Performance Optimization of Elastic Systems

Dmitry Bufistov

Department of Software (LSI)
Technical University of Catalonia (UPC)

September 2010

Acknowledgments

I would like to say thank you to people who helped and supported me during this work.

To Jordi Cortadella for providing opportunity to do a Ph.D. To my colleagues from the University and especially to Jorge Júlvez, Marc Galceran-Oms, and Josep Carmona for their great help not only in the research but also in the everyday life.

Many thanks to all administrative personal of the *Departamento de Lenguajes y Sistemas Informáticos* and especially to Merce Juan Badia who was always compeeting to be my first mother.

I am also very gratefull to the *Oficinas de Relaciones Internacionales* especially to Anna Fabrigas-Punti y Eulàlia Miñarro. They provide a real help to foreign students in this complicated bureaucratic environment.

And of course my last but not least thanks go to my parents, my brother *Костя* and my girlfriend Gladys for their constant encouragement support and love.

This work has been supported by a scholarship from the Spanish government (FPU grant AP2005-4866), a grant from Intel Corporation, research projects GRAMMARS (CICYT TIN 2004-07925-C03-01) and FORMALISM (CICYT TIN2007-66523).

Preface

Elastic systems (ES) offer a simple and elegant approach to tolerate variability in computation and communication delays. An ES is a set of computational nodes interconnected by communication channels. The main property of the computational nodes in an ES is that they synchronize their behavior by local hand-shake protocols. That is, the behavior of a particular node depends only on its neighbors. If some input data is not available, the node waits for it as long as necessary to perform a computation.

Most computational systems one can imagine are elastic: a computer program, a microprocessor (at the architectural level) or an asynchronous circuit are elastic systems, while conventional synchronous sequential circuits are not. In synchronous design it has to be guaranteed that all data are available at the end of the clock period. If some node fails to finish the computation, the result of the operation can be wrong. The advantage of the synchronous approach is that the design process is significantly simplified.

However, as nano-technology continues to scale down, the uncertainty of the delays increases and timing is becoming wire dominated. If a communication is required between two nodes that are situated far apart on the chip, it can take a long time. To meet the timing requirements for this communication, the cycle period may need to be increased, degrading the performance of the circuit.

One solution for this problem of synchronous design is to substitute the global clock signal by local hand-shakes, leading to asynchronous design. Asynchronous design is elastic and does not lose performance because of the clock accommodation for the worst case. However, contemporary Computer Aided Design (CAD) tools are mostly oriented to the synchronous design since it has predominated for more than twenty years.

A discrete version of the asynchronous design was recently proposed by several research groups. This work refers to a such design as Synchronous Elastic Design. Synchronous design is elasticized by associating a control information with the data path of the circuit. The control information indicates the validity of the data that is being transfered or computed and coordinates the flow of valid and non-valid data.

ESs accept a set of correct-by-construction transformations. Correct-by-construction means that the high-level behavior of the system is unchanged. Usually, such transformations can be performed in a totally automated way or with a minor intervention of the user. For example, in the ES the long interconnection wires can be pipelined with *empty registers* (*bubbles*), which is not an easy task in classical synchronous design. Also pipelining of an ES can be a totally automated process.

There is a lot of investigation have been done in this area. However, most of the work is focused either on design aspects or on a local transformation that can potentially improve performance of an ES and there are quite few publications about global performance optimization of ESs using proposed local transformations. The main goal of this work is to show how performance of an ES can be estimated and enhanced using a set of correct-by-construction transformations.

This work uses a simple abstract model to represent an ES. This model is a graph with several properties attached to its vertices and edges. In the very beginning the modeling was done with only synchronous systems in mind, but it looks like asynchronous circuits would also benefit from the proposed optimization techniques.

The first contribution of this work combines *retiming* (a well known sequential optimization technique) with *recycling* (insertion of bubbles) to optimize the performance of ESs. An optimization technique is proposed based on mixed integer linear programming. For this, the performance of ESs is modeled with a set of linear inequalities. The experimental results showed that the simultaneous application of both techniques allows achieve better performance/area trade-offs that their sequential application. However, the cycle time of the circuit could be hardly reduced beyond one retiming achieves and never beyond the cycle time that clock skew optimization provides.

There are situations when a computational node of an ES may produce new valid data having only some valid inputs. A classical example is a multiplexer which selects only one branch depending on the value of the select signal, the other branches are don't care. A special evaluation rule which is called *early evaluation* may improve the global performance of an ES when applied to multiplexer like nodes. The second contribution extends the first to the ESs with early evaluation nodes. The extension is based on the marked graph model for systems with early evaluation nodes (guarded marked graphs). The retiming and recycling applied to ESs with early evaluation nodes may improve its performance far beyond both retiming and clock skew optimization techniques.

The third contribution can be viewed as a synthesis of *slack matching* and *buffer sizing*. Slack matching is a well-known problem in asynchronous design. Slack matching can be viewed as bubble insertion (recycling), but in the way the overall throughput of an ES increases. Buffer sizing has been proposed for performance optimization of synchronous elastic systems. Buffer sizing finds the minimal capacities of each communication buffer of an ES, such that the overall performance does not degrade because of backward stall event propagation. An abstract model of ES is used without differentiating synchronous and asynchronous implementation. Moreover, the first and the second contributions make an assumption that all buffers of the ES have infinite capacity. The third contribution shows how the reasonable sizes for these buffers can be found. Even, in general case, being NP-hard problems, slack-matching and buffer sizing optimization techniques can be applied within a few seconds to the major part of ESs.

Contents

Chapter 1

Introduction

We were living in the continuous world when physicists discovered that the space and the time are discrete and every physical object behaves both like a particle and like a wave. Fortunately, the granularity of the discontinuity of the world has magnitude of the Planck's constant [1] that still makes us feel and think continuously.

The very first electronic digital devices appeared only in the second part of the XX century. Usually, the conversion from continuous structure to discrete makes systems more amenable for analysis, synthesis and verification, but we should bear in mind the integer linear programming [81] which is NP-complete [83] while linear programming has polynomial time solutions [62, 64].

In the last few decades the tremendous progress in the technology of digital circuits manufacturing shook the world. The supercomputer of the 60s is now the property of everybody who have a mobile phone. Microprocessors appeared on the market in the beginning of the 80s and during thirty years the Moore's Law [2] has been holding true so precisely that it makes us to suspect that it is a pure marketing trick of the manufacturers.

Nowadays there is a huge industry of digital circuits manufacturing with a lot of interesting and complex problems. This work is concentrated on some of them. The main contributions of this work are formal methods for performance analysis and optimization of elastic (a.k.a. latency insensitive) systems. Let us start outlining famous problems of the nano-technology that motivate this research.

[1] $h = 6.626068 \cdot 10^{-34} \frac{m^2 \cdot kg}{s}$

[2] Microchips will gain a factor of two in speed and the number of transistors will increase by two every 2 years

1.1 Motivation

The very first digital circuits did not use the global synchronizing signal, also known as *clock*. But it turn out that the introduction of clock simplifies design process and still allows to produce good, from the performance point of view, circuits. This observation resulted in a huge industry of the synchronous circuits manufacturing. Every two years the manufactures would use the advances of the *photo-lithography* to make circuits twice faster.

However, it is easy to observe that in the last years the clock period of the popular microprocessors has been decreasing at a much lower rate than in the previous decades. New problems arise in the 65nm and deeper nano-technologies, namely:

- Interconnection delays [1]. The performance of the modern microprocessors is becoming dominated by the interconnections (wires). The traditional assumption that any two modules of the circuit may communicate in a negligible time is no longer valid. The part of the chip that can be accessed from the given cell of the circuit within one clock cycle is decreasing as technology continues to scale [75]. The crosstalk and coupling capacitances of interconnections start to play a significant role in the circuit timing. The inadmissibility of long wires make clock tree synthesis more complicated since wires/gates sizing and buffers insertion techniques must be applied at this step [44, 69].

  Another consequence of the performance being dominated by wires is the increasing gap between timing estimation at the front and back ends of the design process. A perfectly feasible design at the logical level may become infeasible after detailed physical. The delay of interconnections can be reduced by interconnect optimization techniques such as wire/gates sizing [30], buffer insertion [80, 100] and gate duplication [29, 103], but the gap between delays of interconnects and devices continues to grow [35].

- Design variability [82]. It is one of the leading technical challenges that impede continued technology scaling. There are two type of variability that are usually deserving attention: variability between chips and variability within a chip. The variability between chips appears because of imperfections in photo-lithography technology. Some parameters can not be precisely controlled at nano-meter nodes which makes different chips to have different timing and consequently, different performance. On-chip variability (OCV) comes from the fact that a cell behaves differently (has different timing) depending on the operating conditions of the chip. The temperature, the level of the high voltage affect the timing of the circuit. The only thing the traditional synchronous methodology can do to cope with OCV is to increase the clock cycle, hence, loosing performance. There are *voltage scaling* [85] techniques that allow to adapt circuit to operating conditions of the chip, process workload etc.

- Design complexity. Again, as the Moore's law predicts the contemporaneous microprocessors are approaching to billion transistors integrated on a die. To build such complex systems withing strict time-to-market constraints designers use pre-designed macro-blocks (a.k.a. IPs) [16]. Of course, such approach simplifies the high level design process, but at the same time, the placer and the router tools have less flexibility. Hence, it is more difficult to achieve a good performance in such design methodology because good placement and routing are one of the determine factors of the final circuit performance.

- Power consumption and dissipation. Microprocessor power has become a first-order constraint at run-time. One of the consequences of the Moore's law is that transistor densities have increased exponentially over successive process technologies, but supply voltage has not decreased accordingly [88]. Even-though the optimization techniques proposed in this work do not have power consumption as a main optimization target the elastic design may help do build a more power efficient circuits [38].

All these problems convert the reduction of the synchronous clock period into a difficult task.

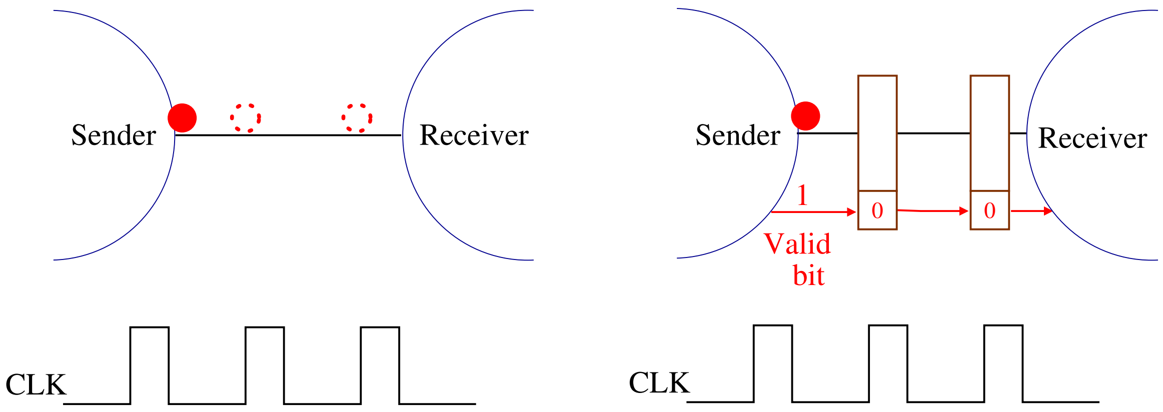


Figure 1.1: Pipelining of a long wire with two registers.

Now let us outline when and how the concept of elasticity may be useful for circuit designers. The low accuracy in the estimation of the delay of wires at the behavioral synthesis stage may lead at later design stages to the situation depicted in the left part of the Figure 1.1. The data item, represented as a solid dark dot, can not be transmitted to the receiving part within one clock cycle. This might lead to the malfunction of the circuit. A possible way to fix such a violation is to pipeline the long wire with several extra registers. The problem is that in the conventional synchronous design the insertion of new registers is a difficult task, since the insertion of the registers without considering global consistency affects circuit functionality and destroys the correctness of system behavior.

Here is where the elastic design may help, since it is always possible to insert empty registers (*bubbles*) into any place of an elastic system (ES). The picture on the right in Figure 1.1 illustrates this transformation. The *valid bit* is propagated by the control logic together with the data item. When the valid bit reaches the receiver, the receiver knows that the input register stores valid data and can start a new computation. Thus, the transformation of the bubble insertion, also known as *recycling* in the literature [16], allows to eliminate timing problems at later design stages. Notice, that this transformation requires from the user only indication of the place on the chip where the bubble should be located, the rest is a totally automatic process.

However, things are not so easy as they might appear at the first glance. Insertion of bubbles increases the latency of the circuit and decreases its throughput (the number of valid data it produces per unit of time). The contributions of this work show how this problem can be reduced or avoided.

Often a computational block can be optimized for a typical case. For example, the addition of integer numbers with small significant bits does not require the carry bit to be propagated to the highest bits [31]. Using this idea, *variable latency* or *telescopic* blocks can be designed [3, 6]. The increasing OCV further motivates creation of such blocks. In an ES such blocks can be handled gracefully, while in the classical synchronous design a complicated scheduler is required.

The effective reuse of existing intellectual-property design components (also known as IP cores) is essential to meet the challenges posed by nano-technologies and to complete a reliable design within time-to-market constraints. An IP core can be as complex as a CPU or an RAM. Using this approach a *system-on-chip* (SoC) can be built. The simplicity of the approach at the behavior level leads to the timing problems at the physical level of the design. Again elastic design rescues designers: an arbitrary number of bubbles can be added to the long wires while preserving sequential behavior of the circuit. This allows to break long interconnections between IPs thus deriving a feasible design [16]. But, the throughput degradation may remove all performance gains in timing. Chapters 4, 5 and 6 address this problem.

Today, pipelining is the key implementation technique to make fast CPUs. In a pipelined system each instruction needs several pipeline stages to be executed, but several instructions can be executed simultaneously. Pipelining may increase the performance of the system. The unacceptability of long wires further motivates the development of pipelined architectures. However, building a good pipeline from a non-pipelined design is a challenging task [56](chapter 6), while the pipelining of the elastic design can be automated [61]. All these potential benefits of the elastic design motivated us to investigate it in depth using simple abstract models. On the other hand, there are a lot of publications in the area of elastic design itself, but there are

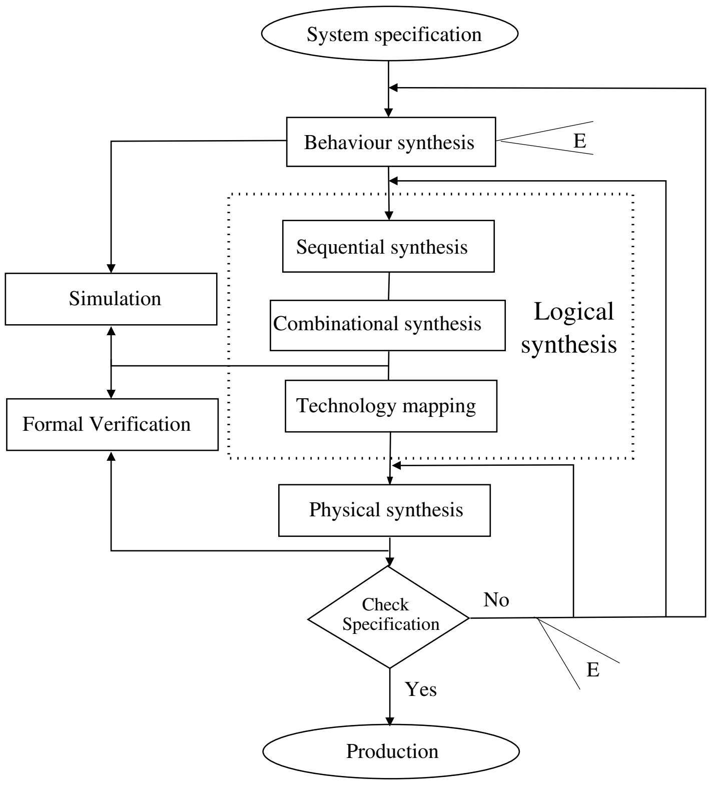


Figure 1.2: The path to silicon.

very few about the global performance optimization of elastic design.

