epsilon^*, \forall k \geq k^* + \beta \tag{3.5}$$

k^* is the number of unfoldings (relative to source event) before the repetition occurs and ϵ^* is the period of this repetition. So, for every additional ϵ^* unfoldings, after the first $k^* + \epsilon^*$ unfoldings, the analysis of m-values will (be the same) as the previous ϵ^* unfoldings.

As it appears in Figure 4.5 the m-values repeat when:

$$m(v_{k+1}) - m(v_k) = r\epsilon* = 5.$$

This is true for $m(a_5) - m(a_4)$ and for increasing occurrences. So, after $k* = 3$ unfoldings relative to a_1 (source event), m-values repeat with period $\epsilon* = 1$.

3.3.4 Function Matrices

The form of function matrices is also widely used. A function matrix is a matrix such that its elements are timing functions as defined previously. For two cutsets X, Y a matrix $F_{X\to Y}$ is comprised of the timing functions from all vetrices of cutset X to those of cutset Y. Each element in position [i,j] of the matrix represents the timing function from the i-th element of source cutset X to the j-th element of the destination cutset Y, *i.e.* $F_{x_i\to y_j}$ for $x_i \in X$, $y_i \in Y$.

Addition of function matrices is defind as the conventional matrix addition but instead of scalar addition, function maximization is used. Multiplication of function matrices is defined as $(\oplus, \otimes)$ multiplication, *i.e.* traditional matrix multiplication using function maximization and composition instead of scalar addition and multiplication, respectively. By multiplying two function matrices, the relation functions are composed. For example, $F_{X\to Y} \times F_{Y\to Z}$ produces the functions from vertices of cutset X to those of cutset Z.

The matrices used for the analysis are three, T, R_i and S_i. Let X_0 be the cutset after which the repetitive behaviour of the system starts. T represents the relation functions from cutset X_0 to t_0, *i.e.* the target event. R_i represents the relation functions from $root_i$ to cutset X_i and S_i the functions from X_{i+1} to X_i. The matrix-multiplication of $R_i \times S_i \times S_{i-1} \times$ ... $\times S_0 \times T$ results in the function F_i relating event $root_i$ with target event t_0. Now the maximum F_i is needed for the evaluation of M-values.

Since m-values repeat for unfoldings $j \geq k^*$ with period ϵ^*, the difference in m-values between any two nodes of the unfolded graph is the same as the difference of the same nodes ϵ^* occurenc es back, from (3.2).

$$m(v_{k+\epsilon^*}) - m(u_{j+\epsilon^*}) = m(v_k) - m(u_j)$$

Consequently, the functions relating these two events will be the same as ϵ^* occurenc es back. So, the graph only needs to be unfolded an analyzed for $k^* + \epsilon^*$ unfoldings. What is need is the evaluation of T, $Ri \forall i \in [0, \epsilon^*)$ and $Si \forall i \in [0, \epsilon^*)$.

The maximum F_i is evaluated as

$$F_{max} = RS^*T \tag{3.6}$$

T doesn't change through unfolding and is evaluated as:

$$T = F_{X\to t_0} \tag{3.7}$$

R and S are evaluated as follows:

$$R = \oplus\{R_i S_{i-1} S_{i-2} ... S_0 | 0 \leq i < \epsilon*\} \tag{3.8}$$

$$S = S_{\epsilon^*} S_{\epsilon^*-1} ... S_0 \tag{3.9}$$

S^* is the matrix closure of S, which is a method to evaluate the convergence of an operation over an infinite number of operands.

In conclusion the final algorithm appears below.

TSE(G,s,t,β,k_{max})

1: compute k^*
2: compute ϵ^*
3: TSE-Unfold(G,s,t,β,$k^* + \beta$)
4: **if** $\Delta^{\perp} \geq \Delta^{\top}$ **then**
5: **return** $\Delta^{\perp}$
6: **end if**
7: compute T
8: **for** $i \leftarrow 0, 1, ..., \epsilon^*$ **do**
9: compute R_i
10: compute S_i
11: **end for**
12: compute S^*
13: $F \leftarrow RS^*T$
14: $\Delta_{max} \leftarrow F(0) - m(t_0)$
15: **return** $max(\Delta_{max}, \Delta^{\perp})$

3.4 Open issues

In this chapter the theory of the TSE algorithm of [5] was presented. This theoretical analysis requires knowledge of various mathematical tools and leaves a long way through the examination of every theoretical aspect to the implementation of the algorithm. Moreover, various issues require further clarification, concerning the application of this algorithm for the timing analysis of asynchronous circuits.

Specifically, the calculation of repetition parameters ϵ^* and k^* are is not clarified enough. Ways are proposed, but are not completely analyzed, in respect to application and effectiveness. Moreover, the concept of cutset is not accurately defined, nor is its identification. Its effect, also needs further examination. One more concept that is insufficiently defined and analyzed is the closure. The practical meaning of this mathematical tool is unclear. Some additional stydying needs to be done on the

application of the specific algorithm for the timing analysis of asynchronous circuits, as, it is targeted for concurrent systems, in general. Questions that need answering are for example, how appropriate is the model TSE algorithm examines, for the modelling of asynchronous circuits, or what modifications are needed. Finally, an issue that is not addressed in [5] is minimum separation analysis. A full analysis on this subject was developed in this thesis.

These issues are presented in detail in the following chapters as they were studied in this thesis. The gap is, also, covered between theory and implementation for timing analysis of circuits.

4

Analysis, Clarifications and Improvements on the TSE Algorithm

The previous chapter presented an overview of an algorithm for the timing separation of events that appears in [5]. However, [5] contains several unclear issues and several aspects need further examination. This work tried to clarify the concepts that needed clarification, explore the aspects that weren't examined and extend the TSE algorithm to form a complete timing analysis algorithm. The advanced issues, that were studied, are in terms of theoretical concepts for the complete understanding of the mathematical formulation of the algorithm, theoretical extensions on the algorithm and specialized application on asynchronous circuits. The clarification of theoretical concepts refer to ideas like cutsets, closure and evaluation of repetition parameters. These issues were insufficiently defined and analyzed in the previous work. Theoretical extensions were developed, such as minimum timing separation analysis. Moreover, all the specific aspects are studied to cover the gap between theoretical analysis and practical implementation of a timing analysis EDA tool for asynchronous circuits.

4.1 Specification Model

The specification model which is used to describe the concurrent systems in [6] is the Process Graph. Process Graph is an ER System, so, it can be used to specify asynchronous circuits. However, the most common way of specifying asynchronous circuits are Petri-nets or Marked Graphs. The following statements apply to Process Graphs modelling circuits and compare and contrast them to Process Graphs and Marked Graphs:

- PGs: $\forall v \in E' : \epsilon(v) \in [0, 1]$, *i.e.* ϵ can either be 0 or 1, but not greater. This holds for PGs that model circuits, as circuits cannot store state prior to one value.
- PGs: rule templates, *i.e.* edges labelled with $\epsilon = 0$ model dependence between events of the same index, whereas edges labelled with $\epsilon = 1$ model parallelism between events of the same index and dependenc es across indices.
- PGs/MGs: Edges with $\epsilon = 1$, along with the $root$ event, are equivalent to the initial token marking of MGs. Loops in MGs must contain at least one token [7] for the MG to be live. Similarly, for every cycle in the PG it should hold that $\epsilon \geq 1$, otherwise the PG would deadlock.
- PGs/STGs: events in PGs are not labelled as "+" or "-"; this is necessary for PGs that model circuits and lead to a similar representation as STGs.

4.2 Critical Cycles and Repetition Parameters

The open questions about the repetitive behavior of a Process Graph concern mainly when it will be reached, with what period it is repeated and how these parameters are evaluated.

As mentioned in the previous chapter, the m-value annotations of the unfolded graph represent the distance of a vertex from the source event s_β. They are computed backwards from it using the lower delay bound, d. Positive m-values of an event v imply that there is a path in the graph from event v to source event. Zero m-values imply either that event v is the source event or that there is no path from event v to source event, i.e the timing of v doesn't affect the timing of s_β.

The m-values are calculated only based on the lower delay bounds of the arcs reaching this event. By repeatedly unfolding the graph the m-values of the events are determined by a repetitive system. So, they will eventually reach an equilibrium state determined by maximum ratio cycles. As appears in Figure 4.1, considering as source the event a, the difference of m-values between successive occurrences of b are: $1, 5, 5, 5, ...$ and between successive occurrences of a are $4, 4, 5, 5, 5, ...$ This indicates that repetition with respect to m-values is reached after the 4-th unfolding.

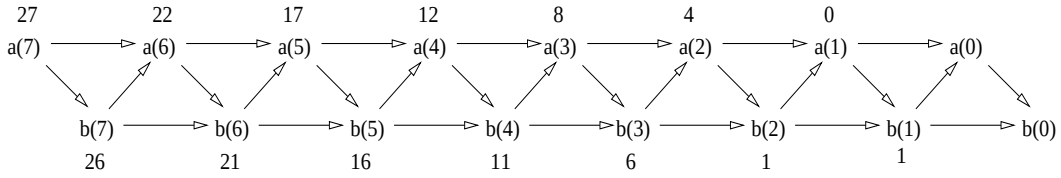

Figure 4.1: Portion of the backwards unfolded graph for the Process Graph of Figure 3.1 labeled with m-values for $s_{(\beta)}$=$a_{(1)}$

The evaluation of the period and the initial transient behavior are analyzed followingly.

4.2.1 Ratio'ed Cycles and Maximum Ratio Cycles

A ratio cycle is defined as a simple cycle where the sum of the d values is ratio'ed by the ϵ values and the ratio of the cycle is:

$$r(c) = \frac{d(c)}{\epsilon(c)} \mid \text{where } c \text{ is simple cycle in } G' \tag{4.1}$$

A maximum ratio cycle is the ratio'ed cycle with the maximum ratio r, which is:

$$r_{max} = max(\frac{d(c)}{\epsilon(c)}) \mid \forall c \text{ where } c\text{'s are simple cycles in } G' \tag{4.2}$$

In a strongly connected graph, the maximum ratio r is unique, which means that the maximum ratio cycles have the same ratio, *i.e.* the maximum ratio, which is the one determining the timing of the circuit.

The equation (3.4) for the repetition of the m-values can also be interpreted as follows to clarify the concept of maximum ratio cycles:

$m(v_{k+\epsilon^*}) - m(v_k) = r\epsilon^*, \forall k \geq k^* + \beta \Rightarrow m(v_{k+\epsilon^*}) - m(v_k) = d(c)$, where c is the maximum ratio cycle.

The fact that the m-values of all events are determined by maximum ratio cycles points that a change in the delays on all these cycles, reducing the maximum ratio, would result in the decrease in the separation of events analysis. As far as actual circuits are concerned, a maximum ratio cycle in the specification graph would (yields) a path or a tree in the circuit that is critical to the timing of the circuit. In accordance to critical paths in synchronous circuits, critical cycles in the specification graph of asynchronous circuits are the ones determining the cycle period of all signal transitions. This provides a specific target for optimization procedure.

4.2.2 Calculation of ϵ^*

As mentioned in [6], by using digraph algorithms [8] it is possible to extract all the cycles in the process graph G' and then find which of them are maximum ratio cycles. The calculation of ϵ^* is complicated by the potential existence of multiple maximum ratio cycles, *as the m-values for different events may use different maximum ratio cycles.* In the trivial case of one unique maximum ratio cycle c, ϵ^*=$\epsilon(c)$.

If more ration cycles exist in the process graph a different analysis is required. G^*, is created as the subgraph of G' containing only the edges of the maximum ratio cycles and G_i^* is denoted as the ith strongly connected component of G^*. If an event, v, can get its m-value from two different maximum ratio cycles, *e.g.* $c1$ and $c2$ and these are in different strongly connected components of G^*, then the occurrence period of v, $\epsilon^*(v)$ is calculated as the Lowest Common Multiple of $c1$ and $c2$:

$$\epsilon^*(v) = LCM(\epsilon(c1), \epsilon(c2)) \tag{4.3}$$

If $c1$ and $c2$ are part of the same strongly connected component of G^*, the occurrence period is calculated as the Greater Common Divisor of $c1$ and $c2$:

$$\epsilon^*(v) = GCD(\epsilon(c1), \epsilon(c2)) \tag{4.4}$$

By enumerating all cycles we get:

$$\epsilon^* = LCM(GCD(\epsilon(c) \mid c \in G_i^*)) \tag{4.5}$$

For the clarification of the meaning of ϵ^* examples aiding intuition are provided. The event graph G^* shown in Figure 4.2 consists of two different strongly connected components. The dotted lines are part of the initial process graph G' but not of process graph G^*, *i.e.* they are not part of any maximum ratio cycle. There are two independent maximum ratio cycles $c1$, $c2$ consisting the graph G^*, such that $r(c1) = \frac{d(c1)}{\epsilon(c1)} = \frac{8}{4} = 2$ and $r(c2) = \frac{d(c2)}{\epsilon(c2)} = \frac{6}{3} = 2$. In this case, the execution of event a in the initial process graph is constrained by events of two different maximum ratio cycles (c, d) but its m-value is eventually determined (after k^* unfoldings) by cycle c2, since the edge $d \rightarrow a$ is not part of a maximum ratio cycle. The events of cycles c1, c2 will occur independently and will be synchronized every $LCM(\epsilon(c1), \epsilon(c2))$. So, the occurrence period is $\epsilon^*(v) = LCM(\epsilon(c1), \epsilon(c2))$. In the example graph of Figure 4.2, $\epsilon^*(v) = LCM(\epsilon(c1), \epsilon(c2)) = LCM(4, 3) = 12$.

The graph G^* shown in Figure 4.3 consists of two maximum ratio cycles that have a common event, *i.e.* a. In this case, the m-values of a are determined by both cycles c1, c2. Since there is a

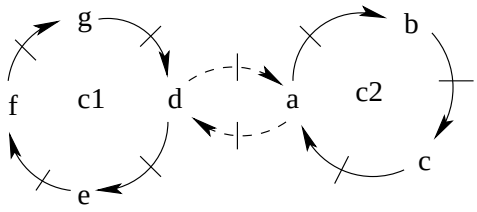

Figure 4.2: Process graph G^* of maximum ratio cycles on two different strongly connected componets. The edges in dotted lines are the edges of the initial process graph G' that are not part of a maximum ratio cycle.

synchronization point at which the events of the graph will be synchronized every $GCD(\epsilon(c1), \epsilon(c2))$, the occurrence period is $\epsilon^*(v) = GCD(\epsilon(c1), \epsilon(c2))$. In the example graph of Figure 4.3, if $r(c1) = \frac{d(c1)}{\epsilon(c1)} = \frac{8}{4} = 2$ and $r(c2) = \frac{d(c2)}{\epsilon(c2)} = \frac{2}{4} = 2$ then $\epsilon^*(v) = GCD(\epsilon(c1), \epsilon(c2)) = GCD(4, 2) = 2$.

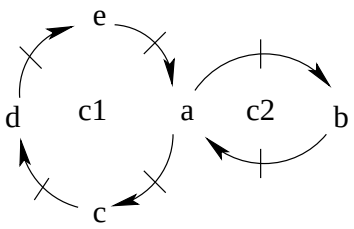

Figure 4.3: Process graph G^* of maximum ratio cycles with two cycles having a common event.

4.2.3 Calculation of k^*

Based on the above discussion it is possible to calculate ϵ^*. The calculation of k^* is based on whether m-values for an unfolding are calculated from maximum ratio cycles. If no, then further unfoldings are needed. If yes, the value of ϵ^* is critical. If ϵ^* is 1, then it is certain that m-values of subsequent unfoldings are being repeatedly calculated depending on the maximum ratio cycle. However, for larger values of ϵ^* and particularly for multiple m-value calculations and multiple maximum ratio cycles, k^* can be determined as a solution to the Frobenius problem as:

$$k^*(v) = k_0 + Frobenius\epsilon(c) \mid c \in G_i^* \tag{4.6}$$

The Frobenius Problem and Number

The Frobenius number is the largest value b for which the Frobenius equation: $a_1x_1 + a_2x_2 + ... + a_nx_n = b$, where a_i are positive integers and the solutions x_i are nonnegative integers, has no solution.

The Frobenius number is intimately related to the so called Coin Problem, whereby the largest sum of money which cannot be formed from a quantity of coins of different value is sought.

However, for more than two variables, *i.e.* cycles, the Frobenius problem has no exact solution. Thus, a solution to the Frobenius problem wasn't a focus of this work. Instead, k^* is evaluated with unfoldings. Since the values of ϵ^* and maximum ratio r_{max} are known, the graph is initially unfolded ϵ^* times. For every further unfolding i, the equation

$$m(v_i) - m(v_{i-\epsilon^*}) = r\epsilon^*$$

is checked. If it is true then $k^* = i$. If it is false, k^* unfolding isn't reached and further unfoldings are needed. This is a way of accurately evaluating k^* without increasing the complexity of the algorithm.

4.2.4 Examples

The calculation of the ϵ^* and k^* variables and the repetition of m-values as it was examined through this work are further clarified in the folowing examples.