1.2 The path to silicon

The manufacturing process of an integrated circuit (IC) is quite complex. Fortunately a big part of this process is automated. There are a lot of computer aided design (CAD) tools that aim at facilitating the design process. The following abstraction levels usually are selected in this process.

- **Behavioral Level**. At this level the design is seen as a black box; the relations between outputs and inputs are given without a reference to the implementation of these relations. Here, the designer describes a circuit in high-level hardware description languages (HDL) such as VHDL, Verilog [68] or SystemC [105].
- **Logical (Structural) Level**. At this level a circuit is seen as a composition of sub-circuits. The description here gives information on the sub-circuits used and the way they are interconnected. Usually this representation is called *net-list*. Net-lists are constructed from *gates*. Logical synthesis includes sequential

synthesis, combinational synthesis and technology mapping. As transistors size decreases the optimizations at logical level without considering physical information is becoming less accurate. There are a lot of good books about logical synthesis, see for example [52, 77].

- **Physical Level**. The physical synthesis translates the net-list to the physical devices (transistors). The main steps in this stage are floor planning, power synthesis, clock tree synthesis (if necessary), placement and routing. Usually, the design process is not terminated at the first iteration. The final design may not fulfill completely the initial specifications (cycle time) or physical specifications (congestion, power consumption). Hence, previous stages must be repeated until overall specifications are satisfied. The book [52] also address many aspects of physical synthesis.

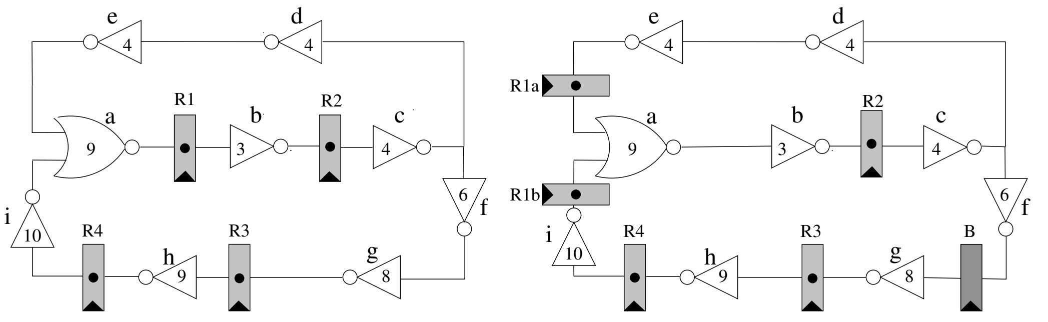
Figure 1.2 shows a basic design flow. Simulation and formal verification are two important steps that can be applied at any stage.

The elastization of the synchronous design can be done at the behavioral level or after the first iteration of the physical synthesis. The corresponding places are marked with the letter E in the figure. To do so, the design must be partitioned into blocks and a control logic automatically generated. After this, a set of correct-by-construction transformations can be applied to explore different design trade-offs. The main problem of applying elastic transformations at the behavioral level is that the timing estimation may significantly differ from the timing of the final layout. But, this is the problem of the most design methodologies.

The elastization after placement or routing does not have the problem of the first approach. The timing characterization of the design is very close to the final timing. At this point the designer may partition design to blocks, leaving long wires as interconnections between these blocks. Then, critical wires can be pipelined with special sequential elements to meet the timing constraints. This approach is challenging because some constraints on the placement of the added elements must be satisfied. More control logic needs more interconnections which may violate wire congestion and make the whole circuit malfunctioning. These issues may be partially addressed in the CAD tools. This discussion deserves a separate investigation and it is left out of the scope of this work. Another drawback of the elastization at later design stages is that not all transformations can be applied with placement and routing constraints. For example, to use variable latency blocks in the elastic design, only the first approach is acceptable.

1.3 Objectives of the work.

The main objectives of the work are the following:



(a) Simple elastic circuit (b) After retiming and recycling optimization

Figure 1.3: Illustration of the retiming and recycling elastic transformations.

- *Investigate potential performance benefits that can be achieved using the concept of elasticity in the circuit design.*
- *Develop formal methods for performance analysis and optimization of elastic systems.*
- *Adapt existing correct-by-construction transformations and techniques, such as retiming, early evaluation/variable-latency nodes to elastic design.*

This work models ESs as a weighted directed graph. The vertices and the edges of the graph have several properties attached. In the most cases the strongly connectivity of the graph is a natural assumption, but in the all proposed techniques this is not a requirement.

We use Linear Programming [81], Markov chains [109] and Petri nets [79, 99] for analysis and performance optimization of ESs. Chapter 2 provides a brief introduction to these fields, more details on the background can be found in the referred literature.

1.4 Retiming and recycling

Elastic design accepts a set of correct-by-construction transformations. Let us exemplify retiming and recycling transformations, since Chapters 4 and 5 are devoted to these optimization techniques.

Consider a simple elastic circuit depicted in Figure 1.3(a). The circuit has nine gates and four registers. The gates are denoted with lower case letters, $a, b, \ldots, i$ while the registers are labeled as R_1, R_2, R_3, R_4. Notice, that all registers are marked with a black dot. This means that all of them store some valid data. This dot helps to distinguish valid data (tokens) from non-valid (bubbles). Each gate is labeled with the corresponding combinational delay. The control module of the circuit is not shown in the figure, since we are interested only in its data-path. The longest combinational path of the circuit (*critical path*) is formed by the nodes c, d, e and a ($c \rightarrow d \rightarrow e \rightarrow a$); its total delay is equal to $4+4+4+9=21$ units of time. Thus, 21 can be used to estimate the performance of the circuit.

Retiming, a classical optimization technique of synchronous design, can also be applied to elastic design. Retiming moves registers across combinational nodes of the circuit, changing its critical paths while preserving sequential behavior. In the example, one can move register $R1$ backward across node a. This leads to the configuration with two registers $R1a$ and $R1b$, as it shows Figure 1.3(b). The critical path of this configuration would be formed by the nodes c, f, g, thus providing performance measure of 18 units of time.

Figure 1.3(b) shows an optimized version of the circuit after applying both retiming and recycling transformations. The following transformations were performed on the original circuit: retiming register $R1$ backward across combinational block a and placed to the inputs of this node, resulting in registers $R1a$ and $R1b$; putting a bubble, denoted as B, to further break long combinational paths. It can be verified that now the critical path has delay of 12 units of time. There are two paths with such delay: $a \rightarrow b$ and $c \rightarrow d \rightarrow e$. On the other hand, the resulting circuit contains a bubble and can not always produce valid data. The highest rate the valid data can be provided at is equal to 4/5, because the bottom most directed cycle has five registers and four tokens. Thus, the throughput of this circuit can be estimated as 4/5.

The question to answer is how the performance of the initial and optimized circuits can be compared? The initial circuit has a critical path with a delay of 21 units of time but can produce valid data at the rate of one. The optimized circuit has a critical path with a delay of 12 units of time. Does it mean that this circuit is twice better than the original one? No, it is not. To obtain a correct performance metrics we should divide the delay of the critical path by the circuit throughput. This value is called *effective cycle time* and it is used as the main metrics for evaluating the performance of elastic circuits. Thus, the effective cycle time of the optimized circuit is equal to $12/0.8 = 15$ units of time, 29% performance improvement comparing with the effective cycle time of the original circuit, remember (21).

One of the main contributions of this work is an algorithm to apply retiming and recycling to achieve the best possible performance of elastic circuit. The proposed method concentrates on the data-path of the circuit and ignores details of the implementation. Thus, it is independent on the control module and can be potentially applied to different implementations. Chapters 4 and 5 are devoted to this optimization technique.

1.5 Organization of the document

This document is organized as follows:

- **Chapter 2** provides a basic overview of the sequential circuits and the mathematical concepts that are used in this work. They are: linear and mixed integer linear programming, Markov chains and Petri nets. Even though the reader might be familiar with all these concepts it is a good idea to browse this chapter to become familiar with the notation used in this work and to refresh the basic definitions.

- **Chapter 3** inspects the fields of investigation this work is based on or related to. They are: retiming, a classical sequential optimization technique; latency insensitive design; buffer sizing and slack matching optimization techniques; static scheduling of discrete event systems.

- **Chapter 4** presents a general model for sequential performance optimization. This model combines retiming with recycling and also can be used for c-slow retiming. It can be also extended to handle early evaluation and variable latency nodes.

- **Chapter 5** extends the optimization model from the previous chapter to handle early evaluation. In this chapter a multi-guarded Petri net model is constructed to estimate the throughput of the ESs with both late and early evaluation nodes.

- **Chapter 6** is about optimal sizes of buffers of the control module of an ES. The retiming and recycling optimization technique relies on the fact that these sizes do not constrain the performance of an ES. Thus, after applying retiming and recycling the sizes of the buffers have to be calculated by using the technique proposed in this chapter.

- **Chapter 7** makes final conclusions and indicates possible directions for the future investigations.

Chapter 2

Preliminaries

This chapter starts with an introduction to elastic circuits. The main differences between synchronous and asynchronous implementations are outlined. Next, linear and mixed integer linear programming optimization techniques are introduced. Linear programming (LP) is a power optimization method. There are efficient algorithms to solve large scale LP problems. This work extensively uses LP for performance analysis and optimization of ESs.

Afterwards, the LP framework for a directed graph cycle ratio problem is presented. The cycle ratio problem is closely related to the performance evaluation of ESs and provides a convenient framework for the optimization. The three contributions of this work use this framework.

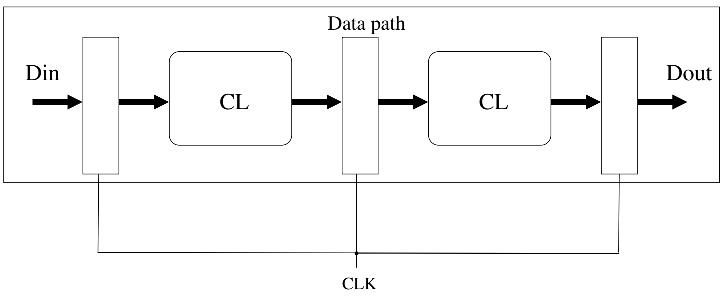
Finally in this chapter, an introduction to Petri nets follows. Marked graphs is an important sub-class of Petri nets. It is widely used for the modeling of discrete concurrent systems. An ES is an example of such a system. *A multi guarded Petri net* model is an extension of Petri net to model computations with early evaluation. Chapter 5 uses this model to optimize the performance of ESs with early evaluation nodes.

2.1 Elastic circuits

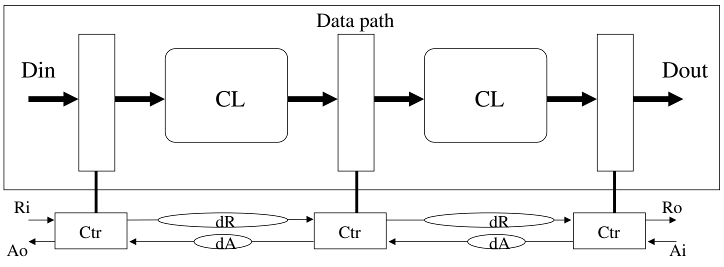
This section provides a brief overview of basic concepts related to synchronous and asynchronous elastic circuits. Please, refer to [20] for more details. Elastic circuits are sequential circuits, hence let us first introduce them.

2.1.1 Sequential circuits

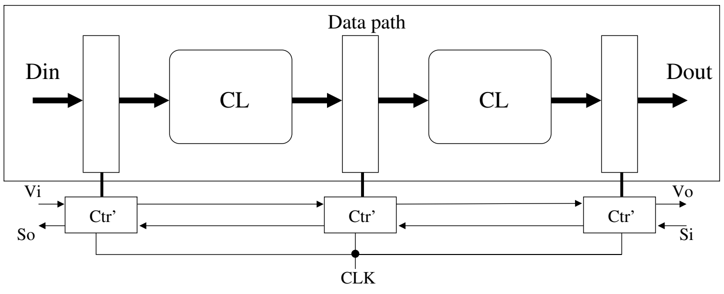
A sequential circuit is a net of registers and gates interconnected via wires. In a sequential circuit the output at any given time depends not only on the input at that



(a) Synchronous pipeline

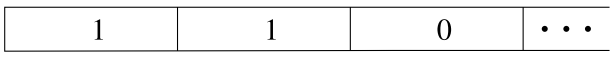


(b) Asynchronous pipeline

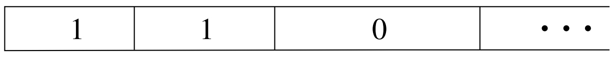


(c) Synchronous elastic pipeline

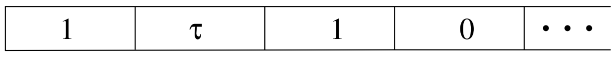
Figure 2.1: Synchronous, asynchronous and synchronous elastic circuits



(a) Synchronous output



(b) Asynchronous output



(c) Synchronous elastic output

Figure 2.2: Sequential circuits. Timing.

time but also on inputs at previous moments of time. A sequential circuit can be *synchronous*, if it depends on the global synchronizing clock signal, or *asynchronous*, if its data are locally synchronizing by means of hand-shake control signals.

Synchronous design

The synchronous approach significantly simplifies design tasks as it enables the use of zero delay abstractions for computation and communication delays. As we already know, one of the drawbacks of the synchronous approach is that the performance of the circuit, global clock cycle, is determined by its slowest part.

Figure 2.1(a) gives an example of a synchronous sequential circuit with two combinational blocks, denoted as CL and three registers, drawn as rectangles. The global synchronizing signal, denoted as CLK, arrives at all registers. Every time the clock signal is enabled, the registers capture their input data. Thus, the result computed by the first combinational block during the previous clock cycle becomes an input for the second combinational block when the clock signal arrives again. The amount of time between two consecutive clock events (*cycle time*) must allow each combinational block to recalculate its output value.

Figure 2.2(a) shows a possible sequence of values of the output signal *Dout* during three clock cycles. The width of each rectangle is a constant value and it is equal to the time separations between two consecutive clocks arriving.

Asynchronous design

In this section a very short overview of the asynchronous design is provided, since the performance optimization methods presented in Chapters 4, 5, and 6 can be applicable to asynchronous circuits as well. There are several books devoted to asynchronous design, see, for example [5, 40, 102].

Asynchronous circuits were first studied in the 50's of the previous century [78, 107]. Actually, the very first digital circuits were asynchronous and not synchronous. Asynchronous circuits have no global synchronizing clock, instead, the computational blocks are locally synchronized with their neighbors using a pair of handshake signals, usually being referred as *request/acknowledge*. The request signal is used by the sender to initiate an action. When the action is completed, the corresponding acknowledge signal is asserted by the receiver. These handshake signals are independent of any global system timing and are only concerned with the local relative temporal relationships between two subsystems sharing a common interface [43].

Figure 2.1(b) illustrates three stages of the asynchronous implementation of the same pipeline. When input environment provides valid input Din it sets input request (R_i) signal to one. Then, the pipeline starts its computation and when the computation is finished, the output request signal Ro is set to one as well. The ellipses dR and dA represent delay lines. Delay lines are combinational blocks of the control logic which mimics the timing behavior of the data path.

Figure 2.2(b) shows the values of the output signal $Dout$ of the asynchronous implementation. One may observe that the first two output values are calculated faster than in the synchronous version and only for the third value the circuit takes longer. It took even longer to calculate third value than the cycle time of the synchronous version.

Not everything is so cloudless in the asynchronous methodology. The most famous issues are the following:

- Design complexity (and therefore area) increases. The control logic for the computational asynchronous blocks waste power and occupies area.

- Standard CAD tools cannot always be used in the design process. The combinational loops that are used in the asynchronous controllers make static timing analysis more difficult. Formal verification and testing of asynchronous design is also more challenging.