In Figure 3.1 we can see a process graph with four simple cycles. Cycle $c1 = \{a \rightarrow a\}$, cycle $c2 = \{a \rightarrow b \rightarrow a\}$, cycle $c3 = \{b \rightarrow a \rightarrow b\}$ and cycle $c4 = \{b \rightarrow b\}$. The ratio of each cycle is $r(c1) = \frac{d(c1)}{\epsilon(c1)} = 4$, $r(c2) = \frac{d(c2)}{\epsilon(c2)} = 2$, $r(c3) = \frac{d(c3)}{\epsilon(c3)} = 2$ and $r(c4) = \frac{d(c4)}{\epsilon(c4)} = 5$. So, $r = 5$ is the maximum ratio and cycle c4 is the maximum ratio cycle. In this example there is only one maximum ratio cycle, so, $\epsilon^* = \epsilon(c4) = 1$. The process graph of Figure 3.1 is unfolded and the m-values are computed. Since the m-values are computed in reverse topological order starting from the source event, backward unfolding method is followed in the representation of the unfolded graph.

Following this method, Figure 4.1 illustrates a portion of the unfolded process graph of Figure 3.1 and the behaviour of the m-values. The m-values repeat when

$$m(v_{k+1}) - m(v_k) = r\epsilon^* = 5. \tag{4.7}$$

This is true for $m(a_5) - m(a_4)$ and for increasing occurrences. So, after $k^* = 3$ unfoldings relative to a_1 (source event), m-values repeat with period $\epsilon^* = 1$.

Figure 4.4 shows a second example of a process graph. One cycle appears in this process graph with $r(c) = \frac{d(c)}{\epsilon(c)} = 7.5$. This is the maximum ratio cycle, and $\epsilon^* = \epsilon(c) = 2$. The repetition of m-values appears in the unfolded process graph of Figure 4.5. After $k^* = 2$ unfoldings relative to a_1, m-values repeat with period $\epsilon^* = 2$ and

$$m(v_{k+2}) - m(v_k) = r\epsilon^* = 15.$$

This is apparent as each event occurs depending only on the other event occurrence of one previous index and on two previous index occurrences of the same event.

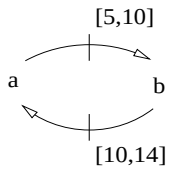

Figure 4.4: Process graph G_2

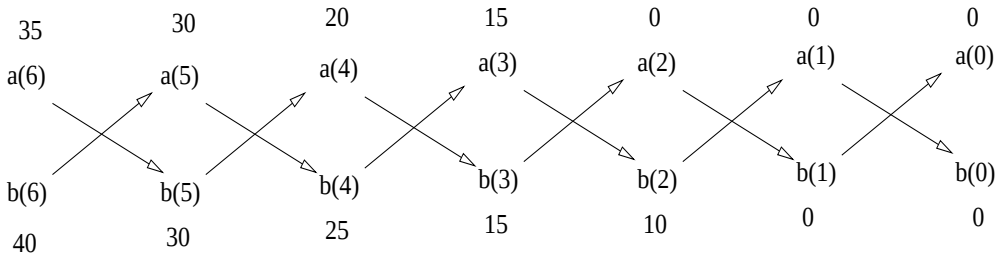

Figure 4.5: Unfolded process graph labeled with m-values for $s_{(\beta)}$=$a_{(1)}$

If a different event is chosen as the source vertex, the analysis would be the same as the maximum ratio cycle won't change. The variable ϵ^* is a function of the maximum ratio cycle and the minimum delays of the edges. These parameters don't change among different pairs of source-terminal vertices, so ϵ^* will be the same. However, the variable k^* can change. The unfolded process graph appears again in Figure 4.6 for $s_{(\beta)}$=$b_{(1)}$. ϵ^* is still 2 and after $k^* = 1$ unfoldings m-values repeat with

$$m(v_{k+2}) - m(v_k) = r\epsilon* = 15. \tag{4.8}$$

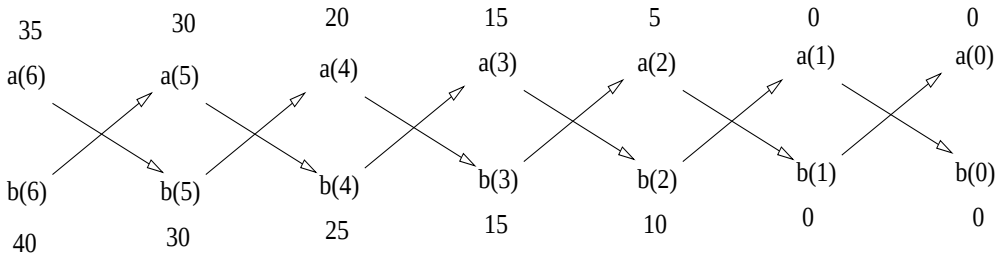

Figure 4.6: Unfolded process graph labeled with m-values for $s_{(\beta)}$=$b_{(1)}$

4.3 Cutsets

The notion of cutset was not fully defined in the previous work and is clarified in this thesis. The identifying of the cutset was, also, not specified. Another question about cutsets is on the effect of

different cutsets on the algorithm.

In graph theory, for a strongly connected graph $G = \langle V, E \rangle$, S is a k-vertex cutset if $S \subseteq V$, $|S| = k$ and $G - S$ is not connected [9]. In our analysis, we are interested in the connectivity of root with the target event t_0. So, we consider cutset S, as the k-vertex cutset, such that $S \subseteq V$, $|S| = k$ and $G - S$ has no path from root to target event.

Various set selections satisfy the cutset definition. In various unfolded graphs G_i, the set of the current unfolding events, v_i, $\forall v \in V$, is a cutset of the graph. However, the choice of cutset can affect the complexity of the algorithm, which is a function of the size of the selected cutset. Thus, the whole unfolding is not an efficient choice of cutset. The best choice would be the minimum cutset, *i.e.* the cutset with minimum size, but it refers to a graph theory problem, that introduces a high complexity on the algorithm.

Instead of identifying a minimum cutset, a minimal cutset can be efficient, leading to correct solutions of the problem.A minimal cutset, that can be easily deduced is the set of events $\{v | v \rightarrow t_0 \in R\}$. This is not a minimum of cutset, but it is a minimal cutset with insignificant complexity. This choice was adopted in this work.

4.4 Closure

In mathematics, a set is said to be closed under some operation if the operation on members of the set produces a member of the set. Given an operation on a closed set X, one can define the closure $C(S)$ of a subset S in X to be the smallest subset closed under that operation that contains S as a subset. The closure of sets, with respect to some operation, defines a closure operator on the subsets of X. The closed sets can be determined from the closure operator; a set is closed if it is equal to its own closure [9].

4.4.1 Closure in Closed Semirings

A closed semiring is a system $(S, +, ., 0, 1)$ where S is a set of elements and $+$ and . are binary operators on S satisfying the following properties:

1. $(S, +, 0)$ and $(S, ., 1)$ are monoids, *i.e.* closed under the operator, are associative, *i.e.* $a + (b + c) = (a + b) + c$ and have the identity property for 0, *i.e.* $a + 0 = a + 0 = a$, and 1 respectively. 0 is also an annihilator for $\cdot$

2. $+$ must be commutative, *i.e.* $a + b = b + a$, and idempotent, *i.e.* $a + a = a$

3. $\cdot$ must distribute over $+$, *i.e.* $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$

4. Finite and infinite sums, using $+$, must exist and be unique. Associativity, commutativity and idempotence must apply to infinite and finite sums

5. $\cdot$ must distribute over countably infinite sums as well as finite ones

Thus, based on the above, the following equation holds:

$$(\sum_i a_i) \cdot (\sum_j b_j) = \sum_i a_i \cdot b_j = \sum_i (\sum_j (a_i \cdot b_j)) \tag{4.9}$$

For example, consider $S_2 = (R, MIN, +, +\infty, 0)$, where R is the set of non-negative real numbers including $+\infty$. It is easy to verify that $+\infty$ is the identity under MIN, whereas 0 is the identity under $+$:

$$MIN(a, +\infty) = MIN(+\infty, a) = a, a + 0 = 0 + a = a$$

The closure operation $a* = \sum_{i=0}^{\infty}$ yields:

$$0* = MIN(0) = 0$$

as the inner operator of the closure formula is $+$ (the outer is MIN) and addition zero times, yields 0. Based on $0*$ the closure of any $a*$ can be calculated as follows; for any $a \in S_2$:

$$a* = MIN(0, a, a + a, a + a + a, ...) = 0$$

Thus $\forall a \in R, a* = 0$.

The practical meaning of closure is that an operation on an infinite sequence of elements (of a closed set) results in a well-defined finite element that, also, belongs to the same set. So, it is a mathematical tool to model and evaluate infinite operations. In the presented analysis, the set of functions is a closed semiring, under the operations of maximization ($\oplus$) and composition ($\otimes$), with identity elements, $\overline{0} =$ and $\overline{1} = \langle 0, +\infty \rangle$, respectively. So, a closure can be defined for functions set. Consequently, the set of matrices is a closed semiring, as it is comprised by functions, on which maximization and composition are performed.

4.5 Relation Matrices

Relation matrices are matrices that are comprised of relation functions and represent all the paths from one cutset to another. When the starting cutset contains the $root$ event and the ending cutset the $target$ event, the relation matrix is a 1-dimensional array with function $F_{root \to t_0}$. This abstract definition may

lead to inaccuracies in the construction of matrices from functions. One point of clarification is that the function-elements of a matrix $M_{X \to Y}$ should not include paths from a vertex $x \in X$ with vertices of the same cutset as intermediate nodes. This needs to hold for one of the cutsets, in order not to include multiple appearances of the same path. As a convention it is kept for the starting cutset.

4.6 Minimum Separation Analysis

The problem definition of Minimum Separation Analysis is to estimate the minimum time separation, *i.e.* δ, between two events, *i.e.* s and t, with a separation in occurrence, *i.e.* β, over all possible occurrences, as it appears in the following equation.

$$\delta \leq \tau(t_k) - \tau(s_{k-\beta}) \tag{4.10}$$

The problem theoretically can be addressed using maximum separation analysis as a mathematical transformation of the equation (3.1) as:

$$\tau(s_k) - \tau(t_{k-(-\beta)}) \leq -\delta \tag{4.11}$$

This is true in theory, but could not be applied in practice. It cannot be done in practice using the same algorithm just by changing the signs. This would imply negative unfoldings and wouldn't respect the relative ordering of source and target events. Minimum analysis doesn't follow the same semantics of the maximum analysis based on m- and M-values as described previously resulting in non-deterministic behavior or inaccurate estimations.

A different analysis was developed for the estimation of the minimum time separation between events, during this work. The minimum analysis following the same baselines as the maximum timing separation analysis. The key idea is to force the source event to occur on its latest possible time and based on this timing assignment to force the target event to occur on its earliest possible time. In order to understand the changes in minimum analysis a closer look should be taken at m- and M-values.

4.6.1 m-values and M-values

m- and M-values are used in the maximum analysis to model the minimum and the maximum offset delay in relation to source event and root event. Specifically, the m- values represent the relative timing separation between the event in question and the source event, for minimum delays. The M-values represent the collective relative delay of events starting from $root$ event, considering minimum time occurrences for events that have a path to source event and maximum time occurrences for events that don't have a path to source event. This appears better in Figure 4.7.

The line represents the time axis with source events as the reference point. The events are placed on the axis on the time of their occurrence. m-value annotation places the events on their minimum

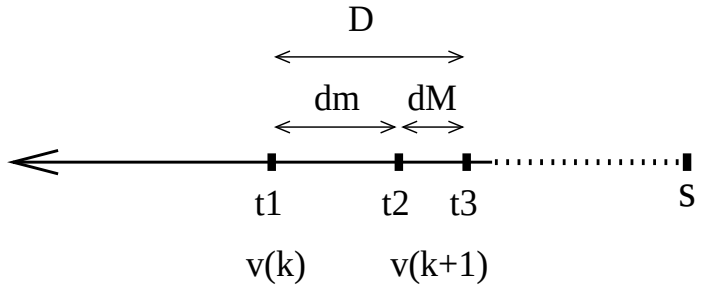

Figure 4.7: Time axis for maximum separation analysis.

time of occurrence in reference to source event. Points $t1$, $t2$ on the axis are the m-values of events v_k and v_{k+1}, respectively. The M- values are computed based on m-values in normal topological order on the graph. Consider a rule $r : v_k \rightarrow v_{k+1}$, where v_k and v_{k+1} can be either different occurrences of the same event or different events. dm on Figure 4.7 represents the offset minimum delay of the two events, *i.e.* $dm = m(v_k) - m(v_{k+1})$ and $dM = D - dm$. The practical meaning is that if event v_k occurs on its minimum possible time, event v_{k+1} may occur in the time range dM if there is no other constraint. M-value of v_{k+1} is calculated as $M(v_{k+1}) = M(v_k) + D - m(v_k) + m(v_{k+1}) = M(v_k) + dM$. If there is a path to source event, this value is minimized with 0. A positive value would imply that event v_{k+1} would occur with a time delay greater than its minimum possible. This is not consistent with the definition of the analysis which forces the the events leading to source event to occur on their earliest time. The maximization is to include all the constraints to an event.

The calculation of m- and M-values in minimum analysis should follow the same principle and be consistent with the definition of the analysis. The m-values are estimated with the same equation but using the upper delay bounds D through the following equation:

$$m(v_k) = \begin{cases} 0 & \text{if} v_k = s_{k-\beta} \\ 0 & \text{if } v_k \text{ has no path to } s_{k-\beta} \\ max\{D + m(v_i) \mid v_k \rightarrow v_i, v_i \text{ has path to } s_{k-\beta}\} & \text{if } v_k \text{ has path to } s_{\alpha-\beta} \end{cases}$$

The equation of the M-values should be the same with before but using the lower bounds. However, a minimization with 0 wouldn't agree with the analysis. The case now is that M-values smaller than $-m(v)$ should be increased to $-m(v)$. This is explained by looking at the time axis again. Now consider $t1$ and $t3$ as the m-values of events v_k and v_{k+1}, respectively, which now represent the maximum delays relative to the source event. Again $dm = m(v_k) - m(v_{k+1})$ and $dM = d - dm$. As it appears in Figure 4.8, $|dM + M(v_k)| < m(v_k)$ should always hold. Otherwise, event v_{k+1} would

occur after event v_k, which doesn't agree with the specification model, since $v_k \rightarrow v_{k+1} \in R$.

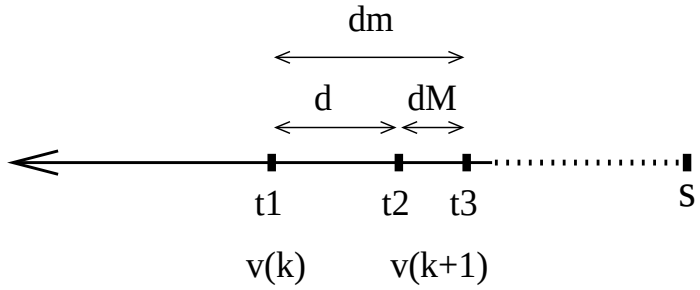

Figure 4.8: Time axis for minimum separation analysis.

So, the M-values of any event v_k estimated through the following equation:

$$M(v_k) = \begin{cases} max\{max(-m(v_k), M(v_i) + d_{i\rightarrow k} + m(v_k) - m(v_i)) \mid v_i \rightarrow v_k\} & \text{if } v_k \text{ has path to } s_{k-\beta} \\ max\{M(v_i) + d_{i\rightarrow k} + m(v_k) - m(v_i) \mid v_i \rightarrow v_k\} & \text{if } v_k \text{ has no path to } s_{\alpha-\beta} \end{cases}$$

An example that shows the evaluation of m- and M-values on a graph for minimum and maximun separation analysis appears in Figure 4.9.

Minimum and maximum separation $\Delta_2 = b_2 - a_2$ is estimated. Greater value is observed on minimum than maximum separation because separation for only two unfoldings is considered. More accurate and tight bounds will be estimated using the TSE version for infinite unfoldings. One more thing to be noted is that with single delay values instead of ranges, m-values as well as M-values of minimum and maximum separation analysis are evaluated the same, as it expected, in order to lead to same minimum and maximum separation.

4.6.2 Cycles and Repetition Parameters

The execution of an event is also determined by the execution of its latest predecessor, *i.e.* for each event to occur, it must wait for all the events that have edges to this event to occur. So, timing assignments, consequently m- and M-values, are also determined by the maximum ratio cycles. The maximum ratio cycles are enumerated based on the initial process graph, so they are the same for minimum and maximum analysis.

As for the the maximum analysis, the m-values are determined repetitively by maximum ratio cycles and will eventually repeat. The period of the repetition, ϵ^*, is also unchanged for the same initial graph, thus, the same for minimum and maximum analysis.

The number of unfoldings until the repetition occurs, k^*, may be different and it is evaluated within the unfoldings.