- Asynchronous blocks should be interfaced with the rest of the system modules which may or may not be asynchronous. Thus, a special communication queues are required for such interfaces. These queues even been highly optimized tend to increase latency of the system, degrading the performance.

The problem of the asynchronous design complexity has been gradually alleviating. In [39] a tool to generate asynchronous controllers from a high-level concurrent specification is described. Also the methods for *de-synchronization* (automatically convert synchronous specification into asynchronous) are proposed [41]. Several asynchronous clones of synchronous microprocessors have been fabricated and tested [47, 74].

The control of asynchronous circuits can be often modeled with *marked graphs*, a sub-class of Petri nets [79]. Using marked graph model the performance of asynchronous circuit can be estimated [91, 92] and enhanced [4, 90].

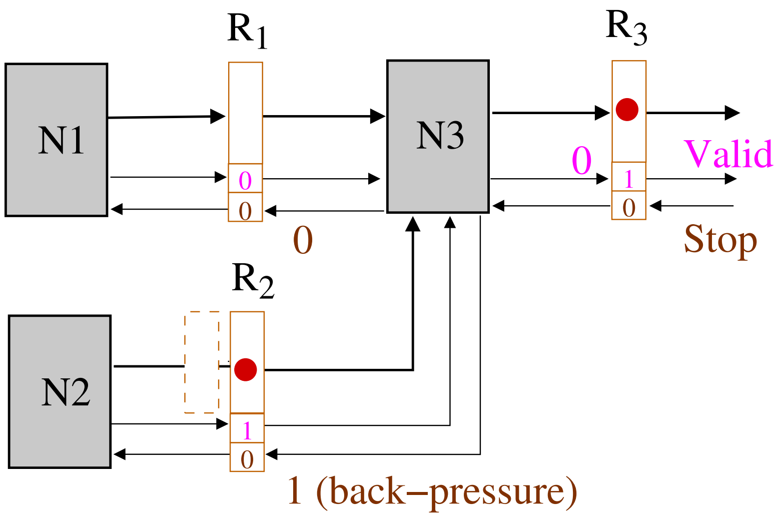


Figure 2.3: Synchronous elastic circuit.

2.1.2 Synchronous elastic circuits

Recently, a bridge between asynchronous and synchronous methodologies has been proposed [16, 38]. This work refers to this approach as *synchronous elastic* design. The term *elastic system*, ES, refers to both synchronous and asynchronous implementations. Synchronous elastic circuits can be seen as a discrete version of the asynchronous ones. The global clock is preserved, but each computational block does not have to provide valid data each clock cycle. It may *stall* [16] an arbitrary number of clock cycles, producing a *bubble* when not all the inputs are available. Figure 2.1(c) illustrates the synchronous elastic version of the synchronous circuit depicted in Figure 2.1(a). The clock signal does not directly affect the data path registers. Instead, first it is processed by the control module. Observe, that all three implementations depicted in Figure 2.1 have the same data-path. The only difference is the way the clock signal for the registers is generated.

To implement the control logic of the synchronous elastic design a pair of signals are associated with the data path. Generally they are referred to as *valid* and *stop* bits. The valid bit (Vi, Vo in the figure) indicates if a register stores valid data (*token*) or not (*bubble*). The stop bit (Si, So) is set when the new data can not be accepted. A simple control logic is introduced to control valid/stop bits flow. A register with associated valid and stop bit is called relay-station [16] or elastic buffer [38]. This work adopts the term Elastic Buffer, EB.

Figure 2.3 provides more details about the control logic of synchronous elastic circuits. The ES has three computational nodes, $N1, N2, N3$, and three EBs, $R1, R2, R3$. The EBs are drawn as rectangles. As usual, a rectangle with a dot inside represents an EB with token, as $R2$ and $R3$ do, while $R1$ stores a bubble; its valid bit is equal to zero. The dashed rectangle right before $R2$ represents an *extra buffer* of the control logic. Extra buffers are required when the output node does not accept the current token while some of the input nodes provide new ones. The size of this buffer is an important parameter, since the performance of an ES directly depends on it. Chapter 6 contributes on how the optimal size of each EB of an ES can be calculated. In Figure 2.3 the node $N3$ can not produce a token because of its input EB $R1$ does not have valid data. Hence, node $N3$ sets output valid bit to zero, produces a bubble. Also $N3$ informs the input $R2$ that its valid data should be hold for some time. The backward communication is done by means of the stop bit. It can be said that $R2$ is under *back-pressure* [18].

The main advantage of elastic design is that it accepts a set of correct-by-construction transformations. For example, a bubble may be inserted into any place of the circuit. Figure 2.2(c) shows the possible output a synchronous elastic circuit may produce during four clock cycles. Symbol τ inside a rectangle indicates that a bubble will be produced at this clock cycle. The contributions of this works explore some of the elastic transformations to enhance the performance of elastic circuits. The application for synchronous design is more evident. Possibly with small modifications, they can be useful for asynchronous design as well.

2.2 Linear Programming

This section provides a very brief overview of the topic, since a lot of excellent literature exist, see for example [81]. The canonical formulation of an LP problem is as simple as follows:

$$\begin{aligned} &\mathit{maximize}: \sum_{i=1}^{n} c_i x_i, \\ &\textbf{subject to:} \\ &\quad Ax \leq b, \\ &\quad x \geq 0. \end{aligned} \tag{2.1}$$

Where:

- $c = (c_1, \ldots, c_n)$ is an n-dimensional vector of real numbers,
- $x = (x_1, \ldots, x_n)$ is an n-dimensional vector of real variables,
- $A \in \mathbb{R}^{m \times n}$ is a matrix of real numbers with m rows and n columns. The ith row of A has elements $a_{i1}, \ldots, a_{in}$. Each row of A represents one constraint of the model, while each column represents a variable.
- $b = (b_1, \ldots, b_m)$ is an m-dimensional vector of real numbers.

The problem is to find an assignment for variables $x_1, \ldots, x_n$ that satisfies all the constraints and maximizes the value of expression $\sum\limits_{i=1}^{n} c_i x_i$ (the objective function). This maximal value is called *optimal value* of the LP. If an assignment satisfies all the constraints it is called *feasible solution* otherwise it is called *infeasible*.

It can be observed that the canonical formulation contains only "less than" constraints. The constraints: $\geq, =$ can be easily transformed into the canonical form. The reader may do it by himself as an exercise or refer to aforementioned literature.

Notice that if for some m-dimensional vector $y \geq 0$ it is verified that $\sum\limits_{j=1}^{m} a_{ij} y_j \geq c_i$, then the sum $\sum\limits_{j=1}^{m} y_j b_j$ provides an upper bound on the value of objective function in (2.1). Indeed, since if x is a feasible solution of (2.1) then

$$\sum_{i=1}^{n} x_i c_i \leq \sum_{i=1}^{n} x_i \sum_{j=1}^{m} a_{ij} y_j = \sum_{j=1}^{m} y_j \sum_{i=1}^{n} a_{ij} x_j \leq \sum_{j=1}^{m} y_j b_j.$$

The minimal upper bound for the objective function in (2.1) can be found by solving the following LP:

$$\begin{aligned} &\textit{minimize}: \ \sum_{j=1}^{m} b_i y_i, \\ &\textbf{subject to:} \\ &\quad A^T y \geq c, \\ &\quad y \geq 0. \end{aligned} \tag{2.2}$$

This LP is called *dual LP* for the LP (2.1). The LP (2.1) is called *primal* for the LP (2.2). It can be proved that if (2.1) is feasible then both (2.1) and (2.2) have equal optimal values [81].

LP problem is a particular case of the convex optimization [8]. There are efficient methods to solve LP problems. The most famous, probably, is simplex algorithm. Both commercial [58] and free [54] implementation of this and another algorithms exist.

2.2.1 Mixed Integer Linear Programming

If some of the variables in an LP model are constrained to have only integer values such type of optimization is called *mixed integer linear programming*, MILP. MILP is an NP-complete [51] problem but also has higher expressive power than LP. For example, if it is desired that only one of the two groups of the constraints are being satisfied in (2.1) we can achieve this introducing one integer variable into the model:

$$\begin{array}{l} maximize : \sum_{i=1}^{n} c_i x_i, \\ \textbf{subject to:} \\ \quad A_1 x \leq b_1 + \alpha \cdot A_{max}, \\ \quad A_2 x \leq b_2 + (1-\alpha) \cdot A_{max}, \\ \quad x \geq 0, \alpha \in \{0,1\}. \end{array} \tag{2.3}$$

Where A_{max} is a big enough constant to guarantee that $\sum_{j=1}^{n} a_{ij} x_j \leq b_i + A_{max}$ for each row i and any feasible solution of (2.1). Usually such constant is easy to find. If the value of the α is equal to zero, then the second group of the constraints is "disabled" and vice-versa.

The naive way to solve a general MILP problem is a branch-and-bound method. At each node of the search tree an LP relaxation is solved. This process can be greatly sped up. For example, one can add extra constraints to MILP that preserve the optimal solution while increase the number of integer values in the solutions of LP relaxations. For a special type of the matrix A (*totally unimodular matrix*) the simplex algorithm always finds integer solutions.

If all integer variables of the MILP are constrained to be equal to zero or one, then such type of MILP is called *mixed binary linear programming.* The complexity of a MILP problem greatly depends on the structure of the constraints. The CPLEX [58] solver can deal with MILP problems with thousands integer variables.

2.3 Cycle ratio of a directed graph

This section gives a formal overview for the minimal cycle ratio problem of a directed weighted graph. This formulation is the basis of the contribution presented in the Chapter 6 and also is used in Chapters 5 and 4. It is assumed that the reader is familiar with the basic graph nomenclature. An excellent introduction to algorithms [36] contains a chapter devoted to graphs.

Let $G = (V, E, W_1, W_2)$ be a bi-weighted directed *multi-graph*, where V is a set of vertices, $E \subseteq V \times V$ is a multi-set of edges, $W_1, W_2 : E \to \mathbb{R}$ are real edge labels (weights), $\sum_{e \in c} W_2(e) > 0$ for each directed cycle c of G. Figure 2.4 provides an example of such a graph. The vertices are drawn as cycles and the edges as arrows. For example, the arrow that goes from vertex s to vertex d means that graph has edge (s, d). Vertices b and c are connected by two edges, thus the multi-set E contains two equal elements (b, c). The value of the first and second weight of each edge is drawn as W_1/W_2 label in the figure. For example, for edge (f, s), $W_1((f, s)) = 0$ and $W_2((f, s)) = 4$.

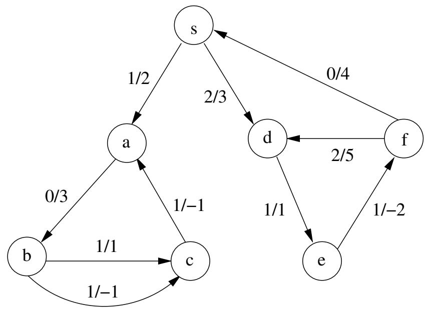


Figure 2.4: A directed bi-weighted graph

Definition 2.3.1 (Cycle ratio) *Define cycle ratio of a directed cycle c as following:*

$$\frac{\sum\limits_{e \in c} W_1(e)}{\sum\limits_{e \in c} W_2(e)}$$

For example, in Figure 2.4 the cycle ratio of the directed cycle $d \to e \to f \to d$ is equal to 1.

Definition 2.3.2 (Maximum cycle ratio) *The maximum cycle ratio of a graph G, $maxcr(G)$, is defined as follows:*

$$maxcr(G) = \max_{c \in C} \frac{\sum\limits_{e \in c} W_1(e)}{\sum\limits_{e \in c} W_2(e)},$$

where C is a set of all directed cycles of G.

Those directed cycles for which their cycle ratio is equal to the $maxcr(G)$ are called *critical*. Similarly, the minimum cycle ratio is defined. The maximum cycle ratio of the graph from the example is equal to 2. The critical cycle consist of the edges (a, b), (b, c) and (c, a), taking bottom most edge (b, c). The minimum cycle ratio of this graph is equal to 2/3; there are two cycles with the minimal ratio, they are $s \to d \to e \to f \to s$ and $a \to b \to c \to a$, but now the top most edge (b, c) must be taken. The following classical theorem allows to reduce the maximum cycle ratio problem to an LP problem.

Theorem 2.3.1 (Maximum cycle ratio) *The maximum cycle ratio of graph G is less than or equal to Θ if and only if there exist a function $d : V \to \mathbb{R}$ such that $W_1(e) - \Theta \cdot W_2(e) \le d(v) - d(u)$ for each edge $e = (u, v)$.*

Proof: $\Rightarrow$ Let d be such a function, then for each directed cycle c we have:

$$\frac{\sum\limits_{e\in c} W_1}{\sum\limits_{e\in c} W_2} = \frac{\sum\limits_{e\in c} W_1 - \sum\limits_{e\in c}\Theta\cdot W_2 + \sum\limits_{e\in c}\Theta\cdot W_2}{\sum\limits_{e\in c} W_2} \leq \frac{\sum\limits_{e\in c}(d(v)-d(u))}{\sum\limits_{e\in c} W_2} + \Theta = \Theta.$$

We used the fact that $\sum\limits_{e=(u,v)\in c}(d(v)-d(u))$ is equal to zero for a cycle c.

$\Leftarrow$ Let us assume that $maxcr(G) \leq \Theta$. Consider the following weighted directed graph $G = (V', E', W)$, where $V' = V \cup s$, $E' = E \cup \{(s,v)|v \in V\}$,

$$W(e) = \begin{cases} W_1(e) - \Theta\cdot W_2(e), & e \in E, \\ 0, & e = (s,v) \end{cases}$$

Basically, the source vertex s has been introduced to G. Notice, that no cycles in G' pass through the vertex s, thus the sets of all directed cycles of G' and G are equal. Then, for each directed cycle c of G' we have

$$\sum_{e\in c} W(e) = \sum_{e\in c}(W1(e) - \Theta\cdot W_2(e)) = \frac{\sum\limits_{e\in c}(W1(e)-\Theta\cdot W_2(e))}{\sum\limits_{e\in c} W_2(e)} \cdot \sum_{e\in c} W_2(e) =$$
$$= (\frac{\sum\limits_{e\in c} W_1(e)}{\sum\limits_{e\in c} W_2(e)} - \Theta)\cdot \sum_{e\in c} W_2(e) \leq 0.$$

The last inequality in the chain is true since $maxcr(G) \leq \Theta$ and $\sum\limits_{e\in c} W_2(e) > 0$. The weight of each cycle of G' is non-positive, then a longest path problem is well defined for G'. Let $d(v)$ be a weight of the longest path from the source vertex s to v, then by definition of the longest path for each edge $e = (u,v)$ we have

$$d(v) \geq d(u) + w(e)$$

or equivalently

$$W_1(e) - \Theta\cdot W_2(e) \leq d(v) - d(u).$$

■

The proof is a repetition of the proof of *Theorem 1 [71]* substituting $w(e) - \Theta$ by $W_1(e) - \Theta\cdot W_2(e)$. In turn, Theorem 1 [71] is a well known fact; it is used in [42] without proof. A similar idea is used in the proof of Karp's theorem [63]. Similarly, the minimum cycle ratio theorem can be proved.

Theorem 2.3.2 (Minimum cycle ratio) *The minimum cycle ratio of a graph G is greater than or equal to* Θ *if and only if there exist a function* $d : V \rightarrow \mathbb{R}$ *such that* $W_1(e) - \Theta\cdot W_2(e) \geq d(v) - d(u)$ *for each edge* $e = (u,v)$.