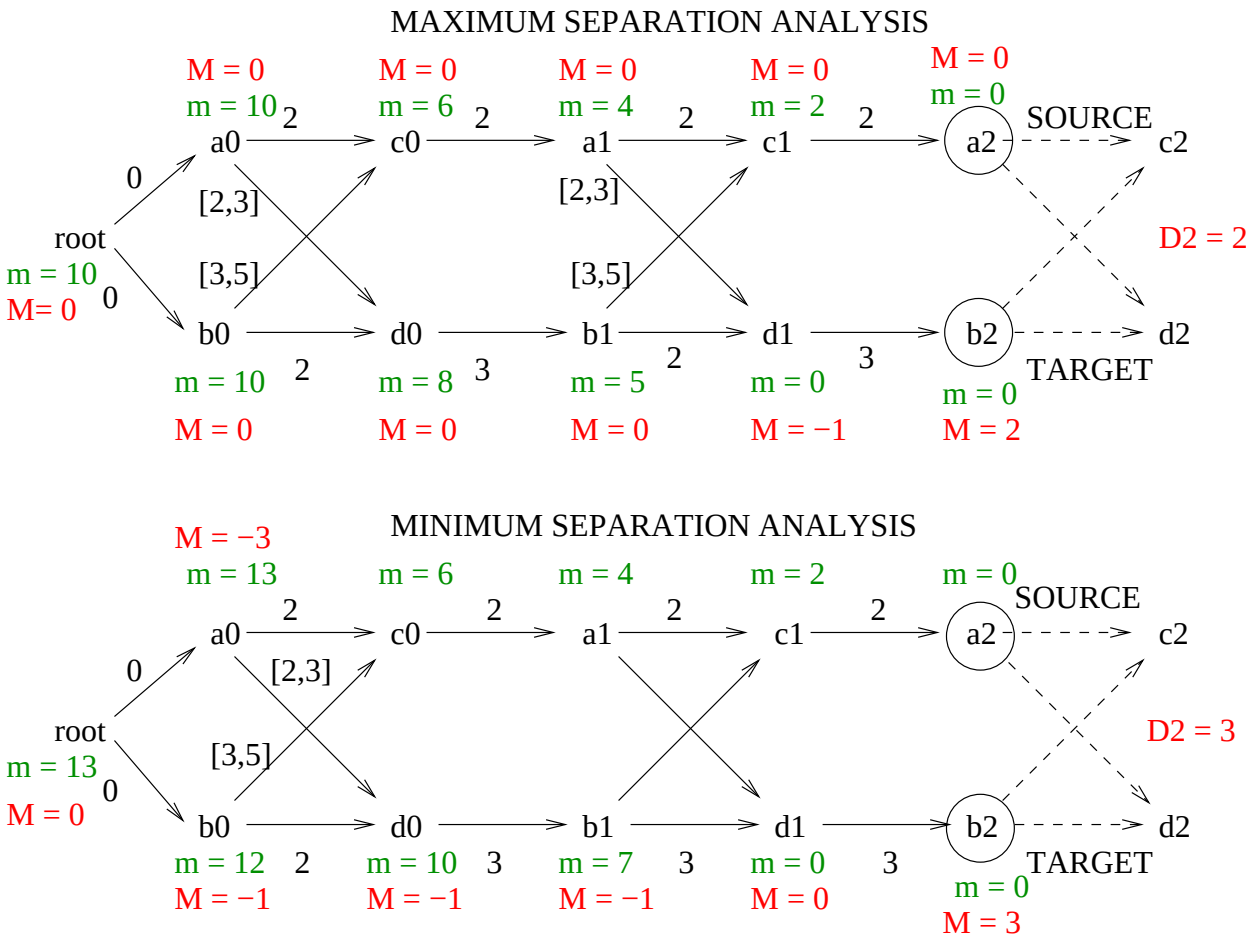

Figure 4.9: Evaluation of m- and M-values in maximum and minimum separation analysis.

4.6.3 Functions

Functions are used in maximum time separation analysis to evaluate the relative time separation between events recursively. In minimum analysis, the same structures are used. Backward unfolding is also used and the functions are estimated in relation to target event in order to provide the M-value when evaluated. Because the M-values have different meaning in minimum analysis, a different evaluation function is needed to provide the minimum delay offset. The new evaluation function is defined as

$$for\ f = \langle l_i, w_i \rangle$$

$$f(x) = max\{max(x + l_i, w_i) | 1 \leq i \leq n\} \tag{4.12}$$

A different composition function $(\otimes)$ is also needed to provide for the minimum analysis meaning.

It is defined as

$$for\ f = \langle l_1, w_1 \rangle\ and\ g = \langle l_2, w_2 \rangle$$
$$(f \otimes g)(x) = \{\langle l_1 + l_2, max(w_1 + l_2, w_2) \rangle\} \tag{4.13}$$

The maximization function $(\oplus)$ is the same. The optimizations implemented in maximum analysis are also applicable in minimum analysis.

One more thing that is needed is a change in the direction in the conditional comparisons, *i.e.* $<\Leftrightarrow>$ in every processing/operation over the relation functions and matrices as well.

The minimum time separation that is estimated, represents the minimum time separation after the execution reaches its repetitive behavior. This assumption ignores the time delays observed in the initial unfoldings of the graph. These might provide for a smaller time separation between events. However, these delays do not appear during the execution and are only part of the transitional behavior of the circuit on reset.

4.7 Implementation and Complexity

Based on the TSE analysis, an Asynchronous Timing Analysis (ATA) tool was implemented for the timing analysis of asynchronous circuits, based on TSE analysis. The ATA tool enumerates about 3500 lines of C code for its implementation. It requires a specification of the circuit in Process Graph format and implements all the described functionality. It evaluates the maximum time separation between two events, as well as the minimum time separation. Moreover, it was extended to support floating point arithmetic, in addition to integer arithmetic.

The ATA tool was developed as two versions, following the theoretical analysis. The first version, **acyc-tse**, supports the evaluation of maximum and minimum timing separation Δ_α of two *event occurrences* of an unfolded process graph G_α. Invocations of this version will provide an estimation of TSE of specific unfoldings. The second version is the complete **tse** application. It designates the critical cycles of the process graph, *i.e.* the maximum ratio cycles, evaluates the repetition parameters and provides exact bounds maximum and minimum on the timing separation of two events over all possible executions of the system.

4.7.1 Acyclic TSE Version

The Acyclic TSE version requires the process graph G' as a command line argument. Subsequently, prompts for the source and target events of the process graph, the occurrence index separation and the required number of unfoldings for the estimation of TSE to be applied. The operation is comprised of two main steps. The first is to unfold the input process graph G' to an acyclic graph G_α and the second is to evaluate the maximum TSE between two specific event occurrences of the G_α. The steps

described above are repeated until the desired number of unfoldings is reached and the TSE is reported for each unfolding.

The unfolding operation is implemented as forward unfolding of the process graph by incrementally adding the elements of each new unfolding. The evaluation of Δ_α in each unfolding is based on the **ATSE** algorithm presented in Chapter 3. The topological order of the graph is acquired through DFS iteration of the graph.

Considering a Process Graph of n vertices and m edges, the complexity of **ATSE** is determined by the DFS invocations and is $O(n+m)$. For each unfolding the complexity is, also, $O(n+m)$. Thus, for unfolding the graph k times the complexity is $O(k(n+m))$.

4.7.2 Complete TSE Version

The complete TSE version also requires the process graph G' as a command line argument. Subsequently, prompts for the source and target events of the process graph and the occurrence index separation. This version operates in three main phases, as it appears in the pseudocode below.

TSE(G,s,t,β)

1: extract cycles from G
2: $G^* \leftarrow$ evaluate maximum cycles
3: compute ϵ^*
4: **while** $\Delta^\perp < \Delta^\top$ or k^* not reached **do**
5: $k \leftarrow$ TSE-Unfold(G',s,t,β)
6: $(\Delta^\perp, \Delta^\top) \leftarrow$ processUnfolding
7: **end while**
8: **if** $\Delta^\perp \geq \Delta^\top$ **then**
9: **return** $\Delta^\perp$
10: **end if**
11: create cutset C
12: create T
13: create R
14: create S
15: compute S^*
16: $F \leftarrow RS^*T$
17: $\Delta_{max} \leftarrow F(0) - m(t_0)$
18: **return** $max(\Delta_{max}, \Delta^\perp)$

The critical cycles are evaluated first, as well as the repetition parameter ϵ^*. This processing is over the Process Graph, not on the unfolded acyclic graph, so it is independent of the unfoldings. On the next phase, the graph is unfolded, evaluating m-values, timing functions and bounds for each unfolding. The unfoldings continue until the bounds converge or $k^* + \epsilon^*$ unfoldings are reached. If bounds converge, the converging value is reported as the solution to the evaluation. If they don't, *i.e.* repetitive state is reached the algebraic processing is applied. The function matrices are evaluated and closure is applied in order to reach a mathematically evaluated result.

Unfolding

The unfolding operation is implemented as backwards unfolding of the Process Graph by constructing an acyclic graph and incrementally adding the elements of the new unfoldings. The operation is applied in the same way as forward unfolding but with a difference in the relative numbering, as explained in the previous chapter. In each unfolding step, the m-values only of the additional events are evaluated, as well as the edge and timing functions for the additional elements. Considering a Process Graph with size of vertex set $|V| = n$ and size of edge set $|E| = m$, the unfolding process requires time $O(n + m)$.

Similarly, a Δ_k is evaluated in each unfolding, based on m-values and functions estimated during the unfolding. DFS iteration is also used to find the topological order of the graph and the processing afterwards is applied on every node of the unfolding. DFS is accomplished in time $O(n + m)$, *i.e.* it is only applied on events of the new unfolding. The processing on each events is completed in linear time. So, the complexity of unfolding is $O(n + m)$, for a single unfolding step.

Timing Functions

The functions for every edge are represented as a struct of two integer fields (l, w). The ∞ is represented with the macro definition MAXINT, which is the maximum integer that fits in the integer type. A data structure library was implemented that supports all the necessary functionality for the functions. Maximization ($\oplus$), composition ($\otimes$) and evaluation are supported.

The timing functions for each event are represented as a list of function pairs, as defined previously. An efficiency optimization, that is proposed in [5] and is implemented in the ATA tool of this thesis, is based on the observation that if $l_i \geq l_j$ and $w_i \geq w_j$ for two edge functions $f_i = \langle l_i, w_i \rangle$ and $f_j = \langle l_j, w_j \rangle$ then f_i subsumes f_j. This is a direct consequence of the definition of the function, *i.e.* $f(x) = max\{min(x + l_i, w_i) | 1 \leq i \leq n\}$, since for all x, $min(x + l_i, w_i) \leq min(x + l_j, w_j)$. So, every function can be represented as an ordered list, which holds the following order:

$$l_1 \leq l_2 \leq ... \leq l_n \; and \; w_1 \geq w_2 \geq ... \geq w_n. \tag{4.14}$$

The function elements of the list that are subsumed by others are redundant and ignored.

By preserving the previous order the operations of maximization and composition are performed efficiently in linear time. Maximization of two functions f, g is a set union and is completed in $O(n_f + n_g)$ time, where $|f| = n_f$ and $|g| = n_g$. To keep the order for the above optimization, the function elements that are subsumed by others are removed and the new elements are added in the appropriate position in list. Composition is, also, performed in linear time as $|f \otimes g| < |f| + |g|$, which is proved by induction.

Cutsets

The construction of cutset is constructed, as proposed in this thesis, as the set $C = \{v | v \rightarrow t_0 \in R\}$, where T_0 is the target event of the analysis. The complexity of this operation is $O(deg(t_0))$.

Relation Matrices

The relation matrices are implemented as one- or two-dimensional arrays. A data structure library was, also, implemented offering matrix addition, multiplication and closure functionality considering all points mentioned in the above sections.

Let X_0 be the cutset after which the repetitive behaviour of the system starts. The construction of matrices T, S and R can be done after unfolding the graph $k*+\beta+\epsilon*$ times. The construction of T is trivial as it consists of timing functions $F_{x_i \rightarrow t_0}$, already calculated as part of the unfolding step. Matrix S is constructed as $S = S_{\epsilon^*} S_{\epsilon^*-1} ... S_0$. Each S_i consists of functions $F_{x_{i+1} \rightarrow x_i}$ where $x_{i+1} \in X_{i+1}$ and $x_i \in X_i$ (X_{i+1} and X_i cutsets as defined previously). Functions $F_{x_{i+1} \rightarrow x_i}$ can be calculated using a DFS algorithm to find all paths from each x_{i+1} to each x_i. Accordingly, R is constructed as $R = \oplus \{R_i S_{i-1} S_{i-2} ... S_0 | 0 \leq i < \epsilon^*\}$. Each element R_i represents functions $F_{root_i \rightarrow x_i}$ which can, also, be calculated using a DFS algorithm to find all paths from event $root_i$ to each x_i.

The matrices S and R for the ϵ^* unfoldings can be constructed recursively as shown in the following pseudocode.

The cost in this case is the cost to calculate R_i and S_i for ϵ^* times, two matrix multiplications and a matrix addition for $\epsilon^* - 1$ times. Let c be the size of the chosen cutset, n the number of vertices and m the number of edges of one unfolding. Based on this, the cost to calculate S_i is $c^2 2(n + m)$ (c^2 DFS pairs of source-destination vertices, each DFS considers vertices and edges of two consequent unfoldings,thus $2(n+m)$). Respectively, the cost of R_i is $c(n+m)$ (c DFS pairs, for each DFS vertices and edges of one unfolding). Moreover, $S_i \cdot S$ requires c^3 operations, the maximization $(R_i \cdot S) \oplus R$

$S = S_0$
$R = R_0$
for $i = 1$ to $\epsilon^* - 1$ **do**
  $R = R_i \cdot S \oplus R$
  $S = S_i \cdot S$
end for

requires $2c$ operations and $R_i \cdot S$ requires c^2 operations. The collective cost is

$$\epsilon^* 2c^2(n+m) + \epsilon^* c(n+m) + (\epsilon^* - 1)c^3 + (\epsilon^* - 1)c^2 + 2(\epsilon^* - 1)c \tag{4.15}$$

Another way of constructing matrices S and R is by exploiting the observation that the multiplication of two function matrices $F_{X\to Y} \times F_{Y\to Z}$ results in the the function matrix $F_{X\to Z}$ for X, Y, Z cutsets of consequent unfoldings. For example, $S = S_{\epsilon^*} S_{\epsilon^*-1} ... S_0$ can be constructed as a single matrix performing DFS to find all the paths for the source-destination pairs of vertices of cutsets X_{ϵ^*} to X_0, avoiding the additional matrix multiplications. This requires $(\epsilon^* + 1)c^2(n+m)$ operations (c^2 DFS pairs of source-destination vertices, each DFS considers vertices and edges of $\epsilon^* + 1$ consequent unfoldings). R can be constructed by evaluating each element and performing the maximization $R = \oplus\{R_i S_{i-1} S_{i-2} ... S_0 | 0 \le i < \epsilon*\}$ using again DFS between pairs of cutsets X_i to X_0, where $0 \le i < \epsilon*$. In this way, the construction of R requires $(1 + \ldots + \epsilon^*)c(n+m) + \epsilon^* c$ operations (ϵ^* elements of maximization, each element $i \ldots c$ DFS pairs of source-destitution, where $0 \le i < \epsilon^*$). The collective cost is

$$(\epsilon^* + 1)c^2(n+m) + (1 + \ldots + \epsilon^*)c(n+m) + \epsilon^* c \tag{4.16}$$

The problem scales with sizes n, m and c. It is clear now how the choice of cutset determines the complexity of the algorithm. The size of cutset is always smaller than total number of nodes plus edges of the graph, *i.e.* $c < n + m$. So, the complexity of both approaches is of $O(n+m)$. However, for great numbers of ϵ^* the second approach is a constant number of operations better.

Overall Complexity

The overall complexity can be derived by looking at the pseudocode of **TSE**. Considering a Process Graph G and the size of vertex set as $|V| = n$ and edge set as $|E| = m$. The extraction of cycles on line 1 is performed by DFS on the Process Graph. This processing is performed in $O(n+m)$ time. The second step of evaluating the maximum ratio cycles requires time $O(L)$, where L is the total number of cycles in graph G. It holds that $L < n$. The computation of ϵ^* requires the construction of graph

G^*, as defined previously. Considering n^* and m^* the size of vertex and edge set of G^*, respectively, the construction of G^* is performed in $O(n^* + m^*)$ and the actual computation of ϵ^* in $O(n^* + m^*)$, as it requires a DFS application on G^*. The unfolding and the processing of the unfolding in lines 5 and 6 require $O(n + m)$ time each. The unfoldings that will be applied are $k^* + \epsilon^* + \beta$. The the complexity of the computation so far is $O((k^* + \epsilon^* + \beta)(n + m))$.

With a high probability the algorithm will terminate in line 9, so with high probability further computation won't be required. If the execution continues the size of cutset is the determining factor for the complexity. Let C be the size of the cutset. The creation of the cutset is performed in $O(C)$ time. Matrix T is $Cx1$ array and is constructed in time $O(C)$. Accordingly, matrices R and S are CxC arrays and are constructed in time $O(C^2)(n + m)$. The reason for this, is that DFS is applied to find all paths from vertices of one cutset to another. The closure of S is evaluated in $O(C^3)$ time. The matrix multiplications on line 16 are performed in $O(C^3)$ time. The last computation requires constant time. Since $C < (n + m)$, the final complexity of TSE algorithm is $O(C^2)(n + m)$.

4.7.3 Minimun Analysis

For the implementation of minimum analysis the same structures and methods as for the maximum analysis are used with the appropriate changes to fit the change of meaning from maximum to minimum timing separation. The minimum analysis was implemented in both versions of the TSE tool.

4.7.4 Floating Point Arithmetic

The implemented tool for the TSE was expanded to accommodate floating point arithmetic,. The presented theory coveres integer values, however, in practice the time values of real; circuits are hardly ever integers. So all of the involved fields in the implementation were changed from **INTEGER** type to **DOUBLE** type. This change required, a careful change due to the wide use of ∞ in theory, especially in the conditional control, where comparisons to the ∞ were used, and in operations with ∞. The representation of the ∞ also changed from **MAXINT** to **FLT_MAX**.

4.8 TSE Analysis vs. worst-case analysis

Special notice should be taken in the difference of meaning of TSE analysis compared to worst-case analysis that is the standard timing analysis process. The difference is shown through an example of both analysis on the graph of Figure 3.1.