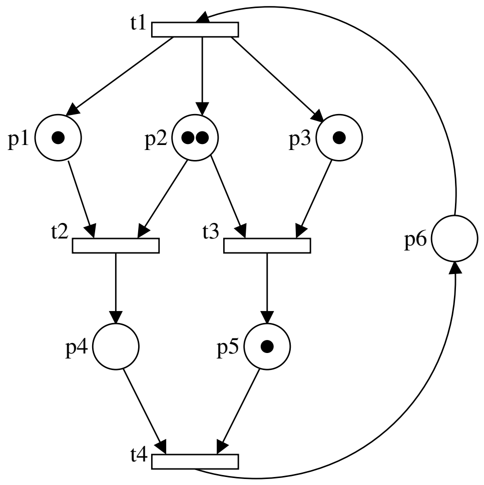


Figure 2.5: A Petri net

Using Theorem 2.3.2 the minimum cycle ratio of a graph G can be found as the optimal value of the following LP:

$$\begin{aligned} &\textit{maximize}: \ \Theta, \\ &\textbf{subject to:} \\ &\quad W_1(e) - \Theta \cdot W_2(e) \geq d(v) - d(u), \quad e = (u, v). \end{aligned} \tag{2.4}$$

The LP (2.4) finds the maximal value for Θ such that the shortest path problem is well defined for the corresponding graph G'. Efficient solutions for this LP exist [13, 32], as well as another efficient algorithms for the cycle ratio problem [42].

2.4 Petri nets

The Petri net (PN) is a mathematical model of a discrete system. It was introduced by professor Carl Adam Petri in his Ph.D. Thesis [86, 87]. This section gives basic definitions and introduces some Petri net nomenclature. This section provides a brief overview of Petri Nets. More details about PN can be found in excellent surveys [79, 99].

A Petri net has two components: a *net* and an *initial marking.* A net is a directed graph with two sort of nodes such that there is no edge between two nodes of the same sort. The graph of a PN is said to be *two colorable* or *bi-partite.* The two sort of nodes are called *places* and *transitions.* Figure 2.5 shows an example of Petri net. It is widely acceptable to draw places as circles, and transitions as boxes. Places can store *tokens*, represented by black dots. A distribution of tokens on the places of a net is called a *marking*, and correspond to the 'state' of the PN. For example, place

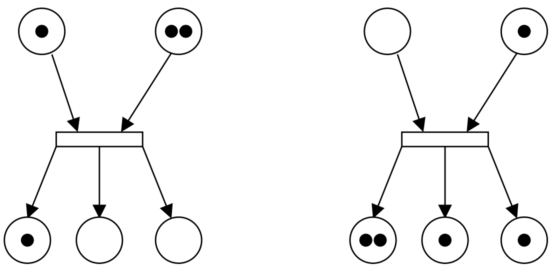


Figure 2.6: A Petri net before and after the firing of the transition

p_3 has one token. A transition of a net is *enabled* at a marking if all its input places (the places from which some edges leads to it) contain at least one token[1]. In the example, the enabled transitions are t_2 and t_3. An enabled transition can *fire*, and its firing changes the marking of the net: it removes one token from each of the input places of the transition, and adds one token to each of its output places[2]. Figure 2.6 shows on the left a PN containing an enabled transition, whose firing changes the marking to the one shown on the right.

Definition 2.4.1 (PN) *A Petri Net (PN) is a tuple $N = (P, T, F, m_0)$, where:*

- *P is a finite set of places*
- *T is a finite set of transitions*
- *$F \subset (T \times P) \bigcup (P \times T)$ is the flow relation*
- *$m : P \to \mathbb{Z}^+$ is the initial marking that assigns a non negative integer $m(p)$ to each place p.*

Usually we will refer to a particular place of a PN as p, and to a transition as t. Let us define the preset and the postset of a transition as ${}^\bullet t = \{p|(p,t) \in F\}$ and $t^\bullet = \{p|(t,p) \in F\}$, respectively. Firing a transition changes the current marking m to a new marking m':

$$m'(p) = \begin{cases} m(p) - 1 & p \in {}^\bullet t - t^\bullet \\ m(p) + 1 & p \in t^\bullet - {}^\bullet t \\ m(p) & otherwise \end{cases}$$

[1]There exist more general models called *weighted PN*. Each edge of a weighted net has an associated integer weight. Then a transition is enabled if the number of tokens in all its input places is at least the weight of the edge from this place to the transition.

[2]In weighted net a transition removes $w(p,t)$ tokens from each input places, $w(p,t)$ is a weight of the corresponding edge and adds $w(t,p)$ tokens to each output place.

let us denote this as $m \xrightarrow{t} m'$. Notice that the firing rule of the Petri Net is the exact model for the valid bit propagation in ESs (a node produce new valid data when all its inputs are valid). This allows us to model ESs with PNs.

Definition 2.4.2 (Reachability) *A marking m' is said to be reachable from marking m if there is a firingr sequence $m \xrightarrow{t_1} m_1 \xrightarrow{t_2} m_2 \ldots \xrightarrow{t_k} m'$*

Definition 2.4.3 (Liveness) *A PN is said to be live if every transition can eventually fire from any reachable marking.*

2.4.1 Marking equation

The structure of the PN can be described by its *incidence matrix.*

Definition 2.4.4 (Incidence matrix) *Let N be a PN (P, T, F, m_0), $|P| = n, |T| = m$. The incidence matrix $C \in \{0, 1, -1\}^{n \times m}$ of N is defined by*

$$C[p_i, t_j] = \begin{cases} -1 & p_i \in {}^\bullet t_j - t_j^\bullet \\ 1 & p_i \in t_j^\bullet - {}^\bullet t_j \\ 0 & \textit{otherwise} \end{cases}$$

For example, the Petri net from the Figure 2.5 has the following incidence matrix:

$$\begin{array}{rllll} & t_1 & t_2 & t_3 & t_4 \\ p_1: & 1 & -1 & 0 & 0 \\ p_2: & 1 & -1 & -1 & 0 \\ p_3: & 1 & 0 & -1 & 0 \\ p_4: & 0 & 1 & 0 & -1 \\ p_5: & 0 & 0 & 1 & -1 \\ p_6: & -1 & 0 & 0 & 1 \end{array}$$

Definition 2.4.5 (Firing count vector) *Let (P, T, F, m_0) be a PN and let $\sigma = (t_1, \ldots, t_k)$ be a finite sequence of transitions. The integer firing count vector $\overline{\sigma} \in \mathbb{Z}^{+|T|}$ of σ maps every transition t to the number of fires of t in σ.*

The new state of a PN after firing some transitions can be calculated usin the following lemma.

Lemma 2.4.1 (Marking Equation) *For any finite firing sequence $m \xrightarrow{\sigma} m'$ of a PN the following Marking Equation holds:*

$$m' = m + C \cdot \sigma \tag{2.5}$$

For example, the marking of the Petri net from Figure 2.5 can be represented as

$$m_0 = (1, 2, 1, 0, 1, 0)^T.$$

The new marking m after the enabled transitions t_2 and t_3 fire can be calculated as follows:

$$m = \begin{pmatrix} 1 \\ 2 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & -1 & 0 & 0 \\ 1 & -1 & -1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ -1 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 2 \\ 0 \end{pmatrix}.$$

The marking equation (2.5) provides a necessary condition for a given marking m' to be reachable from m. Next section states that for *marked graphs* it is also the sufficient condition.

2.5 Marked graphs

Definition 2.5.1 (Marked Graph) *A Marked Graph (MG) is a special class of Petri nets where places have exactly one input and one output transition. That is* $|{}^\bullet p| = |p^\bullet| = 1, p \in P$.

An MG can also be defined as a directed weighted multi-graph (N, E, m_0), where N is a set of nodes (transitions), E is a set of edges (places), m_0 is an initial marking. Conventionally an MG is represented as a graph with tokens placed on the edges. Figure 2.7 shows two equivalent representations of the same MG, the first draws it as a general PN, the second one as a directed graph, where each edge represents a place. Next, we will use notation (t_i, t_j) to refer to the place of an MG. For example, edge (t_2, t_3) refers to the place p_3 of the MG. For the sake of notation, the total number of tokens in a subset $\phi \subseteq P$ at a given marking m is denoted by $m(\phi) = \sum\limits_{p \in \phi} m(p)$.

An MG is strongly connected if its underlined graph is strongly connected. Some useful properties of strongly connected MGs [79] are:

Property 2.5.1 (Liveness) *A MG is live iff every directed cycle* $\mathbf{c}$ *is marked positively at* m_0*, i.e.,* $m_0(\mathbf{c}) > 0$.

Property 2.5.2 (Reachability) *A marking* m *is reachable iff* $m(\mathbf{c}) = m_0(\mathbf{c})$ *for every directed cycle* $\mathbf{c}$ *of the MG.*

The retiming interpretation of these properties allows easy to prove that for a strongly connected retiming graph any registers assignment that preserves the number of registers at each directed cycle of the graph is a valid retiming configuration that can be achieved only with forward retiming (see Section 4.3).

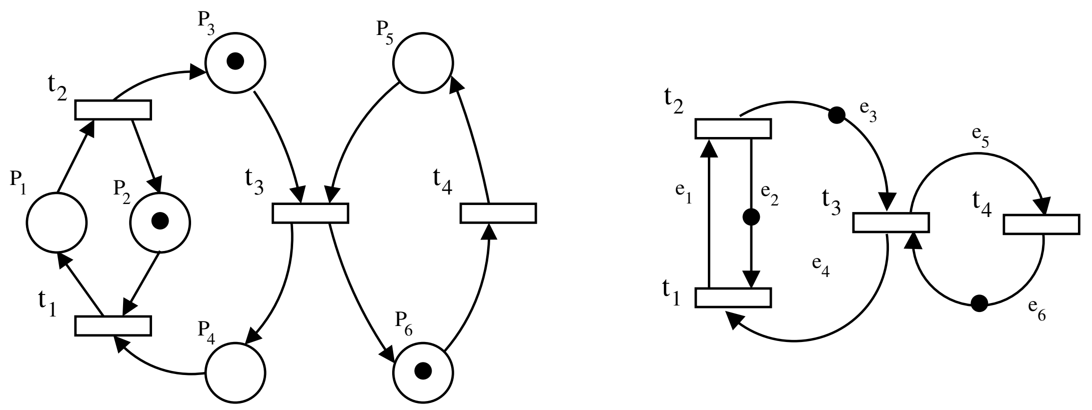


Figure 2.7: Two representations of the same marked graph. The left most — as a general Petri Net, the right most — as a directed graph, where edges represent places.

2.5.1 Timed marked graph

To use MGs for performance estimation a timing information should be added into them. There are two ways of introducing a time information in an MG, namely associating delays with transitions or with places. It can be proved that the both ways are equivalent [98]. In this section the delays are associated with the transitions, since this is more widespread in the literature.

Definition 2.5.2 *A* Timed Marked Graph *(TMG) is a tuple* (N, E, m_0, δ)*, where* (N, E, m_0) *is an MG, and* $\delta : N \to \mathbb{R}^+ \cup \{0\}$ *assigns a non-negative delay to every transition.*

In a TMG, a transition t fires $\delta(t)$ time units after becoming enabled. When the enabling degree of a transition is greater than one, two different firing semantics can be considered:

- *Single server* semantics. No multiple instances of the same transition can fire simultaneously.
- *Infinite server* semantics. An arbitrary number of instances of the same transition can fire simultaneously.

The single server semantics is a particular case of infinite server semantics: the addition of a self-loop place with one token, i.e., a place p such that $p^\bullet = {}^\bullet p$ and $m_0(p) = 1$, around each transition guarantees single server semantics [15]. Henceforth, if it is not stated explicitly, the infinite server semantics will be assumed.

The average marking of a place p, denoted as $\overline{m}(p)$, represents the average occupancy of the place in an infinite execution. Formally the average marking vector for all places is defined as:

$$\overline{m} = \lim_{\tau \to \infty} \frac{1}{\tau} \int_0^\tau m(\xi) d\xi$$

where ξ is a time variable. For marked graphs the average marking is always well-defined [15].

2.5.2 Performance evaluation

Let us now formally define the throughput of a TMG. Then, TMGs can be used for the performance estimation of ESs.

Definition 2.5.3 (Throughput) *The throughput of a transition t of a TMG, $\Theta(t)$, is the average number of times t fires per time unit, in the infinitely long execution of the system.*

$$\Theta(t) = \lim_{\tau \to \infty} \frac{\sigma(\tau)}{\tau}$$

All transitions of a strongly connected component of a TMG have the same throughput. It can be proved that the throughput of a TMG is equal to its minimal cycle ratio tokens/delay [92]. For example, assuming that all transitions of the MG depicted in Figure 2.7 have unit delay, its throughput is equal to 1/3. The critical cycle is $(t_1 \to t_2 \to t_3 \to t_1)$, it has one token and its total delay is equal to three units of time.

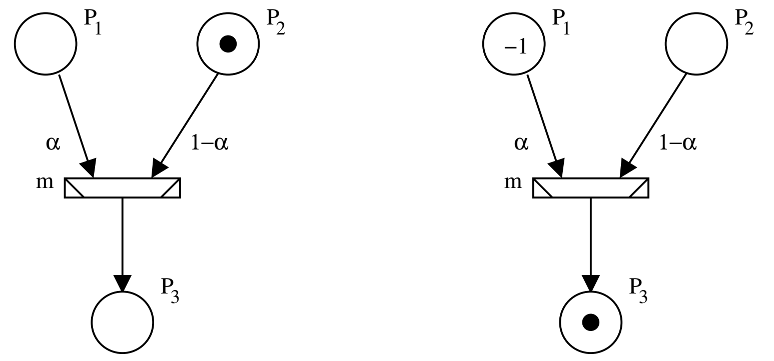
From Theorem 2.3.2 the following LP model can be derived to calculate the throughput of a TMG:

$$\begin{array}{l} \mathit{maximize}: \ \Theta, \\ \textbf{subject to:} \\ \quad \delta(t) \cdot \Theta \leq \widehat{m}(p), \quad p \in {}^\bullet t \\ \quad \widehat{m} = m_0 + C \cdot \sigma. \end{array} \tag{2.6}$$

In this formulation, a firing count vector σ is a vector of real variables. The provided LP can be easily rewritten in the form of (2.4). Another way to obtain the same LP is using the theory of queues and Little's law [76], see for example [15]. If a TMG is not strongly connected, then (2.6) returns minimal throughput of all its strongly connected components. The performance of an ES with only late evaluation nodes can be estimated as the throughput of the corresponding MG [92].

2.5.3 Early evaluation

In order to model ESs with early evaluation nodes, a classical MG model was extended with *multi-guarded transitions* [59]. A guard $g(t)$ of the transition t is a subset of input places $g(t) \subset {}^\bullet t$. Each transition t has a set of guards associated



(a) Enabled early evaluation transition

(b) Marking after firing the early enabled transition

Figure 2.8: Firing of early evaluation transition leads to negative marking in the place P_1

$G(t) = \{g_1(t), \dots, g_k(t)\}$, such that $\bigcup\limits_{i=1}^{k} g_i(t) = {}^{\bullet}t$. At a given moment only one particular guard of each transition can be active. Transition t is enabled if the active guard is $g_i(t)$ and all places of $g_i(t)$ are marked. The firing of an enabled transition changes the marking in the same way a classical transition does. After firing the transition a new active guard is selected non-deterministically. In order to model non-determinism each guard $g_i(t)$ of transition t has associated a probability $\gamma_i(t)$, $\sum\limits_{i=1}^{k} \gamma_i(t) = 1$. The selection of a guard is assumed to be a memoryless process. That is, the probability to be selected does not depend on another guards that were selected before. Notice, that the firing of an early enabled transition may lead to a negative marking. For example, consider early evaluation transition m depicted in Figure 2.8(a). Assume that m has two one-element guards: $g_1(m) = \{P_1\}, g_2(m) = \{P_2\}$, with the associated probabilities $\gamma(g_1) = \alpha, \gamma(g_2) = 1 - \alpha$. Assume that at the given moment the guard g_2 is selected, then m is enabled and fires in according with the Petri net firing rule, producing negative marking in the place P_1 (Figure 2.8(b)).