If worst-case analysis is followed, each event occurrence is assigned the latest possible time value. So, for every edge the higher delay D is chosen. A portion of the unfolded graph of the example graph of Figure 3.1 is shown in Figure 4.10, with a worst-case timing assignment for the event occurrences. It is clear that, for event **a**, after a number of occurrences, the time value of a_i is always defined as

$t(a_i) = t(a_{i-1}) + 20$ and is always determined by the edge $b \rightarrow a$ and the upper bound of delay D.

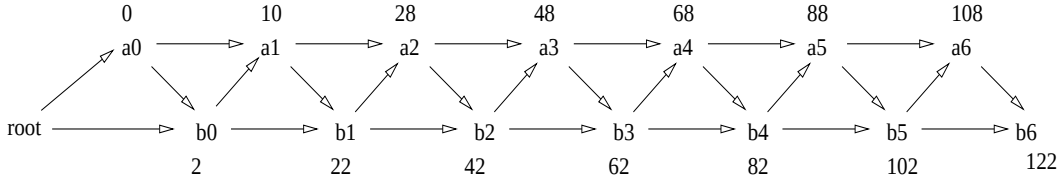

Figure 4.10: Unfolded process graph with worst-case timing analysis annotation

However, if maximum timing separation between events is evaluated, a different timing assignment policy must be followed. For example, if timing separation between events a_5 and a_6 is to be estimated, for the event occurrences from $root$ until a_5 minimum delay assignment is followed (for each edge the minimum delay is chosen), while from a_5 to a_6 maximum delay assignment is followed. This assignment is shown in Figure 4.11. This results in a timing separation of 25, while worst-case analysis results in a separation of 20. This is because worst-case analysis assumes maximum delay for the preceding event occurences, while this isn't the case in maximum time separation analysis.

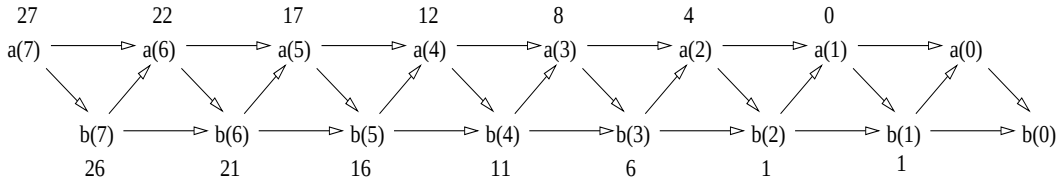

Figure 4.11: Unfolded process graph with maximum timing separation analysis between events a_5 and a_6

One more thing to be noticed is that the maximum ratio cycle in this graph is $b \rightarrow b$, and it is the edge $b_i \rightarrow b_{i+1}$ that determines the delay in both analyses. This indicates that by optimizing the elements that affect these edges will improve the overall timing of the circuit.

In this chapter were presented all the theoretical aspects of the previous work that were examined in this thesis, in order to form a complete picture of the TSE algorithm. Moreover, implementation issues were examined as well as the estimated complexity of the algorithm. The implementation of TSE accomplished during this work was described, as a ATA tool for asynchronous circuits. In the following chapter application of this ATA tool is examined, within well-defined EDA flows, for the optimization and RTC validation.

5

Application of TSE tool

As mentioned in Chapter 2 TSE analysis can be useful for optimization of asynchronous circuits or for the validation of some imposed RTCs. In this Chapter the use of the TSE analysis will be presented within a flow for optimization, compared and contrasted with the corresponding flow in the synchronous design process. The implemented ATA tool was used for the optimization of real designs through the presented flow.

5.1 Optimization

The optimization flow that is followed in synchronous design appears in Figure 5.1. The designer provides a description of the circuit in HDL. The circuit is synthesized into a specific technology using a library of gates. STA is performed on the mapped netlist. STA provides information on the timing of the circuit and on critical paths, which are used for optimizations. The optimizations are changes on the netlist, depending on the information of STA and the technology library that is being used. The optimized netlist is again analyzed to evaluate the new implementation and verify the effectiveness of the applied optimizations. Optimizations are applied in a closed-loop of timing analysis and changing the implemented netlist, until a satisfactory implementation is reached.

An analogous procedure is followed in the flow presented in this work as it appears in Figure 5.2.

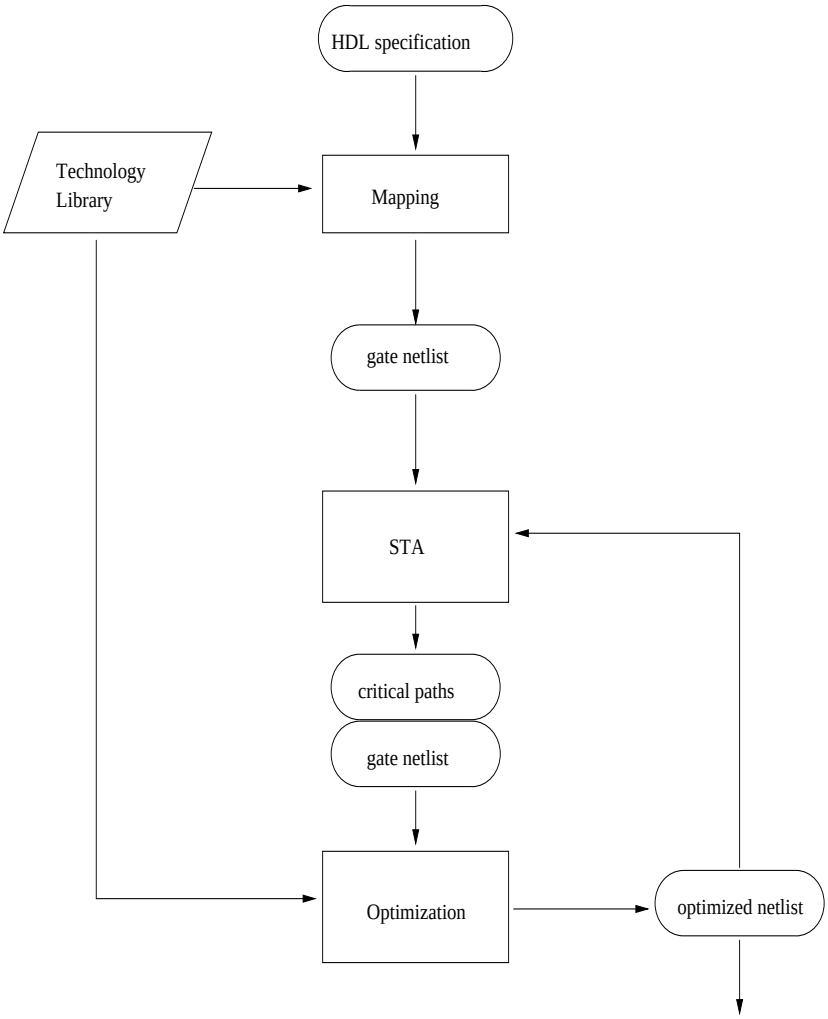

Figure 5.1: Optimization flow using ATA.

Initially, an asynchronous specification is provided and it is mapped in the desired technology library. The mapped netlist is analyzed using ATA. Based on this analysis the circuit is optimized and analyzed again until it reaches an accepted timing performance or cannot be further optimized. The flow is the same with synchronous design, changing the particular steps of the flow with appropriate for asynchronous design techniques. For example, STA is replaced with ATA. The specification, also, requires the STG of the design. Moreover, the concept of critical paths is replaced with critical cycles as a target for optimizations. The steps of the flow are described in more detail in the next sections along with application on real designs.

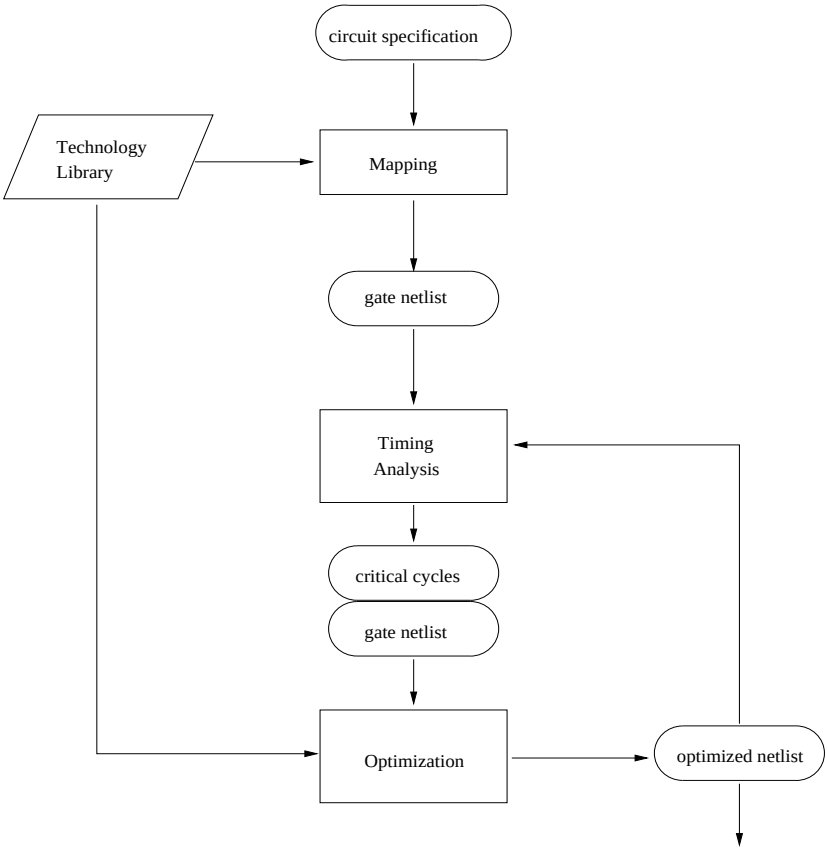

Figure 5.2: Optimization flow using ATA.

5.1.1 Circuit Specification

As examined in Chapter 4, the required specification model for ATA is the process graph. Models specifying the behavior of circuits were also presented in Chapter 2. The specification used throughout this work for the modeling of asynchronous circuit behavior is the STG model, with a delay range labelling on the edges. However, the most direct description of a circuit is in a HDL netlist form. The STG model, describes the behavior of the circuit as well as the behavior of the environment and the interface protocol. An STG can be derived from a circuit, having an HDL description, the interface protocol, and an indication or specification of the behavior of the environment. The conversion from a netlist to STG is presented through an example.

Figure 5.3 shows a netlist of an Relative-Timed Burst Mode (RTBM) controller [10]. This controller implements a handshake protocol on the right and a handshake on the left side. The circuit implements the following boolean equations.

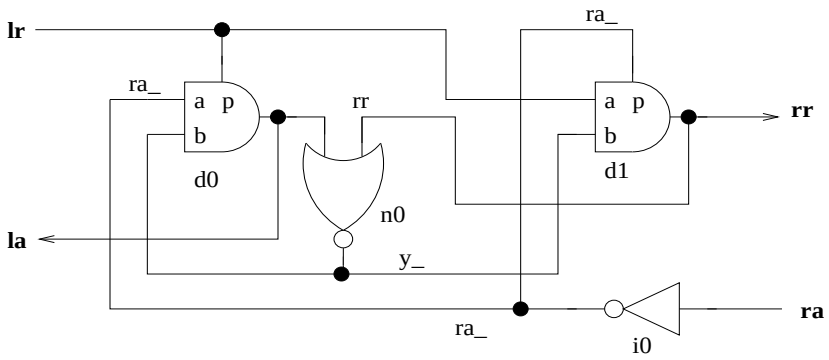

Figure 5.3: controller netlist

$$la = rst' + (lr \cdot ra\_ \cdot y\_) + (lr \cdot la \cdot ra\_') + (lr \cdot la \cdot y\_') \tag{5.1}$$

$$rr = rst' + (ra\_ \cdot lr \cdot y\_) + (ra\_ \cdot rr \cdot lr') + (ra\_ \cdot rr \cdot y\_') \tag{5.2}$$

$$ra\_ = ra' \tag{5.3}$$

$$y\_ = (la + rr)' \tag{5.4}$$

Based on the netlist (boolean equations) and the interface protocol, the state graph of Figure 5.4 can be constructed. The protocol followed is $req_+ \rightarrow ack_+ \rightarrow req_- \rightarrow ack_-$. The red arcs on Figure 5.4 represent a race condition present on the circuit. On lr_+, la_+ and rr_+ will occur. However, if la_+ occurs before rr_+, a race will take place between y_- and rr_+. If y_- occurs before rr_+, rr_+ will not occur. With appropriate gate delays the race is avoided.

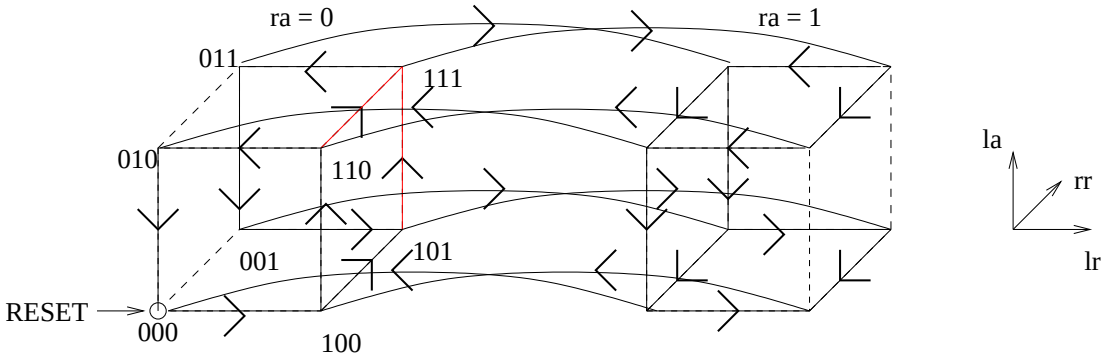

Figure 5.4: State Graph Analysis of the controller

The corresponding STG of Figure 5.5 can be derived from the previous state graph. The main idea is that each transition towards one direction on the state graph corresponds to a signal transition on

the STG. Examining the state space, we can observe that each transition towards the same direction, *i.e.* one transition on the STG, is able to occur only if specific conditions hold. For example, for la_+ to occur, lr must be risen and ra must be fallen. This means that there are two edges on the STG that lead to la_+, $lr_+ \rightarrow la_+$ and $ra_- \rightarrow la_+$. Similarly, for rr_- to occur, ra must be risen, leading to one edge on the STG, $ra_+ \rightarrow rr_-$.

Moreover, tokens are required to model the correct behavior of the controller. They must be placed on appropriate edges, *i.e.* where they are required to initiate the handshakes after reset. Some rules must be followed, such as, each cycle on the STG must contain a token and no redundant tokens must be placed on the STG.

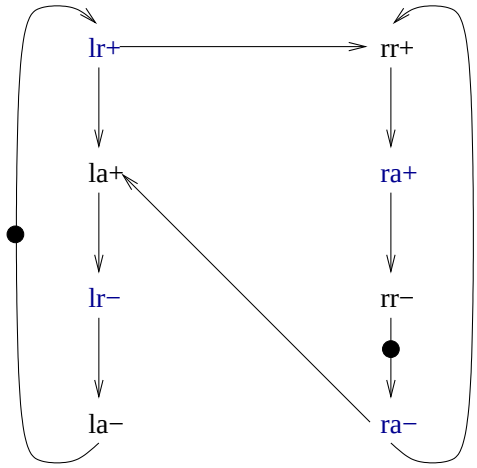

Figure 5.5: Signal Transition Graph of RTBM controller

The delays on the edges are derived from the gate delays of the path that is followed on the circuit, for this signal transition to occur. For example, the edge $lr_+ \rightarrow la_+$ indicates a rising transition of signal lr which will lead to a rising transition of signal la, through the gate $d0$ of the circuit. Thus, the delays of this edge will be the delays of gate $d0$. Accordingly, the edge $ra_- \rightarrow la_+$ refer to the path $i0 - d0$, so the delays will be the delays of gates $i0$ and $d0$.

Some edges refer to the behavior of the environment such as $la_+ \rightarrow lr_-$ or $rr_+ \rightarrow ra_+$. These edges will be assigned delays depending on model to be simulated. If the behaviour of the controller irrelevant of its environment is wanted, then zero delays will be assigned. If fast or slow environment handshaking is required then appropriate delays will be assigned. If the controller is connected with some other logic then the delays will depend on this logic.

5.1.2 ATA and Optimization

Since the design specification is available, ATA can be applied. The separation of different events can be analyzed. What is useful is the cycle period of the circuit that is the time between a successive rising or falling transition of a signal. In the specific case of an asynchronous controller a useful event separation is between consequent rising and falling transition of the request signal.

Whatever the event separation is evaluated, the repetition parameters as well as the critical cycles of the analyzed circuit are the same. Moreover, the critical cycles of the STG are those that mainly determine the timing of the circuit. Thus, critical cycles indicate a target for optimizations, in correspondence to a critical path in synchronous design.

Let's assume the controller of Figure 5.3 with zero delay environment and with gates $i0, n0$ of 1-unit delay while gates $d0, d1$ of 2-units delay. Running the ATA tool on any event separation will provide the critical cycles of the circuit. In the specific example evaluating the event separation $lr_+ \rightarrow lr_+$ yields a maximum timing separation of 6 units, while the event separation $lr_+ \rightarrow la_+$ yields 4 units maximum separation. The critical cycle appears to be $ra_- \rightarrow rr_+ \rightarrow ra_+ \rightarrow rr_- \rightarrow ra_-$. This indicates that optimizations must target this specific cycle of the STG, *i.e.* the gates $i0$ and $d1$. By changing gate $d1$ from 2-units to 1-unit delay gate and evaluating the same event separation will result in reduced delays. The evaluation of $lr_+ \rightarrow lr_+$ gives 5 units of maximum timing separation, while $lr_+ \rightarrow la_+$ gives 3 units maximum separation. Not only the transitions that are directly dependent on the optimized gate ($lr_+ \rightarrow lr_+$) are effected but others as well ($lr_+ \rightarrow la_+$). This is expected since as explained previously the critical cycles are the ones determining the universal timing of the design.