The marked graph with multi-guarded transitions is called *guarded marked graph* (GMG), Figure 2.9 shows examples of GMGs. The only multi-guarded transition, m, is drawn differently. The usual graphical representation of multiplexer is used to select the muli-guarded transition. This is because, in ESs multiplexers are the main candidates to be early evaluation. Transition m has two guards associated

$$g_1(m) = \{(t_2, m), (s, m)\}, \gamma(g_1(m)) = 1 - \alpha$$

and

$$g_2(m) = \{(t_3, m), (s, m)\}, \gamma(g_2(m)) = \alpha.$$

Definition 2.5.4 (Timed Guarded Marked Graph [59]) *A Timed Guarded Marked Graph (TGMG) is a tuple $(N, E, G, m_0, \delta, \gamma)$, where (N, E, m_0, δ) is a TMG, G assigns to each transition a set of guards, γ assigns a probability to each guard to be selected.*

A transition t is called *singleton* if all its guards have exactly one place. A TGMG is called singleton if all its multi-guarded transitions are singleton. A transition is called *simple* (late evaluation) if it has one guard associated.

The steady state throughput of a given transition of a TGMG is defined as the average number of firings per unit of time, the same way as for classical MGs (Definition 2.5.3).

Definition 2.5.5 (Steady state throughput) *The throughput, $\Theta(t)$, of a given transition t of a TGMG is defined by the following equation:*

$$\Theta(t) = \lim_{\tau \to \infty} \frac{\sigma(\tau)}{\tau}.$$

All transitions of a strongly connected TGMG have the same throughput [59].

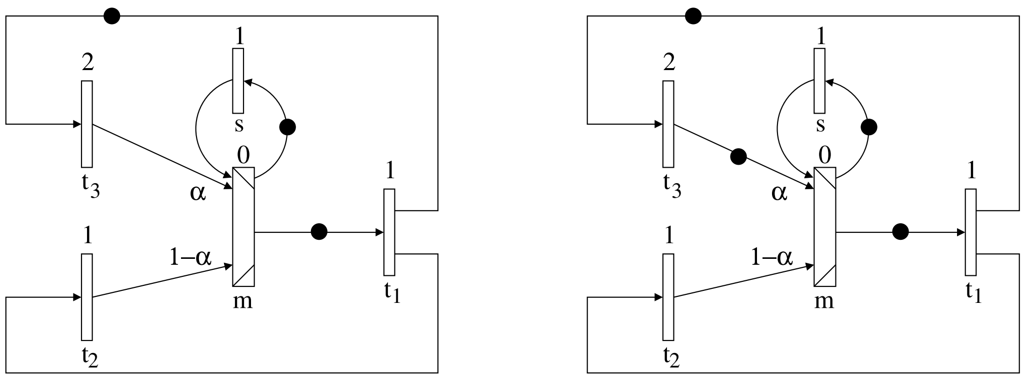
The throughput of a TGMG can be calculated by analyzing a corresponding *Markov chain* [59,109]. Each state of such a chain is defined by a reachable marking of the TGMG. Let us give an example how this works, since the throughput of ESs with early evaluation nodes can be calculated in the same way.

2.5.4 Calculating throughput using Markov chains

To construct a Markov chain that models the behavior of a TGMG a kind of exact simulation should be performed. At each step of this simulation the set of enabled transitions is identified and the next state (marking) is calculated by firing all enabled transitions. If the enabling of some transitions depends on the guards that will be selected, two or more possible markings are considered for this transition for all possible firings. First, the new markings are searched among those that have been already found so far. If the search is unsuccessful a new state is created, otherwise a loop has been found and the simulation is terminated.

Let us again consider a TGMG depicted in Figure 2.9(a). It has five transitions t_1, t_2, t_3, m, s; on the top of each transition its delay is drawn, for example, the delays of the transitions t_1, t_2 and s are equal to one unit of time. Transition m has two guards g_1 and g_2; the rest of the transitions are simple. The throughput of this TGMG without m being early evaluation is equal to 1/2. The delay of critical cycle ($m \to t_1 \to t_2 \to m$) is equal to two and there is one token on it.

Now, let us calculate the throughput of this TGMG based only on Definition 2.5.5, using Markov chain analysis. Let us assume that from the initial state, transition t_3 fired for the first time at the time stamp 1. In general, any assumption about



(a) $\Theta = 3/5, \alpha = 0.5$ (b) $\Theta = 2/3, \alpha = 0.5$

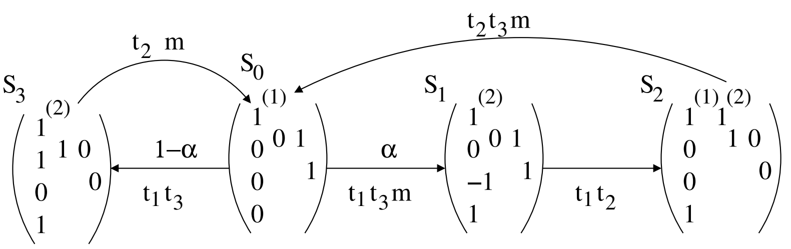
Figure 2.9: Timed multi-guarded marked graphs.

initial state can be done without altering the throughput. On the other hand, such assumptions may simplify the Markov chain by reducing the number of states.

Figure 2.10(a) depicts the Markov chain of the TGMG from Figure 2.9(a). The initial marking of the TGMG corresponds to the state S_0 of the Markov chain in the Figure 2.10(a). Here we just draw the number of tokens in each place, trying to preserve respective positions of the places. For example, the first column corresponds to the places $(t_1, t_3), (t_3, m), (t_2, m), (t_1, t_2)$, from top to bottom; the following corresponds to the places $(s, m), (m, s)$ and finally (m, t_1) follows. The notation $1^{(d)}$ means that the processing of the token will be finished in d units of time. We use this notation only for the place (t_1, t_3), since the delay of the transition t_3 is greater than one and the duration of each state in the Markov chain 2.10(a) is one unit of time. Figure 2.11 shows detailed version of the Markov chain. The anti-token of the state S_1 is drawn as unfilled dot with the sign of minus inside. As before, the notation (d) means that the given token will be processed within d unit of time.

From the initial state S_0 after one unit of time the TGMG may go to the state S_1 or to the state S_3 depending on the guard that will be selected. If the guard $g_2(m)$ is selected, then transitions t_1, t_3, m, s will fire, resulting marking S_1, otherwise the state is changed to S_3 and only three transitions t_1, t_3 and s fire. Each arc in the figure is marked with the transitions that fire. Transition s is not drawn on the arcs to make pictures cleaner. Thus, from the state S_0 the TGMG can go to the state S_1 with the probability α or to the state S_3 with the probability $1 - \alpha$. Repeating the same procedure for the states S_1 and S_3 we will find only one new state, S_2. In the state S_3 the guard is already selected and the only thing we can do is to fire all enabled transitions resulting initial state S_0. In the state S_2 the place (t_1, t_3) has two tokens, one will be processed within one unit of time, another within two units of time.

Each directed cycle of a Markov chain corresponds to a particular behavior of the


(a) Markov chain for the TGMG 2.9(a)

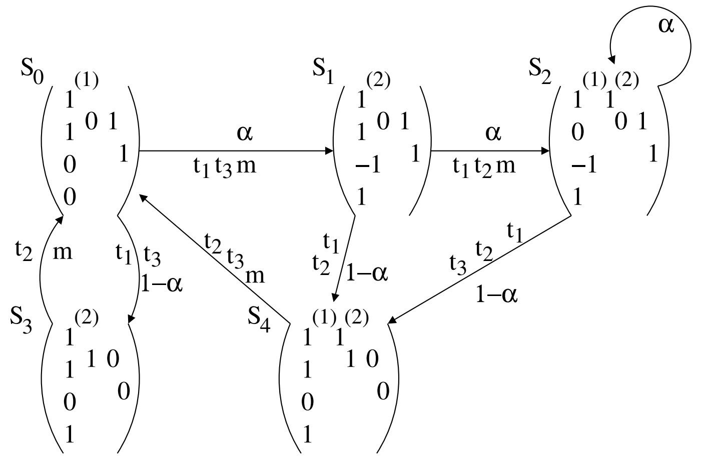

(b) Markov chain for the TGMG 2.9(b)

Figure 2.10: Throughput calculation of TGMG with Markov chains.

TGMG. For example, the directed cycle ($S_0 \to S_3 \to S_0$) corresponds to the behavior when guard g_1 is always selected. At this cycle all the transitions fire exactly once, hence the throughput of this behavior is equal to 1/2. Similarly the throughput of the behavior ($S_0 \to S_1 \to S_2 \to S_0$) can be calculated and it is equal to 2/3. One could make a guess that the throughput of the TGMG is equal to the average of the throughputs of these two behaviors, that is: $\alpha \cdot 1/2 + (1-\alpha) \cdot 2/3$. However, this is not true, for example, if $\alpha = 0.5$ then, as we will see in short, the throughput is equal to 3/5, while the value of the aforementioned expression is equal to $7/12 \approx 0.583$.

To find the throughput of the TGMG we should resolve the Markov chain. That means, for each state $S_i, i = 0, \ldots, 3$ find a probability, $p(S_i)$, for the TGMG to be in this state. Then, for any transition t we can find a probability of being enabled as the following sum [59]:

$$p(enab(t)) = \sum_{\text{t enabled in } S_i} p(S_i).$$

It can be proved [59] that the throughput of t is equal to $p(enab(t))/\delta(t)$.

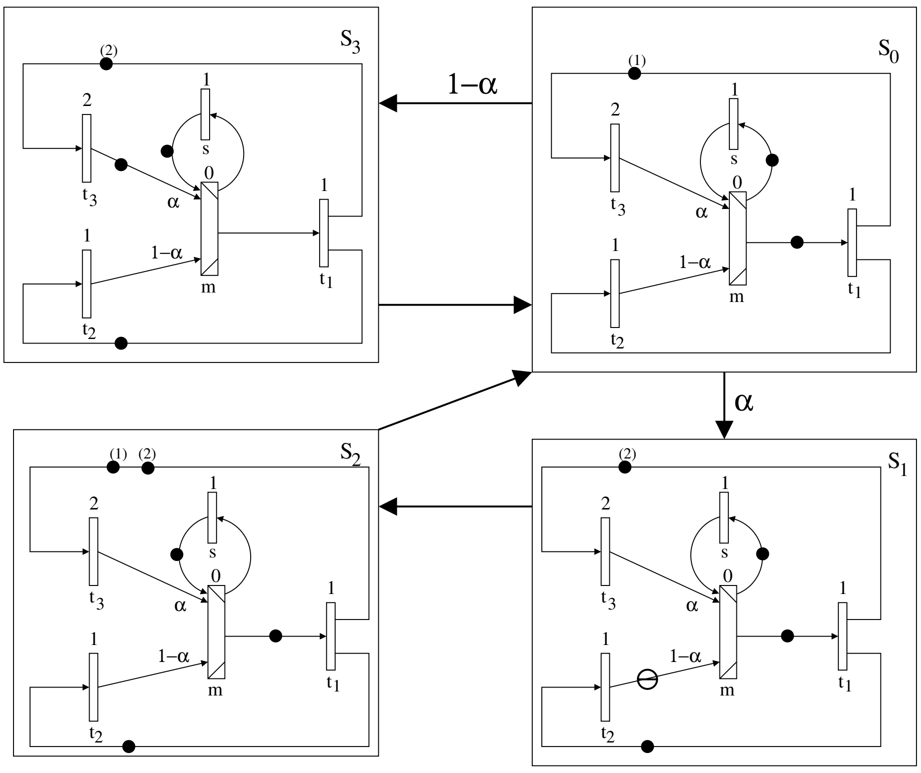


Figure 2.11: Detailed version of the Markov chain from Figure 2.10(a)

To find $p(S_i)$ in our simple example it is enough to carefully look at the Markov chain in Figure 2.10(a). Then

$$p(S_1) = \alpha \cdot p(S_0),\ p(S_2) = p(S_1) = \alpha \cdot p(S_0),\ \text{and } p(S_3) = (1-\alpha) \cdot p(S_0).$$

Since, the sum of the all probabilities must be equal to one, we have the following equation

$$p(S_0)[1 + \alpha + \alpha + 1 - \alpha] = 1,$$

from which it follows that $p(S_0) = 1/(2+\alpha)$. Now let us observe that transition t_1 is enabled only in the states S_0 and S_1, hence the throughput of this TGMG is equal to

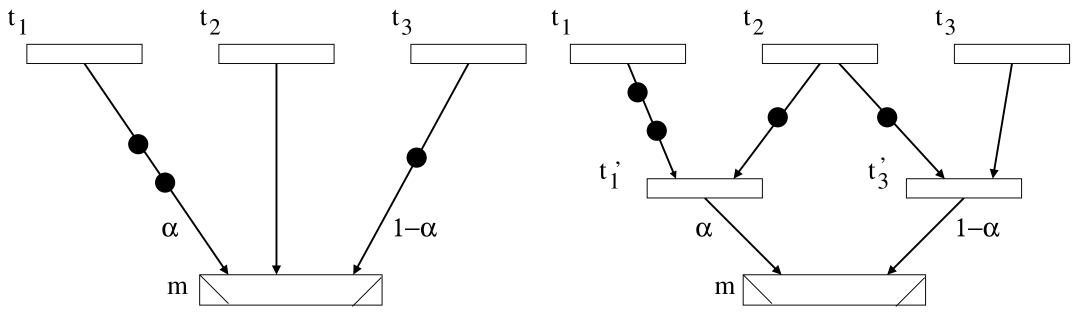
$$\frac{1}{2+\alpha} + \frac{\alpha}{2+\alpha} = \frac{1+\alpha}{2+\alpha}.$$

Remember, that the throughput of all transitions must be the same. If $\alpha = 0$, then the throughput is equal to $1/2$ which correspond to the late evaluation throughput. If $\alpha = 1$, then the throughput is equal to $2/3$, for $\alpha = 0.5$ the throughput is equal to $3/5$.

One may ask if the loop (m, s) has impact on the throughput of the TGMG. In this example it does not, removing this loop does not change the throughput of the TGMG. To construct an example were this loop makes the difference let us add one token to the place (t_3, m), resulting TGMG depicted in Figure 2.9(b). The Markov chain of this TGMG is shown in Figure 2.10(b). Resolving this chain, the following expression for the throughput is obtained $1/(2-\alpha)$. It can be observed that the throughput grows from $1/2$ to 1 when α grows from zero to one. The throughput is equal to $2/3$ for $\alpha = 0.5$. Now let us remove transition s from the TGMG. The Markov chain of the result TGMG is quite big, so to find a throughput for $\alpha = 0.5$ we used a simple simulator which fire each transition many times and calculates the throughput directly by its definition. The value that simulator returns is 0.75, more than 10% greater then the throughput of the same TGMG with the transition s (0.666). Thus, provided example is interesting because a cycle with the ratio tokens/delay greater than the throughput has been removed from the TGMG but this leaded to the throughput increasing.

2.5.5 Calculating a throughput upper bound with LP

The Markov chain analysis is a general way to calculate the throughput of a TGMG. However, the number of states in the Markov chain grows exponentially both with the number of tokens in the TGMG and with the number of places. Thus, such analysis can be unpractical for large TGMGs. From the other hand, Markov chain analysis is not very convenient tool at the time of building algorithms for performance optimization.



(a) A non-singleton transition (b) Equivalent singleton transition

Figure 2.12: Transformation of non-singleton transition to singleton that preserves the throughput of a TGMG

Next, we will overview an LP based approach to estimate the throughput of a singleton TGMG. This approach was proposed in [59].

Let T_1 be a set of simple transitions, T_2 a set of multi-guarded transitions of a singleton TGMG $(T_1 \cup T_2, E, G, m_0, \delta, \gamma)$. Then, the throughput upper bound of the TGMG is the optimal value of the following LP:

$$\begin{array}{ll} \textit{maximize}: \ \Theta, & \\ \textbf{subject to:} & \\ \quad \delta(t)\cdot\Theta \le \sum_{p\in {}^\bullet t} \gamma(p)\cdot\widehat{m}(p), & t\in T_2 \\ \quad \delta(t)\cdot\Theta \le \widehat{m}(p), & p\in {}^\bullet t, t\in T_1 \\ \quad \widehat{m} = m_0 + C\cdot\sigma. & \end{array} \tag{2.7}$$

The notation $\gamma(p)$ is used since each guard of a multi-guarded transition consists on only one place. Comparing with the LP (2.6), the only difference is special constraints for each multi-guarded transition.