The critical cycle is the same after the optimizations but it could as well have changed in a different example or for different timing specifications. Optimizations based on critical cycles could be further applied if allowed by the technology used.

Now, let's assume 2-unit delay gate $d1$, while changing gate $d0$ to 1-unit delay gate. By performing ATA, the event separation $lr_+ \rightarrow lr_+$ is still estimated as 6 time units and the event separation $lr_+ \rightarrow la_+$ is still estimated as 4 units maximum separation. This indicates that by changing a gate outside the critical cycle doesn't lead to optimizing the overall timing of the circuit.

The flow described provides a well-defined optimization procedure that can be followed for the optimization of asynchronous circuits. Provides ways to optimize only critical for the timing parts of the circuit, while keeping the other as they are. So, by initially implementing the design for minimum area and then optimizing it for time on the critical parts, the resulting implementation will be an intermediate trade-off, *i.e.* a point on the Pareto curve.

5.2 Relative Timing Constraints (RTC) Validation

Another application of ATA would be for the validation of RTCs. RTC are assumptions made for the timing behavior of a circuit. The assumption that a signal will arrive before another or a specific unit is faster than another may be convenient for the synthesis of the circuit. A relative timing assumption may lead to various simplifications on the specification of the circuit resulting in an optimized implementation [4]. However, a timing assumption that is incorporated in the implementation, must hold for the circuit to operate properly, *i.e.* constitutes an operational constraint.

Currently, there is no specified way of knowing whether an implementation with RTC will respect this constraint in all cases. The behavior of the implemented system can be examined through simulations. Simulations, though consider single values of timing on event occurrences and cannot simulate a system with variable delays. ATA can be applied in this area too. Since ATA evaluates bounds on the timing separation of events, is a way of identifying whether an event appears before another. The exact bounds provide a way to validate a relative timing assumption between two events or signals, through the flow of Figure 5.6.

5.3 An automated tool for optimization

ATA, so far, appears as a useful tool in many directions. But to evaluate the practical effectiveness of its application, a simple optimization tool was implemented in this work. It realizes the optimization flow described in previous section in an automated procedure. It supports mapping functionality from a netlist description to a specified technology library, timing analysis using the implemented ATA tool and optimization functionality through gate resizing.

The optimization tool enumerates about 2500 lines of C code, as well as various scripts for the manipulation of the technology libraries. It consists of a netlist parser, stg file parser, mapper and optimizer. It requires the description of the circuit in basic functional units, *i.e.* AND, OR, INV and the specification of the behavior of the circuit in STG format. Additionally, it requires a mapping of STG edges to functional units. This is necessary for directing optimizations of STG cycles to optimization of gates in the circuit.

5.3.1 Technology Library

The implemented optimizing tool supports a subset of the `UMC13` 130nm library. It was acquired through various scripts processing `UMC13` library in order to extract the required information and integrate the specification of all three analysis corners (best, typical, worst) of the particular subset. This subset includes gates that implement all the basic functionality and is able to implement any logic

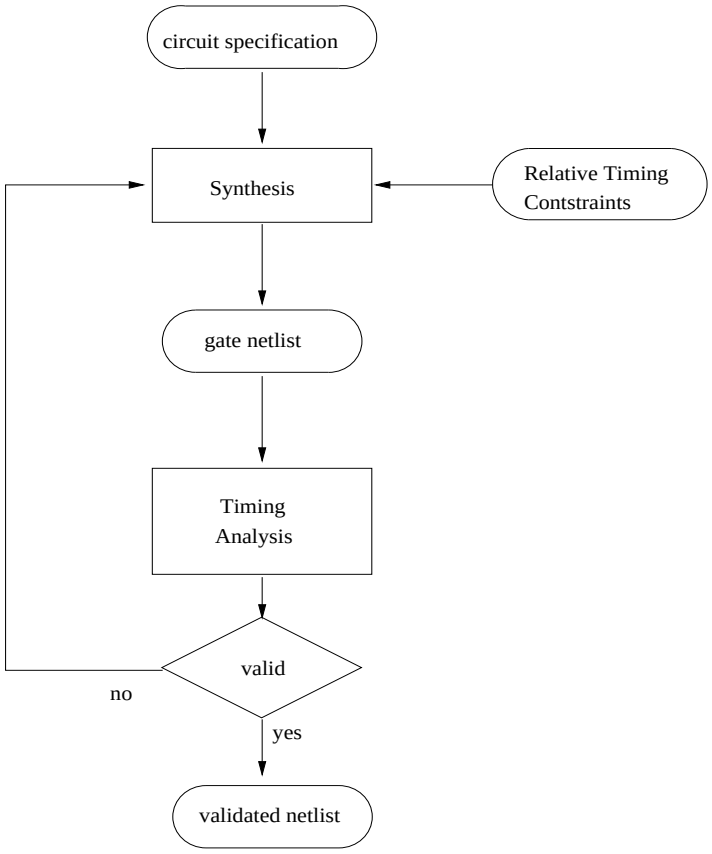

Figure 5.6: Validation flow using ATA.

function. It doesn't include sequential elements, multiplexers, adders and some complex gates. The available attributes for every cell of the library is the area it covers, the capacitance of each input pin, and an array of gate-delays indexed by the drive capacitance of the gate. The delay array supports all three corners of analysis (*i.e.* worst, typical, best-case). Rise and fall times are not considered in the analysis.

A parser of the library was implemented that constructs a directory of the library cells, that provides easy search functionality indexed by the logic function of the cell.

5.3.2 Input Netlist

A parser for the input netlist was a part of the tool. The parsed netlist can be an already mapped in the same library netlist or in an HDL form. That is the description of the netlist as a graph of gates. Specifically, the netlist graph is described using a function library, where each function unit represents a gate. The parser also supports some syntax checking functionality. The netlist is constructed as a graph of nodes, that is later processed for mapping.

5.3.3 Input STG

The STG specification of the circuit is also required for the ATA. If the circuit netlist is available the circuit specification in stg format is derived manually following the process explained in Chapter 5. If the stg specification is available, an implementation of the circuit can be produced. Moreover, an association of stg edges with is netlist gates required. This is a way to relate time ranges of stg edges to gate delays. In other words, to define which part of the implementation determine which part of the behavior of the circuit. This is also constructed manually, by examining the behavior of the netlist in each signal transition.

5.3.4 Optimization Process

Having all the above information the optimization process can be applied. The implemented optimization tool can apply mapping functionality aiming at specific directions. It can provide a mapping of the circuit in a technology library targeting at minimizing area, minimizing timing or exploring intermediate solutions. The applied mapping doesn't perform restructuring of the netlist, but rather resizing of gates. The aim of the specific optimization tool was not a good mapping functionality, but the ability to explore good trade-offs that lie on the Pareto curve, providing a target direction and the means that lead to this direction.

ATA was used in this direction. As explained before, critical cycles are the ones that determine the timing in asynchronous circuits. As shown through examples, a change in the delays of critical cycles will affect the overall timing. Moreover, the timing separation between consequent occurrences of an event (cycle period) can be a performance evaluation metric. These characteristics of ATA were exploited within this optimization tool. The estimated critical cycles constitutes the target of timing optimizations and the TSE between events is the means to decide whether a better performance is achieved.

The optimization process follows the flow of Figure 5.7. Initially, the circuit with its specification is read and an initial mapping is produced if necessary. The delays of the gates are assigned to the STG specification according to the specific mapping, as a pre-step of ATA, which is then performed using the ATA tool. This analysis will provide a performance evaluation of the circuit along with its critical cycles. The netlist can be optimized based on those critical cycles, which means that only the parts of the netlist that affect the critical cycles will be optimized. The delays of the new mapping are set and ATA is again performed. The performance evaluation gives a qualitative and quantitative metric on the optimizations performed and the current critical cycles give a new direction for optimizations. The closed loop of optimization and analysis will repeat until a the netlist cannot be further optimized

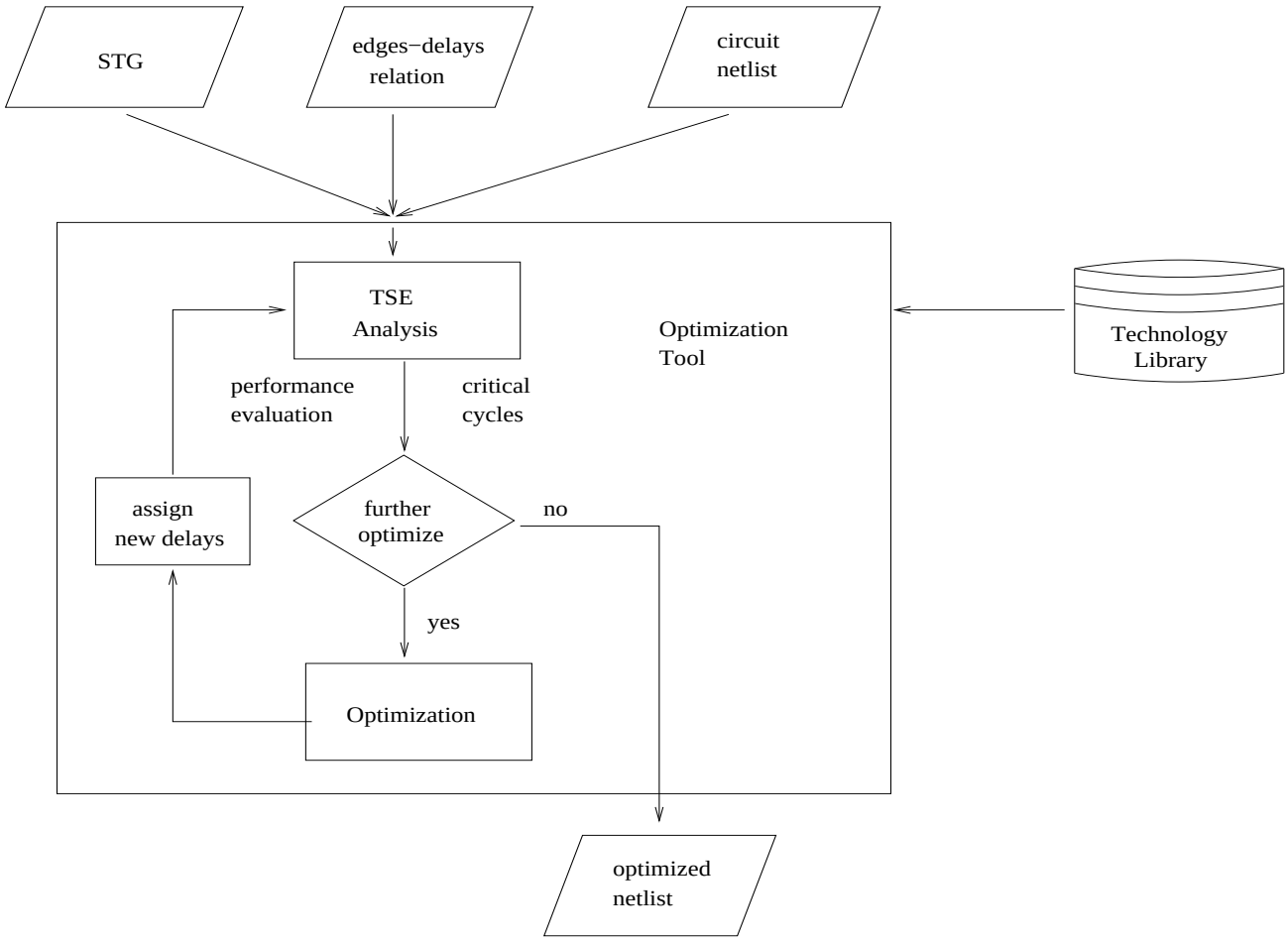

Figure 5.7: Optimization tool process

on the changes result in a worst-implementation. Changes on how the terminating decision is taken may lead to different ways of exploring the space of implementations.

On performing an optimization on a critical cycle of the STG, requires extracting the paths or trees of the netlist that are associated with this cycle. When they are acquired, resizing is performed in each path starting from the endpoint towards the starting points. During this process drive capacity of each gate is considered and their delays are set accordingly. If the implementation of a gate changes, the gates driving the changed one are additionally optimized in order to drive the new capacitance.

The delays that are set after an optimization is performed are the delays of the gates, as they are defined in the library, respecting the capacitance that needs to be driven in the particular mapping. Moreover, the optimization tool can be configured to consider worst-, best-, typical-case corners or a range considering all possible cases.

In this Chapter some applications of the ATA on the asynchronous hardware design were presented. As shown, an ATA tool can form the basic building block in various processes that will facilitate asynchronous design. It can form the base functionality for optimization and RTC validation. During this work, the effectiveness of ATA was studied through the implementation of an automated optimization tool. Several asynchronous circuits were analyzed and optimized using the implemented tools and the flows described. The results are presented in the following chapter.

6 Results

This chapter demonstrates the application of the implemented tools and flow for the optimization of real, asynchronous circuits. The conventional ”point-to-point” STA optimization flow was used as a point of reference. The results were compared and contrasted in order to evaluate the effectiveness of ATA optimization flow on asynchronous designs, compared to STA.

6.1 Experimental Procedure

The circuits used to generate comparative results were real asynchronous control circuits used in practical designs. The circuit benchmarks include various types of latch controllers, based on asynchronous handshakes, circuits useful for desynchronization of synchronous systems and a VME bus control circuit.

The experimental procedure followed is shown in the flow diagram of Figure 6.1. Two separate flows were used. The conventional optimization flow, which is based on widely used conventional EDA tools and the novel proposed flow, which is based on ATA and the optimization tool developed in this work.

The circuit’s description was provided in HDL. Its STG specification was derived manually, as described in Chapter 5. A relation between STG edges and netlist’s gates was also derived manually.

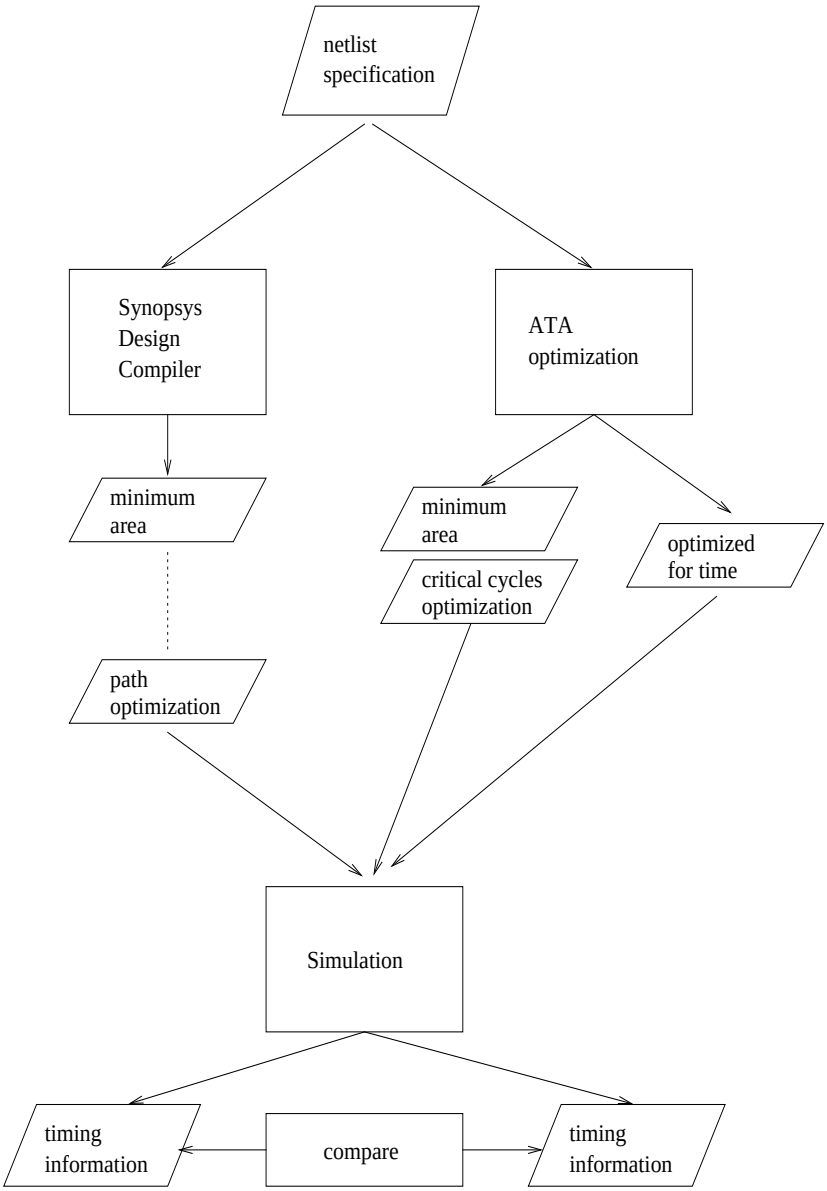

Figure 6.1: Experimental Procedure Flow.

These were provided as inputs to the flow.