A TGMG always can be transformed to a singleton TGMG with the same throughput [59]. Figure 2.12 illustrates this transformation. Originally, transition m in Figure 2.12(a) has two guards with two places in each one: $\{(t_1, m)(t_2, m)\}$ and $\{(t_3, m), (t_2, m)\}$. After transformation, m is singleton transition with two guards. Figure 2.12(b) shows the result TGMG. The delays of transitions t_1' and t_3' are equal to zero. After this transformation the throughput upper bound of an arbitrary TGMG can be calculated with LP (2.7).

In Chapter 5 we will construct a TGMG model for ESs with early evaluation nodes. The model will allow the analysis and optimization of the performance of ESs with early evaluation nodes.

Chapter 3

State of the art

This chapter starts with a formal review of the retiming optimization technique, since the first and second contributions combine retiming with recycling to optimize the performance of ESs. Afterward, the overview of synchronous elastic designs, proposed in the literature, follows. After this, the buffer sizing and buffer insertion (slack matching) optimization techniques are introduced. The last contribution combines these techniques to optimize the performance of ESs. Finally, it is illustrated how the synchronous elastic system can be built using static scheduling.

3.1 Retiming

Retiming [66] is a classical technique for sequential optimization. Retiming represents a sequential circuit as a weighted directed multi-graph $G = (V, E, d, R)$, (*retiming graph*), where V is a set of vertices that represent combinational nodes, E is a set of edges that represent interconnections between combinational nodes, $d : V \to \mathbb{R}^+$ is a non-negative real function that represents a propagation delay for each combinational node, $R : E \to \mathbb{Z}^+$ represents the number of registers on the edge. If $R(e) = 0$ for edge $e \in E$ then e represents a *wire*. A retiming graph G has a well-defined structure as a circuit if the following property holds: in any directed cycle of G, there is some edge which is not a wire. For the corresponding sequential circuit this property is equivalent to the absense of combinational cycles.

Figure 3.1(a) shows a retiming graph of the elastic circuit depicted in Figure 1.3(a). The nodes (cycles) are labeled with their delays. The edges are labeled with the corresponding number of registers, unlabeled edges have no registers. The combinational delay of the path $c \to f \to g$ is equal to 18 units of time. For the path $c \to d \to e \to a$ it is equal to 21 units of time. There no path with the delay greater than 21 exist, hence, 21 is the *cycle time* of the circuit.

Formally, retiming is defined in [66] as an integer function $r : V \to \mathbb{Z}$ that trans-

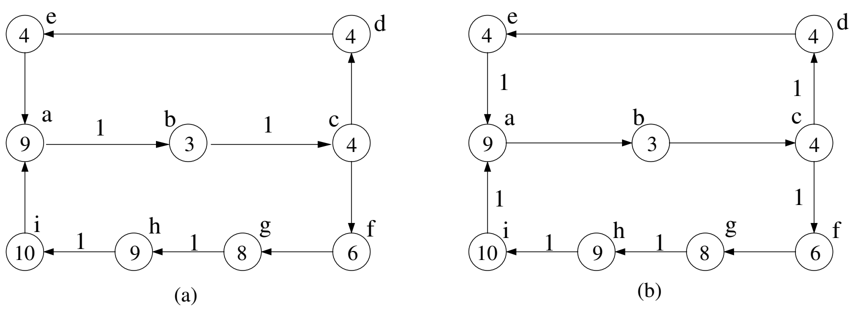


Figure 3.1: (a)Example of a retiming graph. (b) Min-delay retiming configuration.

forms the number of registers for each edge e, $R(e)$ to $R'(e)$ as follows:

$$R'(e) = R(e) + r(v) - r(u), \quad R'(e) \geq 0.$$

Let us apply the following retiming on graph in Figure 3.1(a): $r(a) = 1, r(c) = -1$, r is zero for the rest of the nodes. In order to apply $r(a) = 1$ it is enough to remove one register from each output edge of node a and add one register to each input edge of it; $r(c) = -1$ moves register across node c in the opposite direction. Figure 3.1(b) shows the result graph. The combinational path with the greatest delay is $a \rightarrow b \rightarrow c$, the cycle time is equal to $9 + 3 + 4 = 16$ units of time.

Retiming may change the cycle time and the number of registers in the circuit while preserving its sequential behavior [65]. With the decreased cycle time the number of registers (area) in the example has been increased. There are two general formulations of the retiming problem:

- Min-delay retiming. The optimization target is the cycle time of the circuit.
- Min-area retiming. The optimization target is the area (number of registers) under cycle time constraints.

Min-delay retiming can be solved in polynomial time [66,96,97]. The min-area retiming is an NP complete [66] problem but some instances of it can be solved fast [57]. The exact algorithm for min-area retiming tend to be based on integer linear programming.

A more accurate model for min-area retiming takes into account fan out registers sharing. For example, in Figure 3.1(b) the output of the node c is driven to the nodes: d and f. Both edges (c, d) and (c, f) have a register, thus, the physical implementation of such circuit requires only one register. Registers sharing can be incorporated to retiming by introducing a fake fork node with zero delay [66,95]. Figure 3.2 illustrates the idea for the node c.

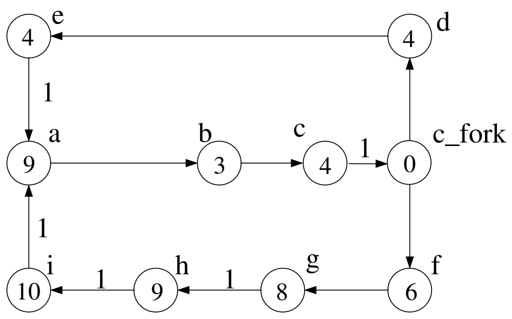


Figure 3.2: Registers sharing in the retiming.

In [84] it is shown that the cycle time retiming can achieve is lower bounded by the following expression

$$\max_{c\in C} \frac{\sum\limits_{v\in c} d(v)}{\sum\limits_{e\in c} R(e)}, \tag{3.1}$$

where C is the set of all directed cycles of the retiming graph. For example, for the retiming graph depicted in Figure 3.1 the lower bound (3.1) is equal to

$$\max(12/2, 49/4) = 12.25.$$

Retiming changes the initial state of the circuit. After applying retiming the equivalent initial state should be calculated. It is always possible if the retiming graph is strongly connected. In general, finding an equivalent initial state of the retimed circuit is not always possible [106].

Retiming is applicable to ESs as well. The only extra constraint is that only the equally initialized EBs (tokens or bubbles) can be moved across computational nodes.

3.1.1 C-slow retiming

The main limitation of the retiming is that it preserves the number of registers at each directed cycle of the retiming graph [84]. C-slow transformation overcome this limitation making c copies of each register. By c-slowing a circuit, the throughput (number of input data processed per cycle) is reduced to $1/c$ but a circuit can be made feasible for a very small clock period.

Figure 3.3(a) shows a 2-slow version of the circuit from Figure 3.1(a). Applying to it the retiming: $r(a) = 1, r(b) = 0, r(c) = -1, r(d) = -1, r(e) = 0, r(f) = 1, r(g) = 2, r(h) = 1, r(i) = 0$ leads to the circuit depicted in Figure 3.3(b). The result circuit has cycle time of 10 units of time but processes every input data during two clock cycles (its throughput is equal to 0.5). The performance of this circuit can

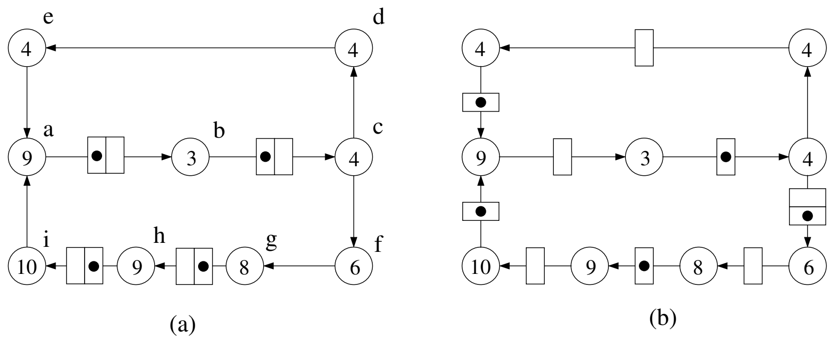


Figure 3.3: (a) 2-slow version of the circuit from Figure 3.1(a), (b) 2-slow and retiming

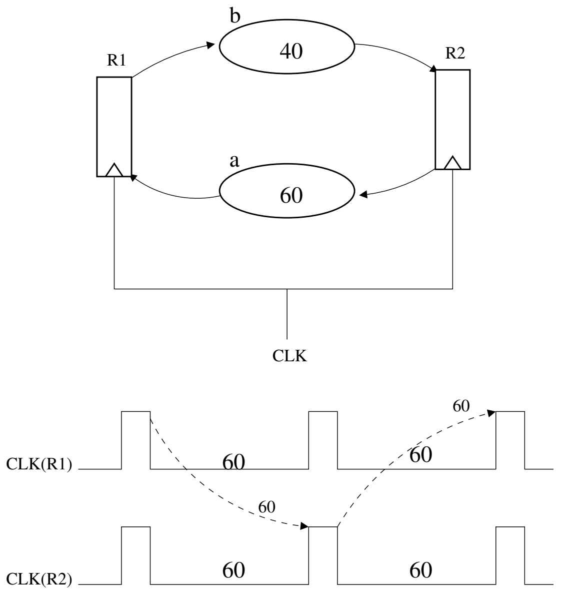
be estimated as the cycle time of a sequential circuit with cycle time of 20 units of time, $10/0.5 = 20$, which is worse than its min-delay retiming cycle time.

3.1.2 Retiming and clock skew

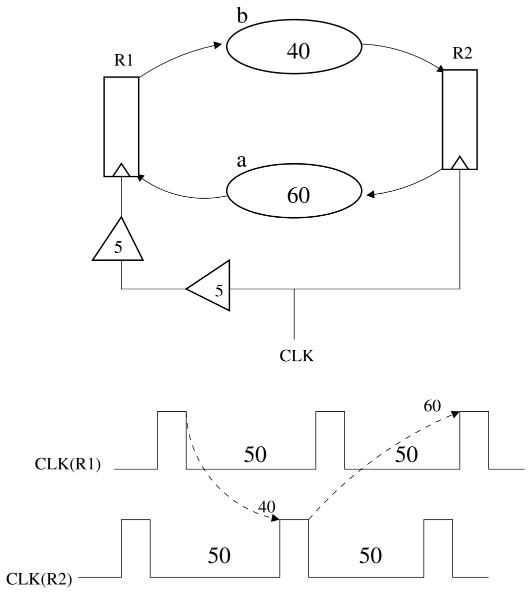
Retiming has a strong analogy with the clock skew [46] optimization technique. Clock skew is the difference between the time moments the clock signal arrives at different registers of a circuit. It can be shown that retiming of a node n $r(n) = 1$ can be achieved by increasing clock skew of the corresponding registers [46, 96]. In general, clock skew and retiming can be viewed as continuous and discrete optimizations with the same effect.

Since clock skew is a continuous optimization, it can provide performance beyond the min-delay retiming cycle time. For example, consider a simple circuit depicted in Figure 3.4(a) with two registers $R1, R2$ and two combinational nodes a, b. The nodes are labeled with their combinational delays. The cycle time of this circuit is equal to 60 units of time and retiming can not improve this value. However, the lower bound (3.1) for the cycle time time is equal to $100/2 = 50$ units of time. The diagram underneath the circuit reflects the fact that clock signal arrives simultaneously at both registers and the clock period is equal to 60 units of time. Now, let us postpone the clock for $R1$ by ten units of time as it is shown in Figure 3.4(b). With such configuration the circuit can work with cycle time of 50 units of time. The timing diagram is shown in the bottom of the Figure 3.4(b). Observe, that node b has 40 units of time to recalculate its value, while node a still has 60 units of time.

The analogy between the skew optimization and the retiming problems has been exploited to propose efficient algorithms for retiming of large circuits [96].



(a) Clock cycle is equal to 60. Clock skew is equal to zero for both registers.



(b) Clock cycle is equal to 50. Clock skew is equal to 10 for register $R1$.

Figure 3.4: Using clock skew for the optimization of the cycle time.

3.2 Synchronous elastic circuits

3.2.1 Latency insensitive design

Latency insensitive design (LID) [16, 17] was one of the first paradigms proposed in the literature to elasticize synchronous design while preserving global clock and hence standard synchronous design flow.

In [16] some registers of a synchronous circuit are converted to *relay-stations* by associating a *void* bit with each of them. When the void bit associated with a register is asserted, the data stored in the register is assumed to be non-informative and should be ignored. If some of the input channels provide valid data while another does not the node must be able to freeze its state (*stall*) and also to inform channels to resend the same valid data until all inputs become valid. This logic is implemented by a *shell* which is added to each computational block of the design. Computational blocks are some big sequential blocks of the circuit. Figure 3.6 shows the illustrative implementation that was presented in [18]. While one of the inputs does not provide valid data (a *voidIn* is set) or one of the outputs cannot accept new data (a *stopIn* is set) the clock signal is disabled for the whole block making it to freez.

The registers with void/stop bit and the control logic is called a *relay station*, they are marked with shadowed rectangles in the figure. In [18] an implementation of relay station is also presented. Unfortunately, the control logic is left as a black

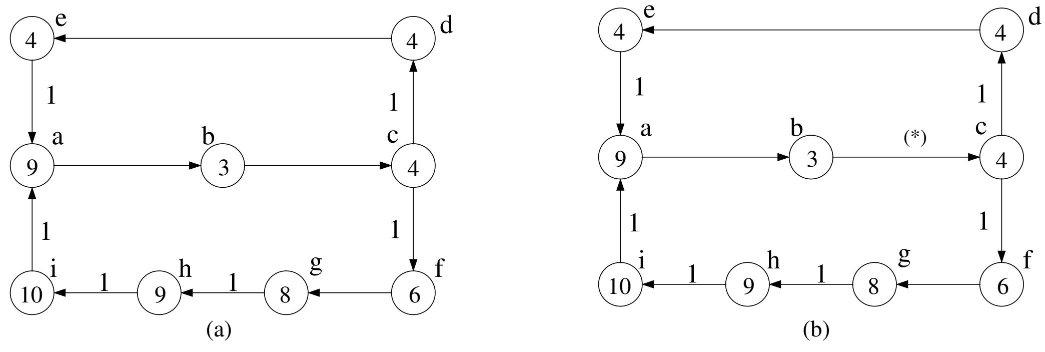


Figure 3.5: (a) Retiming graph from the Figure 3.1(b), (b) After applying recycling, (*) represents a bubble.