Using the implemented optimization tool the netlist was initially resized for minimum area. When resizing for area, smallest library elements were used, taking no consideration to timing constraints. Then, the optimization loop was applied on the minimum area netlist, focusing on critical cycles. Several iterations of the loop produced several implementations depending on the critical cycles of the current implementation. The terminating condition is that no further optimization could be performed on the critical cycle delays. Area constraints weren't considered through this optimization. However,

optimizing only the critical parts of the netlist leaves the remaining parts already optimized for minimum area. Another netlist was produced resized for minimum delay, examining each component of the netlist. Area constraints were not considered in this implementation.

All the implemented netlists were simulated, in order to measure the actual performance of each design. For the simulations the tools `Cadence VerilogXL-Simvision` were used. One point of inaccuracy not in favor of this flow is that rising and falling transition times are not accounted for. The implemented tool doesn't take into account transition times, as the aim was not an efficient mapping tool but the evaluation of ATA analysis for optimization.

Alternative optimization was performed based on the conventional flow. The conventional flow using `Synopsys Design Compiler` performs "point-to-point" STA breaking the cycles in the netlist, disabling cyclic dependencies. Timing optimizations must, thus, be applied only to paths that aren't disabled. Thus, the path optimizations that were considered, concerned critical paths, as identified by `Synopsys Design Compiler`, typically all paths from inputs to outputs or specific internal netlist paths, depending on the design. This flow required manual exploration for the paths to be optimized, which were determined manually, by selecting end-points.

A further difference between the two flows is that `Synopsys Design Compiler` performs technology mapping on circuits, not gate resizing. In order to fairly compare the two flows, the option that forces `Synopsys Design Compiler` to apply resizing only was used.

Finally, all of these implementations were simulated and the results were compared.

In the following sections, timing and area measurement results for each implemented design are presented, as obtained from simulations, ATA and the `Synopsys Design Compiler` Timing Engine.

6.2 Two-phase Overlapping Desynchronization Controller

The desynchronization controller is a circuit which controls data flow in a desynchronized design, implementing left and right handshakes and latch control signals, according to the STG specification shown in Figure 6.2. The transitions which appear in blue, are input signals provided by the environment. Thus, such signals and edges leading to them, also in blue, represent the environment's behavior and their actual delays are dictated by the environment. The controller's implementation is shown in Figure 6.2.

The STG along with the circuit's netlist and a mapping of edges to gates are provided as input to the optimization tool. ATA based optimization was performed for the two-phase controller in a number of configurations, *i.e.* 1-scale and 3-scale ring and a fork-join pipeline.

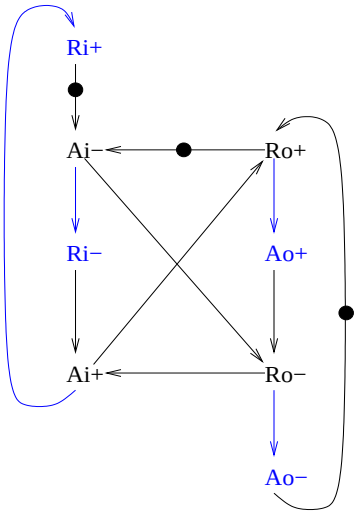

Figure 6.2: Desynchronization controller STG

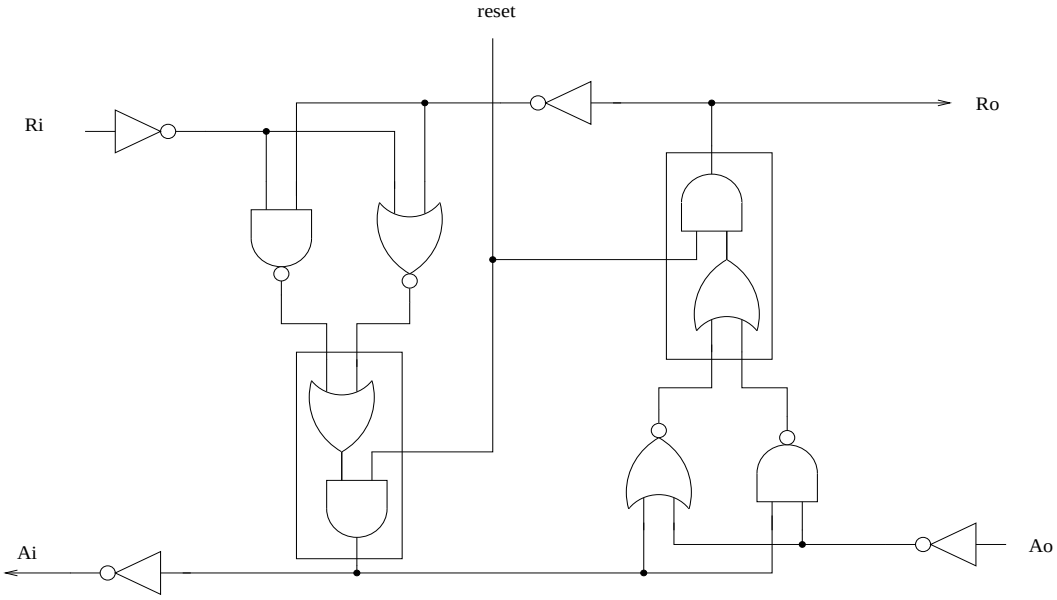

Figure 6.3: Desynchronization controller netlist

6.2.1 Scale of 1 ring controller

1-scale ring is shown in Figure 6.4 and abides with the behavior specified by the resultant STG shown at the same figure. The results of the measurements are shown in Table 6.1.

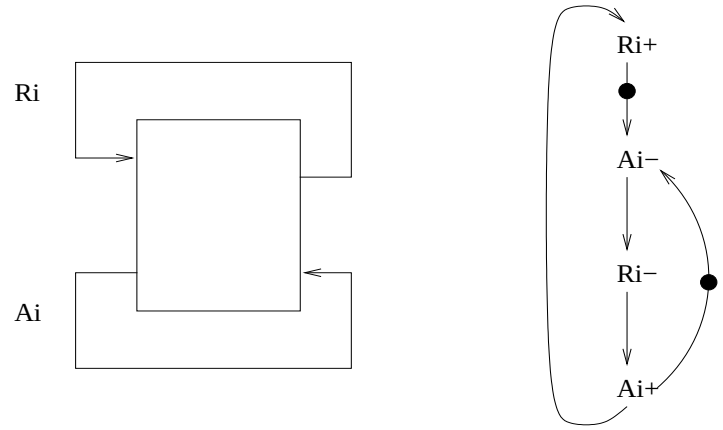

Figure 6.4: Two-phase desynchronization controller netlist in 1-scale ring

6.2.2 Scale of 3 ring controllers

The desynchronization controller was, also, examined in a 3-scale ring pipeline. The block digram of the topology and the resultant STG for this configuration are shown in Figure 6.5, while results obtained by measurements are shown in Table 6.1.

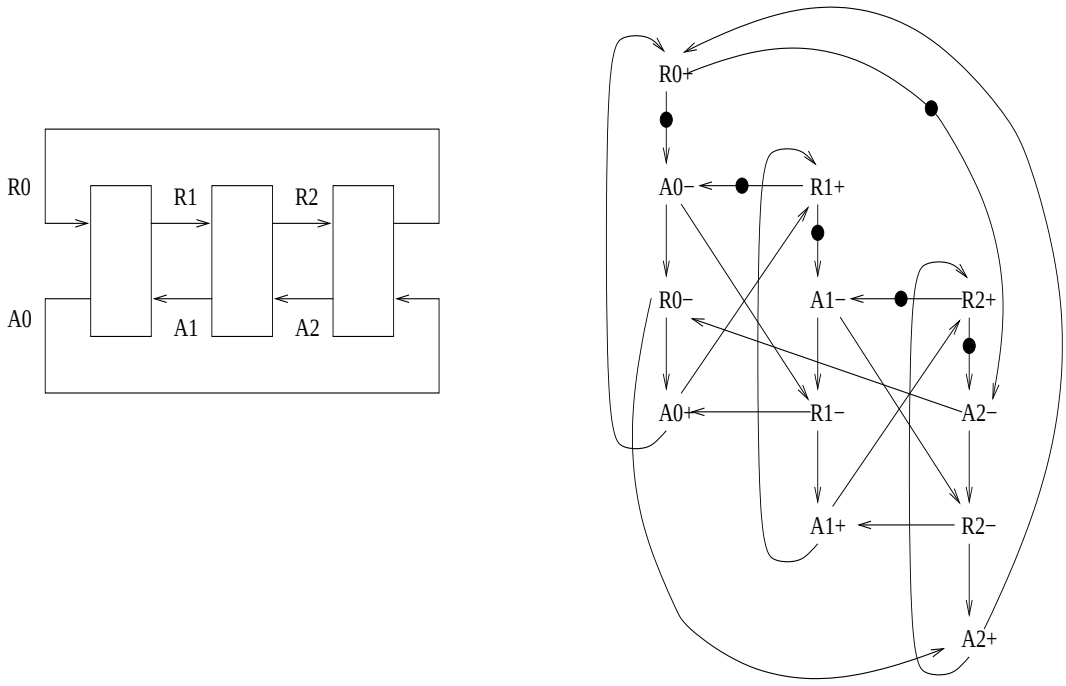

Figure 6.5: Two-phase desynchronization controller netlist in 3-scale ring

6.2.3 Scale of 3 ring controllers including wire delays

In order to model the delay spent on a long wire, a load was assigned to wires of a connection. On the design of the desynchronization controller connected in a 3-ring pipeline, a load was assigned to wires connecting the last and the first controller, modelling a long connection after place and routing. The results appear in Table 6.1.

Table 6.1 shows three rows of implemented netlists for each configuration. The first row includes the netlists which were mapped and optimized using the novel, proposed ATA based optimization tool. The first netlist of each row was resized for minimum area without considering the timing of the netlist. The second netlist was optimized focusing on critical cycles that were obtained using ATA tool. The third netlist of the row was mapped focusing on minimum delay on each element of the netlist individually.

The other two rows show results obtained using the conventional flow. Netlists of the second row were obtained applying only resizing and the netlists of the third row not only through resizing but also through resynthesizing.

The first column shows simulation measurements for the implementations between consequent Ri_+ occurrences. In the parentheses in the same column the same measurement is shown, as obtained by the ATA tool. In the second column the corresponding area measurements are shown. Finally, in the third column the percentage improvement of ATA-based optimization to STA-based optimization is shown. The comparison is between Critical Cycle Optimized netlist and the most timing efficient Path Optimized netlist.

6.2.4 Fork-join Pipeline Structure

A more complicated structure is shown in Figure 6.6. Timing and area results of implementations, obtained using the novel ATA and the conventional flow are shown in Table 6.2.

Based on simulation results of Tables 6.1 and 6.2 several observations can be made. ATA-based optimization, based on critical cycles, is able to effectively optimize timing and in all cases produced improved timing compared with conventional flow.

Gate delay optimization, *i.e.* optimization of every gate of the netlist, didn't prove better than critical cycle optimization in most cases. This is explained since the effort to optimize every element of the implementation also optimized the non critical elements. This might have affected the drive load capability of gates on the critical cycles leading to worse timing and greater area.

The difference of the ATA and the timing measured in simulations in the netlists produced through ATA is an estimation error caused by rising and falling transition times. This is further supported by

Desynchronization Controller			Simulation (ns) $Ri_+ \rightarrow Ri_+$	Area (μm^2)	Improvement %
1-scale ring	ATA	Minimum Area	$3.269ns(2.046ns)$	$79.488\mu m^2$	
		Critical Cycle Optimization	$1.425ns(1.174ns)$	$124.416\mu m^2$	
		Gate Delay Optimization	$1.464ns(1.174ns)$	$129.6\mu m^2$	
	STA-resizing	Minimum Area	$1.603ns$	$115.776\mu m^2$	
		Path OptimizationA	$1.718ns$	$89.856\mu m^2$	**11%**
		Path OptimizationB	$1.627ns$	$91.584\mu m^2$	
	STA-resynthesis	Minimum Area	$1.664ns$	$77.76\mu m^2$	
		Path OptimizationA	$1.639ns$	$103.68\mu m^2$	**5 %**
		Path OptimizationB	$1.508ns$	$86.4\mu m^2$	
3-scale ring	ATA	Minimum Area	$2.789ns(2ns)$	$165.888\mu m^2$	
		Critical Cycle Optimization	$1.377ns(1.204ns)$	$297.216\mu m^2$	
		Gate Delay Optimization	$1.396ns(1.174ns)$	$300.672\mu m^2$	
	STA-resizing	Minimum Area	$1.458ns$	$264.384\mu m^2$	
		Path OptimizationA	$1.66ns$	$224.64\mu m^2$	**17 %**
		Path OptimizationB	$1.682ns$	$238.464\mu m^2$	
	STA-resynthesis	Minimum Area	$1.694ns$	$120.960\mu m^2$	
		Path OptimizationA	$1.329ns$	$193.536\mu m^2$	**-3 %**
		Path OptimizationB	–	$139.968\mu m^2$	
3-scale ring with wire delay	ATA	Minimum Area	$2.735ns(2ns)$	$176.256\mu m^2$	
		Critical Cycle Optimization	$1.372ns(1.429ns)$	$305.856\mu m^2$	
		Gate Delay Optimization	$1.372ns(1.47ns)$	$362.880\mu m^2$	
	STA-resizing	Minimum Area	$1.78ns$	$267.84\mu m^2$	
		Path Optimization	$1.996ns$	$231.552\mu m^2$	**31 %**
	STA-resynthesis	Minimum Area	$1.853ns$	$177.984\mu m^2$	
		Path Optimization	$1.912ns$	$196.992\mu m^2$	**28 %**

Table 6.1: Desynchronization controller in 1-scale, 3-scale ring and 3-scale ring including wire delays results.

the fact that the difference is greater in the first netlist, *i.e.* optimized for minimum area, where smaller gates can be more affected by transition times.

Through the ATA optimization process, several implementations of the same netlist were produced, between changes on gates. Simulations were also performed on these intermediate netlists and the measurements appear on the graphs of Figures 6.7, 6.8, 6.9, 6.7.

6.3 RTBM Controller

RTBM controller is a Relative-Timed, Burst Mode latch controller. Its netlist and STG were shown in Chapter 5 and are shown again in Figure 6.11 and Figure 6.12. In this section we consider its implementation. As this circuit includes a timing assumption, it is possible for netlist optimization to

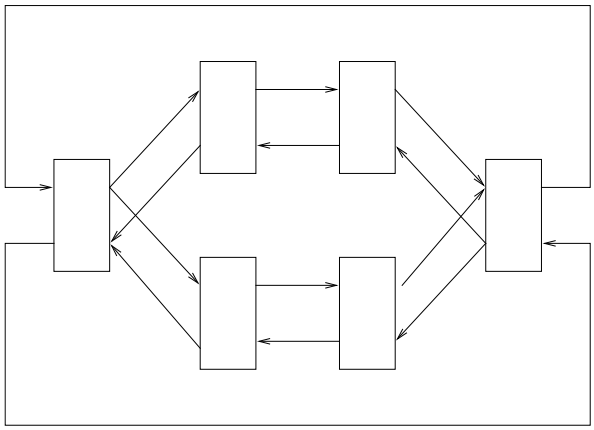

Figure 6.6: A fork-join desynchronization controllers' structure

Desynchronization Controller			**Simulation** (ns) $Ri_+ \rightarrow Ri_+$	**Area** (μm^2)	**Improvement %**
fork-join pipeline	**ATA**	Minimum Area	$2.875ns(2.217ns)$	$345.6\mu m^2$	
		Critical Cycle Optimization	$1.631ns(1.427ns)$	$615.168\mu m^2$	
		Gate Delay Optimization	$1.762ns(1.39ns)$	$615.168\mu m^2$	
	STA-resizing	Minimum Area	$1.664ns$	$537.408\mu m^2$	
		Path OptimizationA	$1.902ns$	$468.288\mu m^2$	**3 %**
		Path OptimizationB	$1.677ns$	$437.184\mu m^2$	
	STA-resynthesis	Minimum Area	$1.953ns$	$388.8\mu m^2$	
		Path OptimizationA	$1.914ns$	$499.392\mu m^2$	**15 %**
		Path OptimizationB	$2.137ns$	$343.872\mu m^2$	

Table 6.2: Desynchronization controller in fork-join pipeline results.

violate it. This would result in incorrect realization. Two configurations were studied, *i.e.* 1-scale and 4-scale rings. The results are shown in Table 6.3.

The measurements indicated as **FAILED** on Table 6.3, on the simulation field, indicates that in these cases the relative timing constraint was violated, and the design didn't operate correctly.

Similar observations can be made, with respect to this design, as they were in the previous, based on simulation results. The optimization performed based on critical cycles was comparable, but slightly worse, in the 1-scale ring and better in the 4-scale ring. The timing performance reached through critical cycles optimization was very close to overall gate delay optimization. The difference between ATA and simulation measurements were apparent in this design, as well.