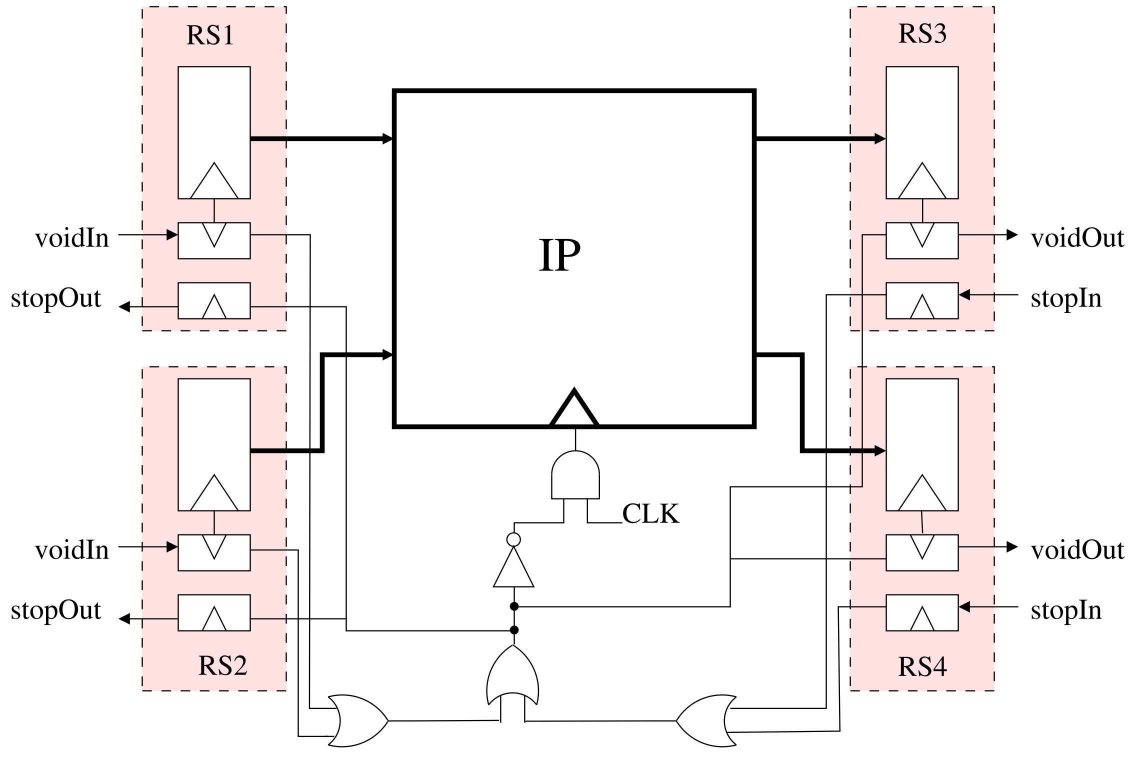


Figure 3.6: Shell encapsulation.

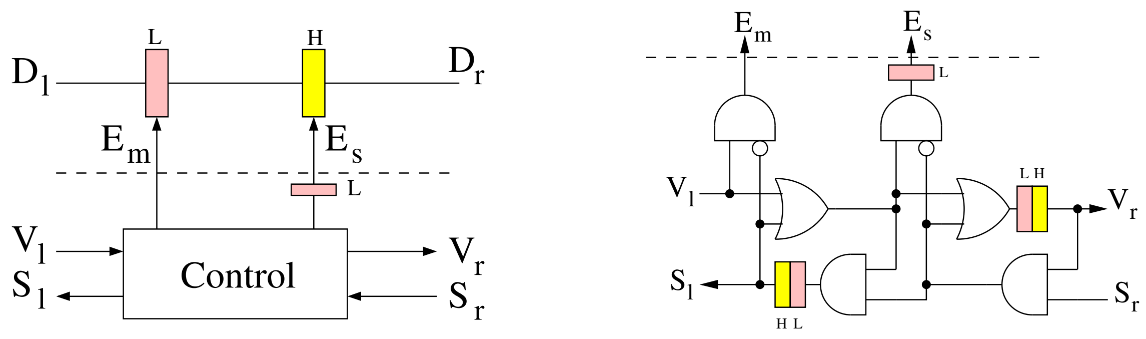


Figure 3.7: A possible implementation of Elastic Buffer proposed in [38]

box. In the next section an alternative implementation is presented with the details of the implementation of EB.

Different schemes of relaxation the strict firing semantics of computational blocks were later proposed [25, 67]. These schemes aid at improving the performance of the LID.

Recycling

LID enables the automatic insertion of bubbles. In [16] such transformation was called *recycling*.

Figure 3.5 shows how the critical combinational path of the retiming graph from the Figure 3.1(b) can be broken by putting a bubble on the edge (b, c). The bubble is drawn as a star symbol. The cycle time of the result graph is equal to 14 units of time. Remember that min-delay retiming achieves cycle time of 16 units. However, recycling does not come for free. The insertion of bubbles decreases the *throughput* of the ES. The throughput is defined as the average number of valid data that circuit produces each clock cycle. As an example, the throughput of the circuit in Figure 3.5(b) is equal to 2/3 because the upper-most directed cycle (a, b, c, d, e) has latency equal to three but only two valid data. The throughput of LID can be statically analyzed and improved [18].

In [19] it was proposed to combine recycling with retiming to optimize the performance of ESs. The first and second contributions of this work further develop a similar idea.

3.2.2 Synchronous elastic flow

In [38] an alternative implementation to LID has been proposed. To implement an EB a flip-flop is decoupled into master and slave latches and the slave latch is used as a buffer for the control logic [89]. The automated design flow, Synchronous Elastic

Flow (SELF), has been proposed for converting synchronous design to elastic.

Figure 3.7 depicts the logic of the EB implementation in SELF. A node may have several inputs (join node) and several outputs (fork node). In order to handle such nodes EB controllers should be connected by join/fork controllers. Their implementations are also proposed in [38]. The variable latency nodes can be easily handled in this design. Later, SELF was extended to handle *early evaluation* nodes and *anti-tokens* counterflow [37], also it was formally verified.

In [26] a detailed comparative analysis of SELF and LID has been done. In the experiments that are presented in Chapter 5 the Verilog simulation uses SELF based design.

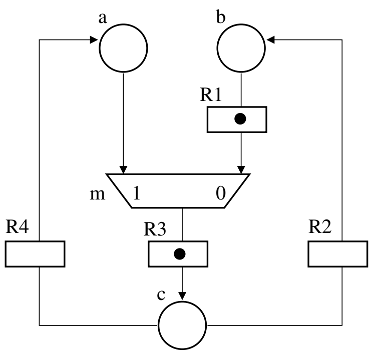
There are several other implementations of synchronous elastic protocol have been proposed [27,70]. The paper [104] provides a framework for formal verification of different implementations of latency insensitive protocols. For example, it is shown how the property of sequential equivalence can be verified. Two systems are *sequentially equivalent* if their valid traces (traces with all bubbles removed) are equal.

3.2.3 Early evaluation

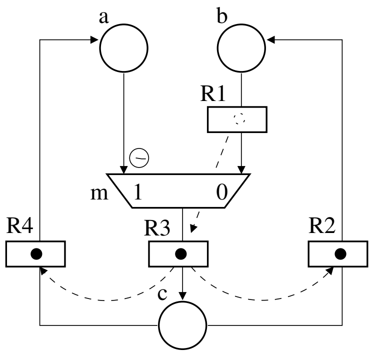
Conventional ESs are based on *late evaluation*: the computation is initiated only when all inputs are available. Sometimes, this requirement is too strict; for example, once a multiplexer received a select signal, it is sufficient to wait for the selected data channel to produce a token, the other data channels are "don't care". *Early evaluation, EE,* takes an advantage of this flexibility to improve the performance of ESs.

Figure 3.8 exemplifies the idea; Figure 3.8(a) represents a simple ES with four computational nodes, a, b, c and m. Node m represents a multiplexer (mux). The select signal of the multiplexer is omitted for simplicity. The one and zero branches of the mux are labeled with 1 and 0 respectively. As before, rectangles with dots represent initialized EBs, while empty rectangles represent bubbles. Assume that the select signal of the mux in Figure 3.8(a) is equal to zero, then, all required data are available for m to produce a new token. Figure 3.8(b) shows the configuration that is produce when m fires, dashed lines show the movements of the tokens. Notice, that a negative token (anti-token), represented as a circle with the sign of a minus inside, has been generated in one branch of the mux. This anti-token is stored in a special buffer of the control logic [37]. The forward delay of such a buffer is equal to zero. The goal of the anti-token is to remember that the input data that will arrive later must be discarded. In our example this happens in the next clock cycle, as it is shown in Figure 3.8(c). Notice, that the total number of tokens at each directed cycle is always constant. The directed cycle ($a \to m \to c \to a$) has one token and the directed cycle ($b \to m \to c \to b$) has two tokens.

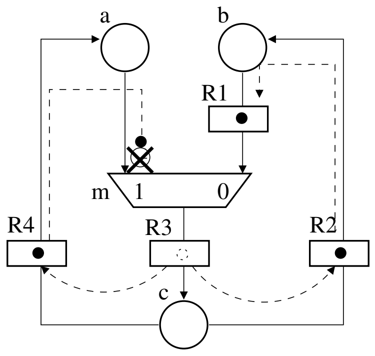
Anti-tokens can be *passive*, waiting for the token to arrive (as in Figure 3.8), or



(a) ES with a Mux



(b) Step 1



(c) Step 2

Figure 3.8: Early Evaluation.

active, traveling backwards through the control until they meet a token. Recently, different schemes to handle EE have been proposed [2,9,24,25,37,67,93,101]. But, to the best of our knowledge, there were not formal methods for performance optimization using concept of EE.

How the performance of an ES with EE can be evaluated? The late evaluation throughput of the ES depicted in Figure 3.8(a) is equal to 1/2. This is because the directed cycle ($a \to m \to c \to a$) has one token and two EBs. Let us assume that the select signal of the mux m is always available and selects zero branch of m with the probability $0 \leq \alpha \leq 1$. Consequently, the branch one is selected with the probability $1 - \alpha$. The behavior of ESs with early evaluation can be modeled using *Markov chains* [109] (see Chapter 5 for examples). The Markov chain analysis allows to calculate exactly the throughput of an ES with EE nodes under assumption that each branch selection is independent on the previously selected branches. Whether such assumption impose big restrictions on the performance evaluation is out of the scope of this work. In our example the throughput is equal to:

$$(1 + \alpha)/(2 + \alpha).$$

One may suppose that the throughput of the ES in Figure 3.8(a) is equal to the average of the throughputs of two directed cycles, that is

$$1/2 \cdot (1 - \alpha) + 2/3 \cdot \alpha.$$

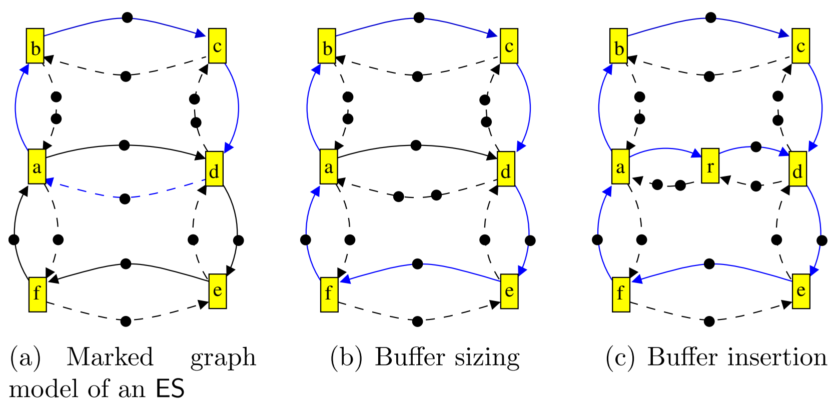
But as it can be seen, this value is not equal to the throughput of the ES.

Markov chain analysis provides exact value of the throughput of an ES with EE nodes and memoryless branch selections. The problem is that the number of the states in the Markov chain grows exponentially with the number of EBs in the ES. On the other hand, Markov chain analysis is not a very convenient tool for building algorithms for the performance optimization. In [59] marked graphs with early evaluation nodes are introduced and an LP model is provided to estimate the throughput of such marked graphs. Chapter 5 uses the results of [59] to optimize the performance of ES with EE nodes.

3.2.4 Slack matching and buffer sizing

There is a phenomena that may degrade the performance of ES. The performance degradation happens when stall event is propagated backward. The main source of this problem are *unbalanced fork-join paths* [4] or *re-convergent paths* [23, 34] in an ES.

Let us exemplify the problem using the ES depicted in Figure 3.9(a). Here we use a *marked graph* model of ES. For more details about marked graphs refer to Section 2.5, for our purposes marked graphs can be treated as usual directed weighted graphs.



(a) Marked graph model of an ES (b) Buffer sizing (c) Buffer insertion

Figure 3.9: Illustration of buffer sizing and slack matching optimization techniques.

The vertices of the graph represent the computational blocks of ES (rectangles in the picture). Each EB is represented as a pair of edges: the *forward* edge and the *backward* edge. Forward edges are continuous lines, they model the data propagation. The number of black dots on a forward edge corresponds to the number of tokens in the EB. Backward edges are dashed lines, they model the back-pressure propagation. The number of tokens on a backward edge corresponds to the number of free registers in the buffer of the control logic. For example, EB (a, b) has no tokens (it is a bubble) and has two free registers.

The sum of the tokens at the forward and backward edge of each EB is a constant. In our example, it is always equal to two. This value represents the *capacity* of the EB. Capacity gives the maximum number of valid data an EB may store before the stop bit for its input will be asserted. Two is the minimal capacity an EB must have: one register is for usual operation and the one register to store the input token if the output asserts the stop bit. Without the extra register it is not always possible to guarantee the correct behavior of ESs. Using the marked graph model, the performance of the ES can be calculated as the *minimal cycle ratio* tokens/delay [92].

Let us assume that the delay of forward and backward edges of each EB in Figure 3.9(a) is equal to one. Such a model corresponds to a synchronous ES. The minimal delay of data propagation from input of an EB to its output is equal to one clock cycle. The same is true for stop bit propagation. Then, for example, the directed cycle $(a \to b \to c \to d \to e \to f \to a)$ has four tokens and its total delay is equal to six. The total delay is calculated as the sum of the delays of each edge of the cycle. Thus, the ratio tokens/delay for this cycle is equal to $4/6 = 2/3$. The minimal cycle ratio tokens/delay in our example is provided by the directed cycle $(a \to b \to c \to d \to a)$ and it is equal to $1/2$. Thus, the throughput of the ES can be estimated as $1/2$. This means, that in infinite execution the ES will produce one valid data every two time units, on average. The cycle, that has minimal cycle ratio token/delay is called *critical cycle.* Notice, that the critical cycle in the example contains the backward edge (d, a).

Let us assume that the capacities of all EBs in the example are equal to infinity. In the terms of the marked graph model this means that the weights of the all backward edges are equal to infinity. Then, it can be verified that the minimal ratio tokens/delay of the marked graph depicted in Figure 3.9(a) is equal to 2/3 ant the critical cycle is $(a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow a)$. This provides 33% performance improvement comparing with the original throughput. However, the same throughput can be achieved by setting only the capacity of the EB (a, d) be equal to three and leaving the rest of the EBs unchanged. Figure 3.9(b) shows the result marked graph.

One may try to find a minimal capacity that is sufficient to add for each EB to remove all backward edges from the critical cycles of the ES. This problem is known as *buffer sizing* problem. It can be shown that the buffer sizing is NP-complete (see, for example, Theorem 6.5.1). Still, most of the instances of this problem can be solved very fast because in practice a very few EBs should be resized to achieve maximal throughput.

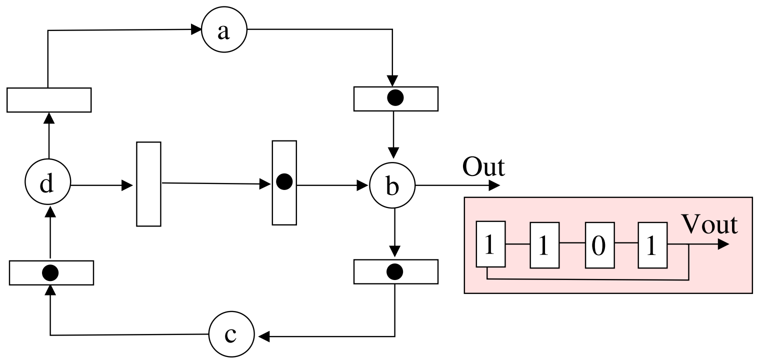
Re-convergent paths can be balanced by inserting bubbles into short (fast) branches. This technique is well known in the area of asynchronous design [73, 108] where it is called *slack matching.* In this work we refer to this transformation as *buffer insertion.* Figure 3.9(c) illustrates the buffer insertion transformation applied to the marked graph from Figure 3.9(a). The bubble (a, r) has been inserted between the computational nodes a and d. Again, the throughput of the result ES has been increased to 2/3.