The negative percentages of the first configuration, *i.e.* 1-scale ring, and the small improvement on the second configuration, *i.e.* 4-scale ring, compare with the STA-results, are due to the STG

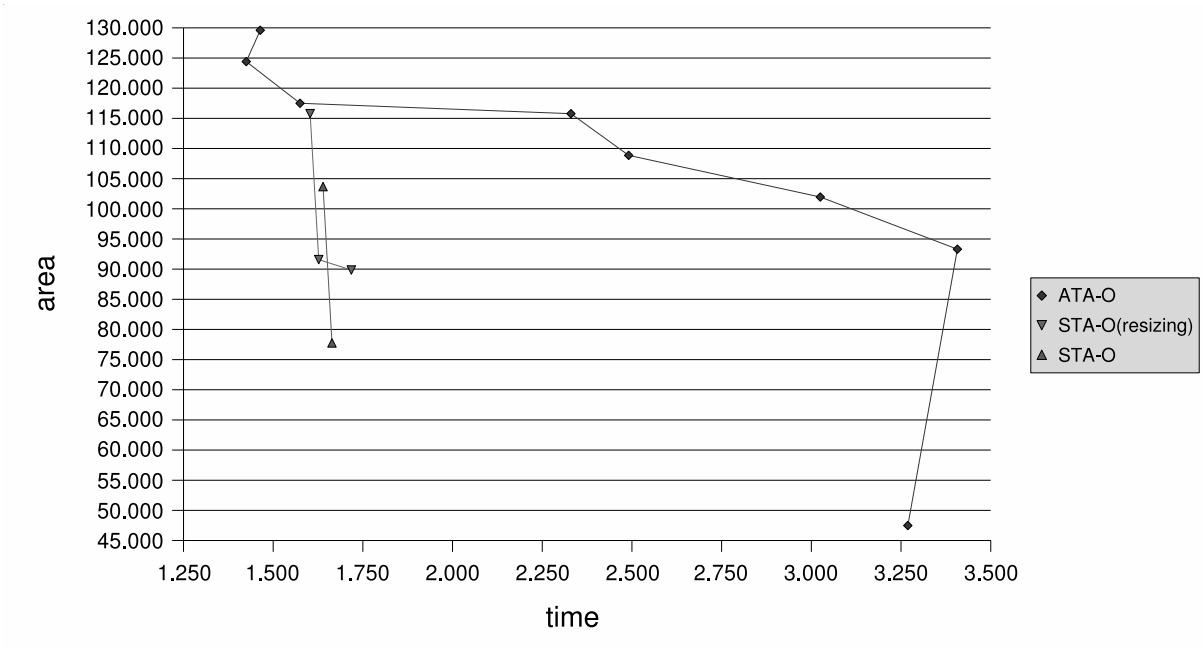

Figure 6.7: Time/area results of desynchronization controller in 1-scale ring.

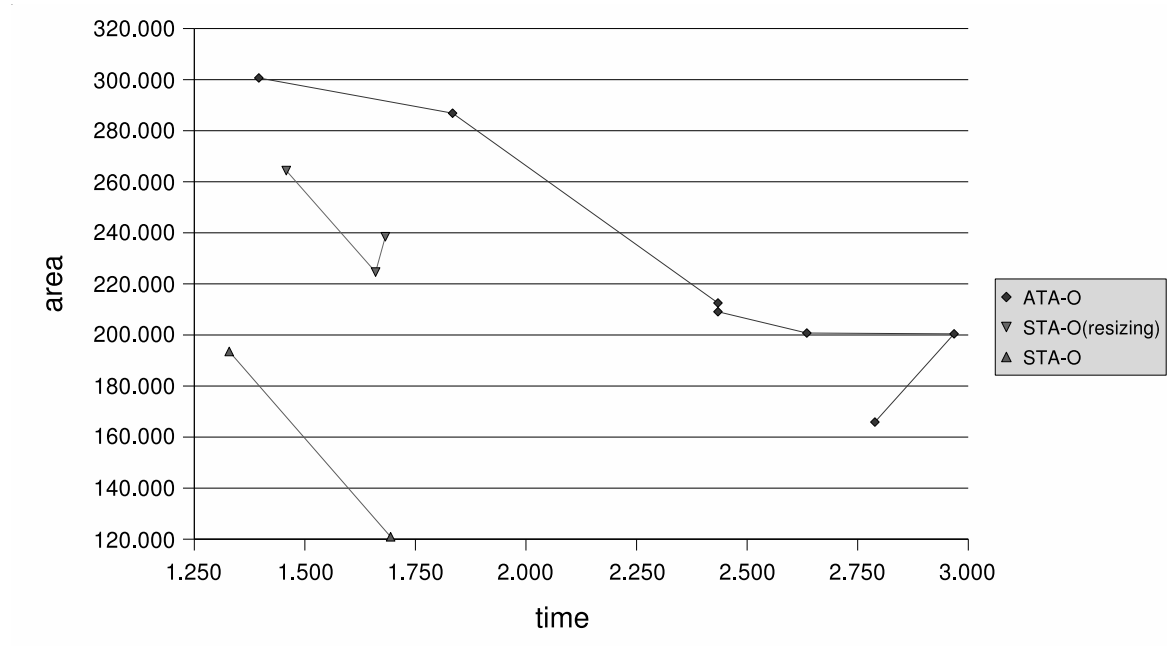

Figure 6.8: Time/area results of desynchronization controller in 3-scale ring.

formulation, and the edge to gate correspondence of the timing analysis. During the construction

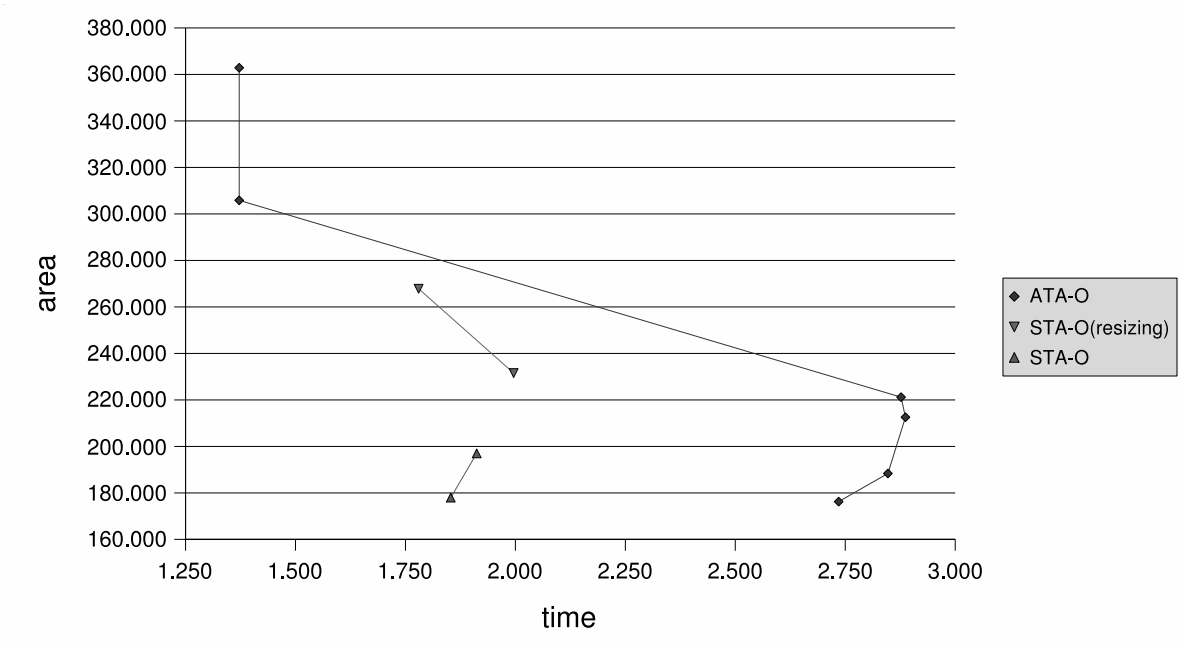

Figure 6.9: Time/area results of desynchronization controller in 3-scale ring, with wire delay.

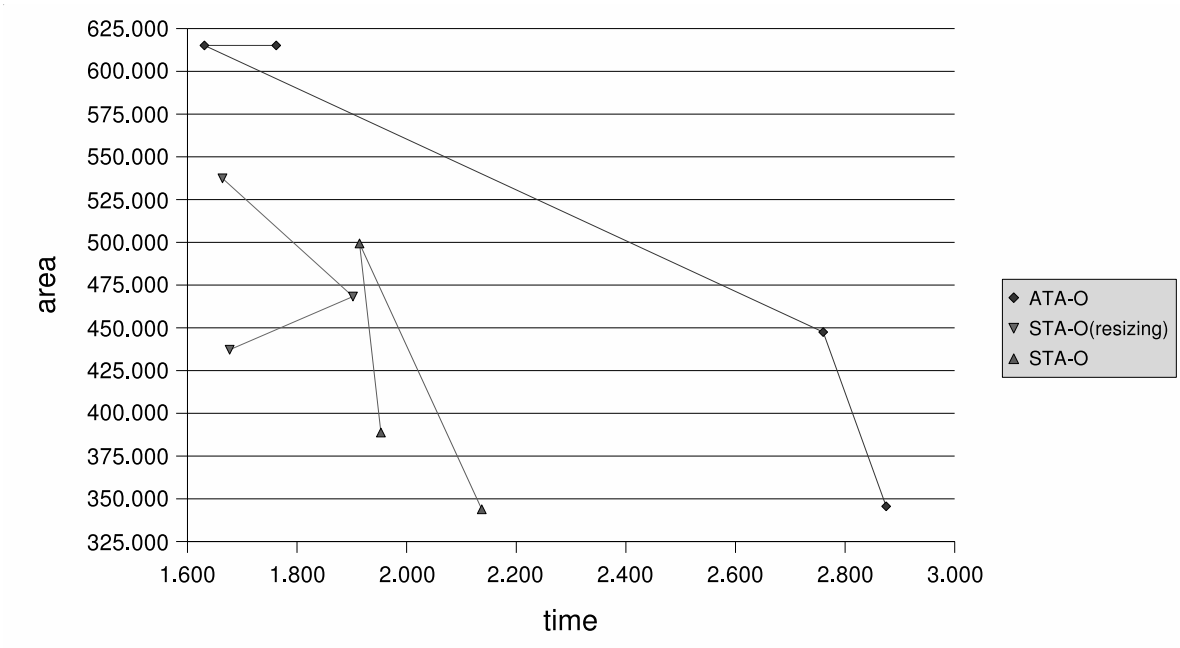

Figure 6.10: Time/area results of desynchronization controller in fork-join pipeline.

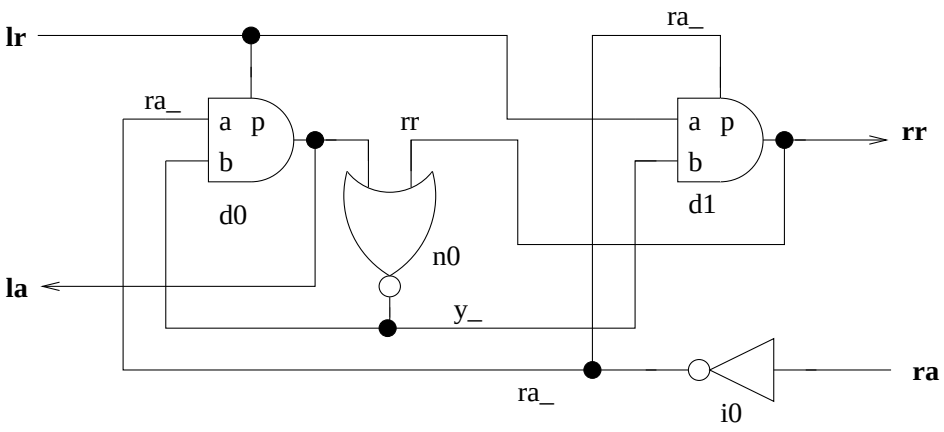

Figure 6.11: RTBM controller netlist in 1-scale ring topology

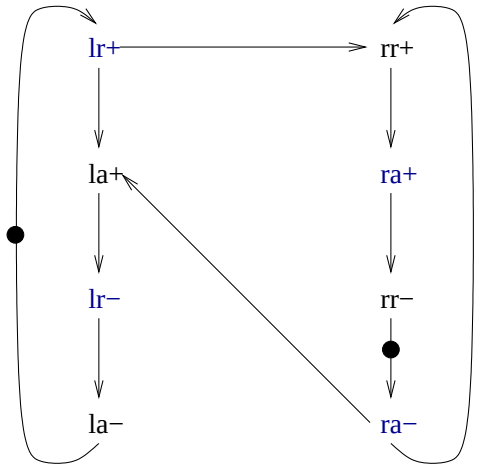

Figure 6.12: Signal Transition Graph of RTBM controller

of this STG, only the handshake signals were considered, *i.e.* lr, la, rr, ra. Thus, only timing dependencies between the transitions of these signals are accounted for in the STG. Moreover, the relation between STG edges and netlist gates is a defining factor for the optimization. If a netlist gate, is not included in the STG edges, *i.e.* it does not produce an STG signal, and is not included in the edge-gate relation, it is simply left out of the optimization process. In this specific example, the design's STG hides a timing assumption, at gate $n0$, as shown in Figure 6.11. Signal $y\_$ is not considered in STG formulation and gate $n0$ is, thus, not considered in the optimization. This might hide a set of optimized implementations and must be further explored.

Time/area results for all the netlists produced appear in the graphs of Figure 6.13 and Figure 6.14.

RTBM Controller			Simulation (ns) $Ri_+ \rightarrow Ri_+$	Area (μm^2)	Improvement %
1-scale ring	ATA	Minimum Area	$2.427ns(1.577ns)$	$84.672\mu m^2$	
		Critical Cycle Optimization	$1.246ns(0.8548ns)$	$120.96\mu m^2$	
		Gate Delay Optimization	$1.265ns(0.85ns)$	$127.872\mu m^2$	
	STA-resizing	Minimum Area	$1.294ns$	$91.584\mu m^2$	
		Path OptimizationA	$1.242ns$	$96.768\mu m^2$	**-0.3 %**
		Path OptimizationB	$1.242ns$	$96.768\mu m^2$	
	STA-resynthesis	Minimum Area	$1.368ns$	$88.128\mu m^2$	
		Path OptimizationA	FAILED	$55.296\mu m^2$	**-27 %**
		Path OptimizationB	$0.974ns$	$76.032\mu m^2$	
4-scale ring	ATA	Minimum Area	$4.155ns(2.049ns)$	$307.584\mu m^2$	
		Critical Cycle Optimization	$1.265ns(1.251ns)$	$423.36\mu m^2$	
		Gate Delay Optimization	$1.237ns(1.223ns)$	$454.464\mu m^2$	
	STA-resizing	Minimum Area	$2.119ns$	$331.776\mu m^2$	
		Path OptimizationA	$2ns$	$381.888\mu m^2$	**3 %**
		Path OptimizationB	$1.304ns$	$374.976\mu m^2$	
	STA-resynthesis	Minimum Area	FAILED	$283.392\mu m^2$	
		Path OptimizationA	$1.758ns$	$395.712\mu m^2$	**14 %**
		Path OptimizationB	$1.514ns$	$485.568\mu m^2$	

Table 6.3: RTBM controller in 1-scale and 4-scale ring results.

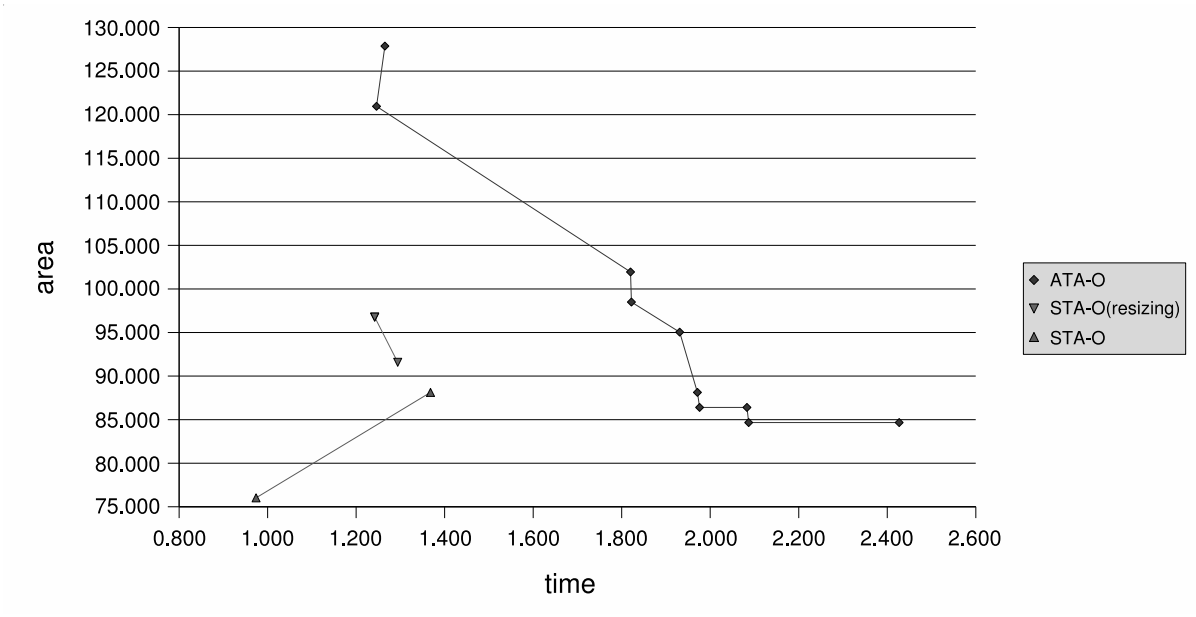

Figure 6.13: Time/area results of RTBM controller in 1-scale ring.

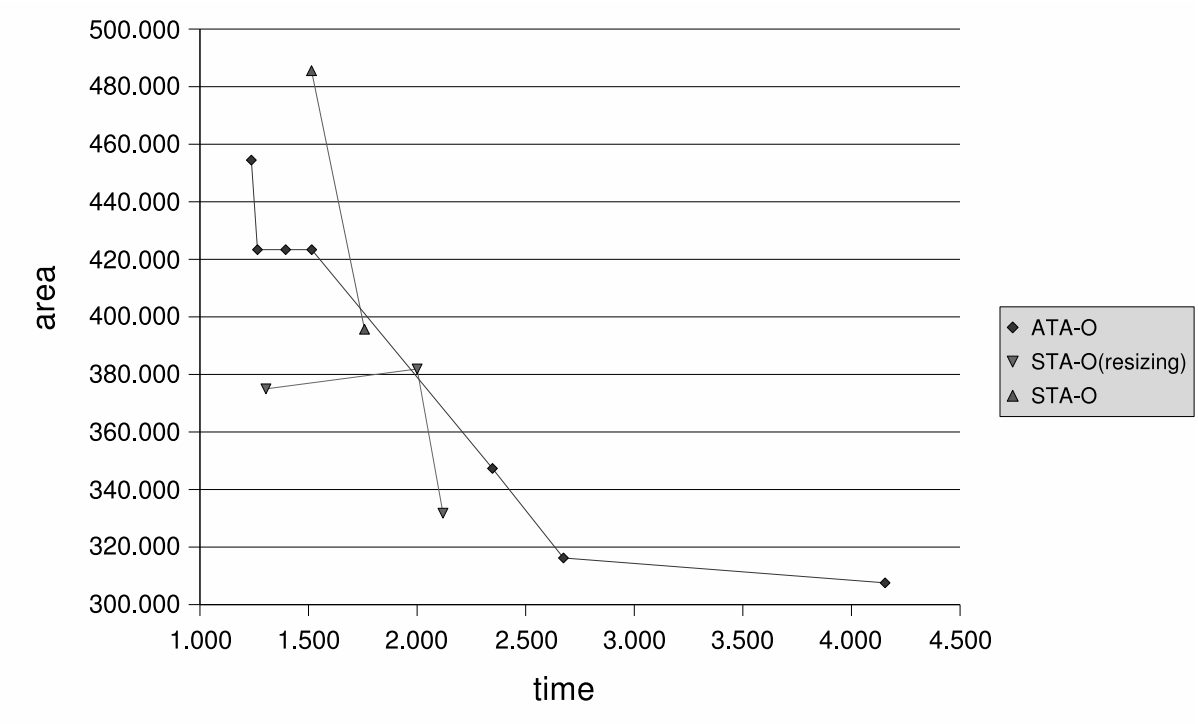

Figure 6.14: Time/area results of RTBM controller in 4-scale ring.

6.4 C-Muller Pipeline

C-Muller Pipeline is a pipeline of controllers which synchronize stages of pipelined logic [11]. Each controller is implemented as a C-Muller gate. A simple implementation of a C-Muller gate appears in Figure 6.15. Each controller produces a request signal for the next stage which is also an acknowledgement signal for the previous stage. A pipeline of four controllers was studied which were connected as shown in Figure 6.16. The results of the experimental procedure are shown in Table 6.4.

The results of this design follow the main points noticed on the previous designs. Graphs of time/area measurements of all the netlists produced through optimizations appear in Figure 6.17.

6.5 VME Bus controller

Except from latch controllers, another control circuit was studied, *i.e.* a VME bus controller, which controls read and write operations through a bus. The protocol followed can be described with an STG with choice. However, a read-write operation imply choice, which was not supported in this work. Thus, only the read operation was examined here. The write operation can be analyzed by using a separate STG. The block digram and the STG that describes the interface protocol for a read

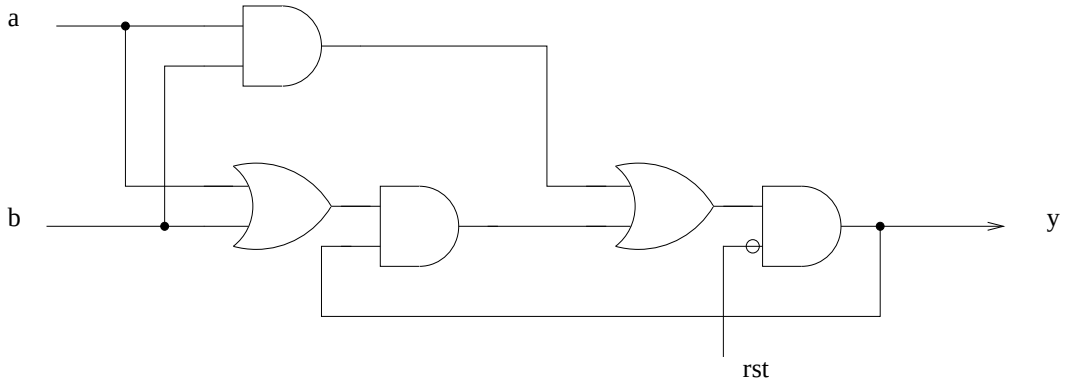

Figure 6.15: C-muller gate implementation

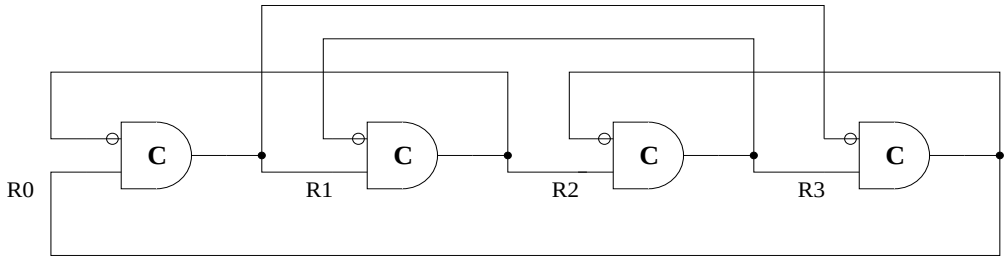

Figure 6.16: C-muller pipeline

C-Muller Pipeline			**Simulation** (ns) $Ri_+ \rightarrow Ri_+$	**Area** (μm^2)	**Improvement %**
4-scale ring	**ATA**	Minimum Area	$2.929ns(1.354ns)$	$134.784\mu m^2$	
		Critical Cycle Optimization	$1.73ns(1.134ns)$	$186.624\mu m^2$	
		Gate Delay Optimization	$1.641ns(1.131ns)$	$196.992\mu m^2$	
	STA-resizing	Minimum Area	$1.714ns$	$141.696\mu m^2$	
		Path OptimizationA	$2.068ns$	$176.256\mu m^2$	**-1 %**
		Path OptimizationB	$1.712ns$	$141.696\mu m^2$	
	STA-resynthesis	Minimum Area	$2.765ns$	$139.968\mu m^2$	
		Path OptimizationA	$2.194ns$	$138.688\mu m^2$	**13 %**
		Path OptimizationB	$1.985ns$	$139.968\mu m^2$	

Table 6.4: C-Muller pipeline results

operation is shown in Figure 6.18. The results for this design are shown in Table 6.4.

In this design the timing results of the critical cycles optimization were the same as the ones of gate delay optimization and they were slightly better than the ones produced by the conventional

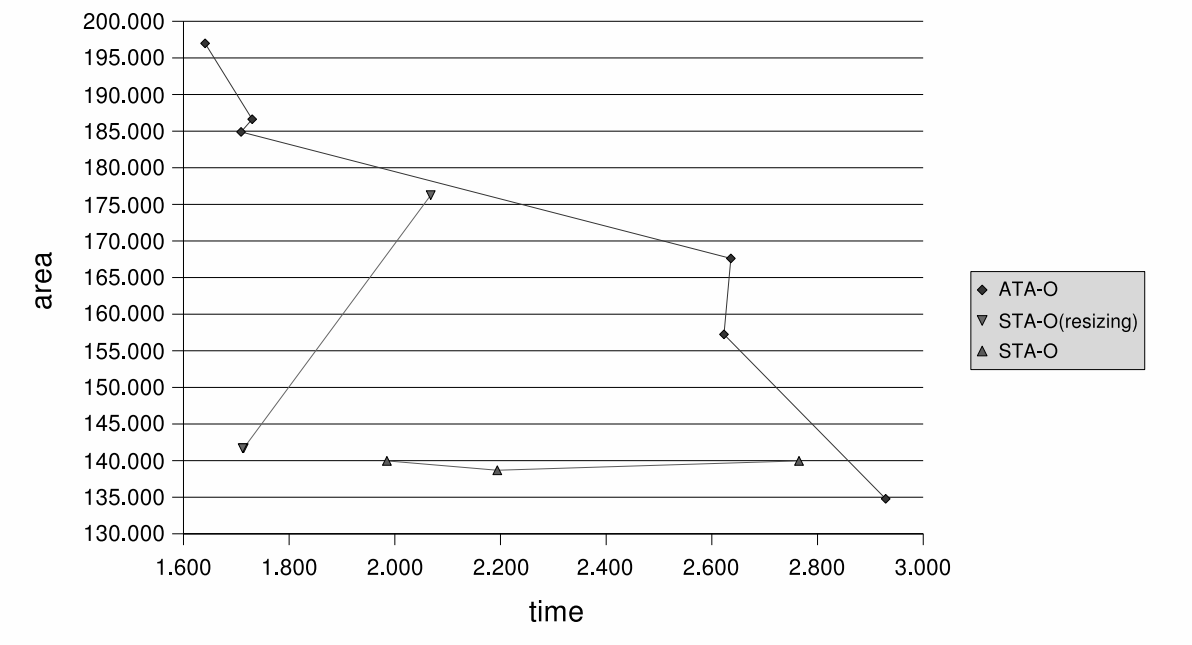

Figure 6.17: Time/area results of a 4-scale c-Muller pipeline.

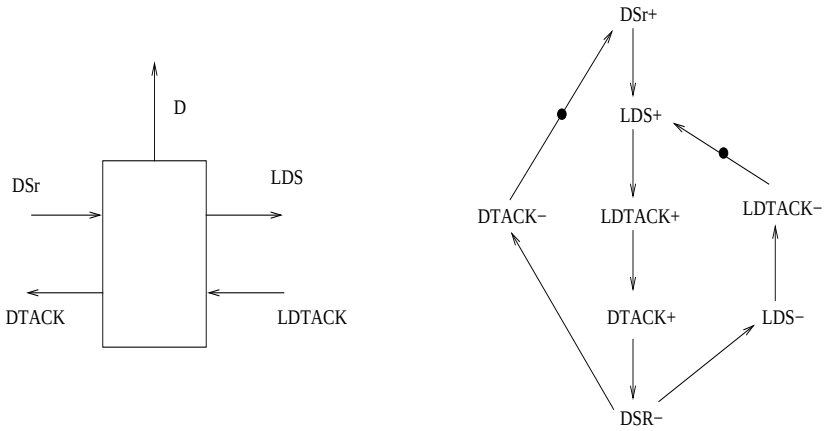

Figure 6.18: STG of a read operation in a vme bus controller

optimization through resizing. The netlist of this circuit is small and each signal is determined by one or two gates as it is indicated by the following equations:

$$LDS = D + CSC$$

$$DTACK = D$$

$$D = LDTACK \cdot CSC$$

VME Bus Controller			**Simulation** (ns) $Ri_+ \rightarrow Ri_+$	**Area** (μm^2)	**Improvement %**
4-scale ring	ATA	Minimum Area	$1.253ns(0.92ns)$	$31.104\mu m^2$	
		Critical Cycle Optimization	$0.734ns(0.777ns)$	$43.2\mu m^2$	
		Gate Delay Optimization	$0.734ns(0.777ns)$	$43.2\mu m^2$	
	STA-resizing	Minimum Area	$0.806ns$	$31.104\mu m^2$	
		Path OptimizationA	$0.745ns$	$39.744\mu m^2$	**1 %**
		Path OptimizationB	$0.745ns$	$39.744\mu m^2$	
	STA-resynthesis	Minimum Area	$0.762ns$	$32.832\mu m^2$	
		Path OptimizationA	$0.349ns$	$50.112\mu m^2$	**-61 %**
		Path OptimizationB	$0.454ns$	$48.384\mu m^2$	

Table 6.5: VME Bus controller results

$$CSC = DSR \cdot (CSC + \overline{LDTACK})$$

The specific design doesn't contain many cycles dependencies, thus, STA optimization can be effective as well as ATA optimization. Figure 6.19 shows the timing/area results of all the netlists produced.

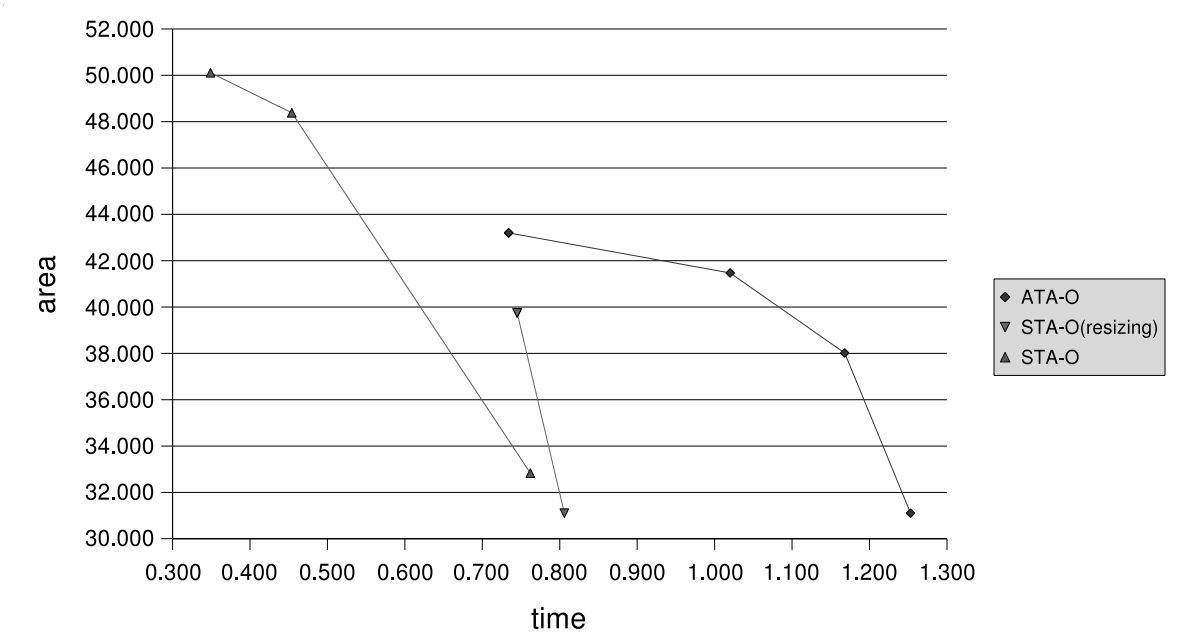

Figure 6.19: Time/area results of a VME Bus Controller.

7

Conclusions

This work has shown that Asynchronous Timing Analysis (ATA) and optimization through a complete EDA flow for asynchronous circuits is feasible. Prior work on an algorithm, evaluating exact bounds on the Timing Separation of Events has been clarified and completed. Theoretical issues have been examined, as well as application and complexity issues. The gap between theory and implementation was covered successfully, achieving the development of a complete EDA tool for the timing analysis of asynchronous circuits. A well-defined flow for the application of the ATA tool was proposed for the optimization of asynchronous circuits and for validation of RTC. The optimization flow was implemented within an EDA optimization tool, which was used for the optimization of several asynchronous designs. The implemented and optimized through the ATA netlists were compared with implementations derived from contemporary flows of synchronous design. The results proved the proposed flow viable for exploring various implementations.

Future work mainly consists of extentions on the ATA algorithm and improvements for the EDA optimization flow. Additionally to delay ranges, probabilistic functions may be associated with delay constraints. This would be a step towards incorporation of statistical information in the ATA tool, *i.e.* for the development of a Statistical ATA engine, and with respect to the ATA algorithm internals, the direction of Symbolic Timing Analysis may be persued. Additional work is required for the EDA flow

to handle circuit netlists automatically.Further examination is needed in the field of STG functionality and incorporation of choice in specification. The generation of Petri-Nets, which are now required to include choice, as Free-Choice Petri-Nets, require further investigation. Moreover, the optimization functionality can be further extended. Full technology-mapping functionality may be implemented, as well as full HDL parsing and library coverage. Finally, more technology attributes, such as transition times, can be included in the analysis to implement a more accurate EDA tool, able to better explore the implementation space of asynchronous circuits.

Bibliography

[1] Joep Kessels, Torsten Kramer, Ad Peeters, and Volker Timm. DESCALE: a design experiment for a smart card application consuming low energy. In Jens Sparsø and Steve Furber, editors, *Principles of Asynchronous Circuit Design: A Systems Perspective*, chapter 13. Kluwer Academic Publishers, 2001.

[2] T. Murata. Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, pages 541–580, April 1989.

[3] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis of Asynchronous Controllers and Interfaces*. Springer-Verlag, 2002.

[4] Ken Stevens, Ran Ginosar, and Shai Rotem. Relative timing. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 208–218, April 1999.

[5] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello. An algorithm for exact bounds on the time separation of events in concurrent systems. *IEEE Transactions on Computers*, 44(11):1306–1317, November 1995.

[6] Henrik Hulgaard, Steven M. Burns, and Gaetano Borriello. Testing asynchronous circuits: A survey. *Integration, the VLSI journal*, 19(3):111–131, November 1995.

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

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

[9] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[10] Ken Stevens. Personsal contact.

[11] Ivan E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, June 1989.