Often, buffer insertion allows to achieve the same throughput as buffer sizing. The third contribution of this work provides an example of a marked graph when buffer insertion is unable to achieve the same throughput as buffer sizing does. After this, a general formulation which combines buffer sizing and buffer insertion for throughput optimization of ESs is proposed.

The main advantage of the buffer insertion with respect to buffer sizing is that the control logic of EB remains as simple as shows Figure 3.7. While the implementation of EBs with capacity greater than two requires extra effort. For example, it can be obtained by combination the implementations proposed in [70] and [28]. A weakness of buffer insertion is that it increases the forward latency of the ES and may achieve less throughput than the buffer sizing. Since a lot of research devoted to these transformations have been done, a short overview of some publications follows next.

Related work

Buffer sizing was proposed as a technique for optimization of synchronous elastic systems in [71]. In [71] a mixed integer linear programming model is proposed to



Cycle \ Node	a	b	c	d
1	0	1	1	1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	0
5	0	1	1	1

The valid bit in the first 5 clock cycles

Figure 3.10: ES with two bubbles. The capacity of all EBs is equal to two. The table on the right shows whether each computational block produced token or bubble during first five clock cycles.

maximize the throughput of ESs by using buffer sizing. In [34], a polynomial time heuristics for buffer sizing is proposed. Moreover, it was observed that the throughput achievable with the simultaneous use of buffer sizing and buffer insertion may be greater than the throughput achievable only with buffer insertion.

Buffer insertion has a long history in the area of asynchronous design, where it is usually called slack matching. In [108], it is shown that the performance of a self-timed ring is maximized when it is balanced with respect to the ratio tokens/bubbles, and a quantitative approach to calculate such an optimal balance is presented. In [14], a precise linear approximation for the performance analysis of iterative computations in self-timed rings is presented. The model is based on event-rule systems and it is closely related to previous models based on Petri nets [91,92]. In [45], an iterative algorithm for slack matching in asynchronous systems is presented. The algorithm does not guarantee the minimal number of extra registers will be used to achieve maximal throughput. In [4, 90], algorithms for buffer insertion in choice-free asynchronous systems are proposed, the algorithms are based on linear programming. The marked graph model of ESs that is presented in this section is used in the most of the papers about slack matching of asynchronous design.

3.2.5 Scheduling elastic designs

In the infinite execution the behavior of an ES with only late evaluation nodes is periodic [7,23]. Thus, if one would find such a period for each node, then the control logic for the ES can be reduced to a simple ring of appropriately initialized registers.

This, in turn, removes the necessity of routing of extra wires that elastic protocol adds. It is important, since high wire congestion can make the design malfunctional. The drawback of the approach is that the schedule itself can be long, which introduces area overhead. It is very likely that many of the nodes will have the same or shifted scheduling. Thus, one may share such schedulings to reduce the area overhead [21]. However, only scheduling of the nodes that are located close to each other on the chip can be shared, otherwise, routability problems may again appear [21]. A more substantial drawback of the scheduling follows from the fact that scheduling does not exist for ESs with early evaluation nodes. The behavior of such ESs is not periodic. On other hand, without early evaluation the performance of an ESs may hardly be improved compared with the original circuit (see Chapter 4).

Let us illustrate the idea of the scheduling with an example. Figure 3.10 shows a simple ES with four computational blocks and six EBs. Only two EBs are not initialized, the rest have a token. Let us simulate the behavior of this ES assuming synchronous model. In the first clock cycle all nodes have valid inputs except node a. Node a produces a *bubble* in this cycle. This corresponds to the first line of the table in the right part of the figure. In the second clock cycle node b produces a *bubble*, since both its input EBs have no token. Then, bubble is propagated as it is shown in the table. After four cycles the circuit is in the initial state. Thus, in the next four clock cycles the behavior shown in the table will be repeated. Each node produces three valid data at each period which gives throughput of 3/4. The reader may verify that this value is equal to the ratio tokens/latency of the directed cycles $(d \to b \to c \to d)$ and $(d \to a \to b \to c \to d)$.

The actual implementation of this ES does not need valid and stop bits at all. The only thing we should remember is the order in which node b produces valid data, i.e., its schedule. The schedule of the node b is an automaton computing the following sequence: $(1\ 0\ 1\ 1)^*$. Such automaton can be implemented as a ring of four registers with initial state (1 0 1 1) respectively, as it is shown in the figure. In general, finding a static scheduling is not always so easy, as in the example. Sometimes EBs with rational capacities are required [7].

3.3 Conclusions

Synchronous elastic design is a relatively new field, but a lot of work has been already done here. However, the most part of this work is focused on the design aspects and local transformations that might improve the performance of ESs and a very few algorithms for the global performance optimization of ESs are provided.

Chapter 4

Retiming and Recycling

This chapter presents the first contribution of this dissertation. It is a general formulation that combines retiming and recycling for performance optimization of ESs. It can also cover C-slow retiming as a particular case. The proposed model is based on mixed integer linear programming. A set of experiments has been designed to show the benefits that can be obtained by combining retiming and recycling with respect to applying these two techniques separately. The contribution presented in this chapter has been published in [11].

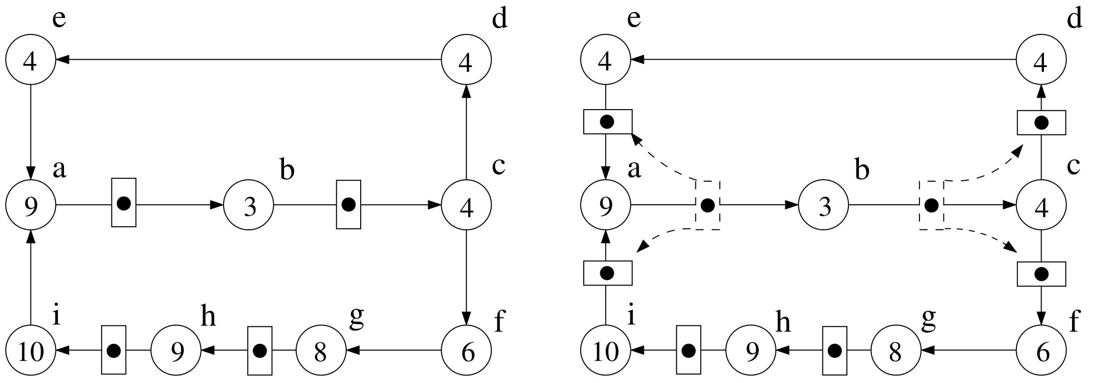
The model can be extended to handle several other transformations of ESs. Some of these extensions are discussed in the next chapter.

4.1 Introductory example

In this chapter an ES is modeled as a *retiming and recycling graph* (RRG). The model is an extension of a retiming graph model and it can handle early evaluation, retiming of Elastic Buffers (EBs) and insertion of empty EBs (recycling). This chapter focuses only on the ESs with *late evaluation* nodes.

Figure 4.1(a) shows an RRG for the retiming graph depicted in the Figure 3.1(a). As before, vertices represent combinational nodes and are annotated with their combinational delays. The rectangles with dots inside represent initialized EBs. In traditional synchronous design all registers have to be initialized, but in an ES this is not a requirement, bubbles may appear in the initial configuration.

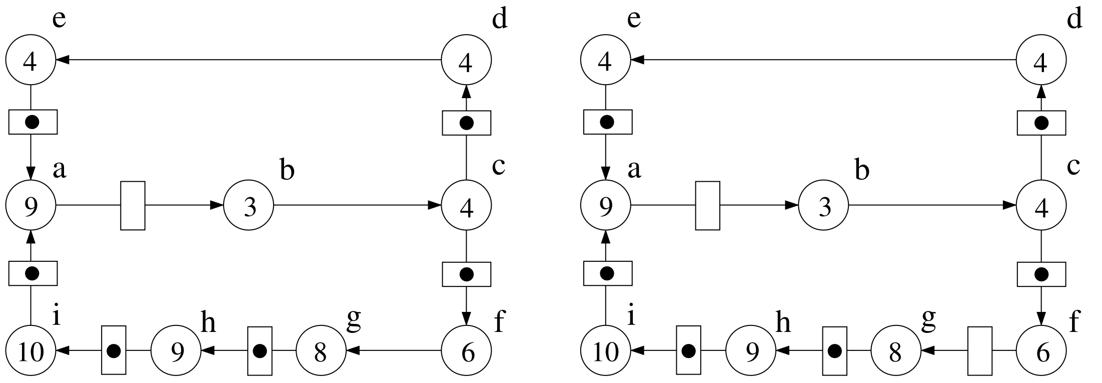
In order to distinguish initialized EBs from bubbles, one more edge property is introduced to the RRG compared with the retiming graph. In an RRG each edge has two weights: the number of tokens and the number of EBs assigned. For example, the edge (a, b) in Figure 4.1(a) has one EB and one token. The combinational path of RRG is defined similar to the combinational path of retiming graph (section 3.1) substituting the registers by the EBs.



(a) RRG of the reiming graph from Figure 3.1(a)

(b) Min-delay retiming configuration (see Figure 3.1(b))

Figure 4.1: RRGs for the retiming graphs depicted in Figure 3.1.



(a) Recycling, step 1

(b) Recycling, step 2

Figure 4.2: Separate application of retiming and recycling.

Let us exemplify how retiming and recycling transformations can be combined to optimize the performance of ESs. Consider the RRG depicted in Figure 4.1(a) and let us apply only retiming transformation first. This results in the min-delay retiming configuration depicted in Figure 4.1(b). The dashed arrows show the movement of the EBs. The cycle time of the min-delay retiming configuration is equal to 16 units of time with the critical combinational path $a \rightarrow b \rightarrow c$. Now let us break the critical combinational path by putting a bubble on the edge (a, b). This leads to the cycle time of 14 with the critical path $f \rightarrow g$. Figure 4.2(a) shows the result RRG, the bubble is represented by an empty rectangle. Remember, the resulting circuit does not produce a valid data at each clock cycle. The directed cycle $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow a$ has three EBs and two tokens. Thus, it may produce two tokens every three clock cycles in the best case. The throughput of this RRG is equal to 2/3.

The main metrics of the RRG performance is its *effective cycle time*, which is defined as the ratio of cycle time by the throughput. The effective cycle time of the RRG in Figure 4.2(a) is equal to $14/0.667 = 21$ units of time. This means, that the corresponding ES produces new valid data every 21 units of time, which is worse than the effective cycle time of the min-delay retiming configuration. The effective cycle time of a conventional synchronous circuit is equal to its cycle time, since its throughput is always equal to one.

Let us continue the recycling of the first example and put a new bubble on the edge (f, g), resulting in the RRG depicted in Figure 4.2(b). The cycle time of this configuration is equal to 10 units of time. The throughput is still 2/3, because the directed cycle $a \rightarrow b \rightarrow c \rightarrow d \rightarrow f \rightarrow g \rightarrow h \rightarrow i \rightarrow a$ has four tokens and five EBs. The cycle with minimal token to EB ratio (*critical cycle*) is still the topmost cycle $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow a$. The effective cycle time of this RRG is equal to $10/0.667 = 15$ units of time. This is better than the min-delay retiming cycle time. No more bubbles can decrease the cycle time since node i has combinational delay of 10 units of time.

Can we achieve the same performance with only one bubble insertion? Yes, we can. Starting from the RRG depicted in Figure 4.1(a) we put a bubble on the edge (f, g) and retime node c by one. The result in configuration with the cycle time of 12 units of time depicted in Figure 4.3. There are two critical combinational paths in this configuration, they are $a \rightarrow b$ and $c \rightarrow d \rightarrow e$. The throughput is equal to 4/5, the critical cycle has four tokens and five EBs. The effective cycle time is equal to $12/0.8 = 15$, the same as for the RRG 4.2(b). But this configuration has fewer tokens and EBs, which, in turn, would lead to the implementation with less area.

In this simple example, the separate application of retiming and recycling (Figures 4.1 and 4.2) leads to the RRG with the minimal effective cycle time (Figure 4.2(b)). However, the combination of these two techniques might achieve a better performance than their separate application. In the experiments that were performed to validate this contribution it happened in half of the test cases. Unfortunately, it is

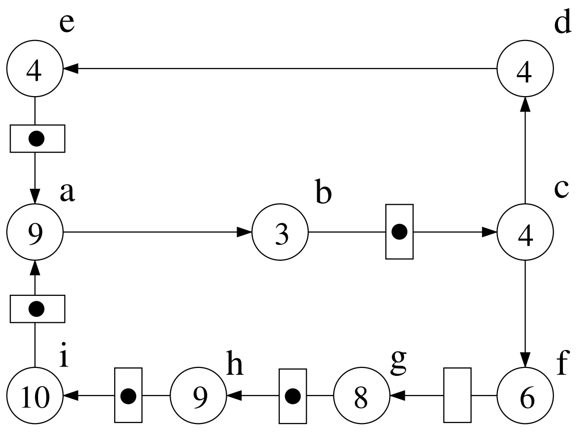


Figure 4.3: Retiming and recycling solution.

difficult to find a small RRG with such properties to be used as introductory example.

The main contribution of this chapter is an MILP-based model to find minimal effective cycle time of an RRG by using retiming and recycling transformations. Henceforth and for the sake of brevity we will refer to the retiming and recycling transformation as R&R.

4.2 Basic definitions

This section formalizes basic concepts by giving formal definitions of RRG and its cycle time and throughput.

Definition 4.2.1 (RRG) *A* Retiming and Recycling Graph *(RRG) is a tuple* (S, β, T, R)*, where*

- $S = (N, E)$ *is the underlying multi-graph of the ES, N is the set of nodes and E is the set of edges.*
- $\beta : N \to \mathbb{R}^+$ *assigns combinational delay to each node.*
- $T : E \to \mathbb{Z}^+$ *is the number of the tokens on each edge in the initial state.*
- $R : E \to \mathbb{Z}^+$ *is the number of EBs on each edge. The condition $R \geq T$ must hold.*

We have already seen plenty of RRGs. For example, the edge (b, c) of the RRG depicted in Figure 4.3 has one token, $T(b, c) = 1$, and one EB, $R(b, c) = 1$; combinational delays of nodes b and c are equal to three and four units of time, respectively; that is $\beta(b) = 3$ and $\beta(c) = 4$. Notice that any retiming graph is also an RRG, with $R = T$ for all edges.

Definition 4.2.2 (Retiming and recycling configuration (RC)) *Given an RRG, a* retiming and recycling configuration*, (RC), is a pair of vectors T and R.*

Notice, that R&R of an RRG changes only its RC.

Definition 4.2.3 (Retiming vector) *A* retiming vector *$r \in \mathbb{Z}^{|N|}$ is a function $V \to \mathbb{Z}$ such that for every edge $e = (u, v)$ of an RRG transforms $T(e)$ to $T'(e)$ as follows:*

$$T'(e) = T(e) + r(v) - r(u).$$

After applying retiming to RRG the number of EBs on each edge must be adjusted such that the inequality $R \geq T$ holds. A way to achieve this is by applying the same retiming r to R that has been applied to T. It might seem that it is more convenient to define retiming of an RRG as a transformation that modifies both T and R simultaneously. However, the proposed definition is more convenient for our purposes since it helps to separate retiming from recycling.

The recycling on edge e can be achieved by incrementing $R(e)$. Let us notice that it is never worthwhile to increment $R(e)$ if $T(e)$ is positive; such transformation does not modify the cycle time of the RRG, but may decrease its throughput. Hence, the only candidates for recycling are edges without EBs (wires).

We now define some concepts related to timing and performance in RRGs.

Definition 4.2.4 (Combinational paths and cycle time.) *Given an RRG, a* combinational path *P is a sequence of edges $n_0 \xrightarrow{e_1} n_1 \xrightarrow{e_1} \cdots \xrightarrow{e_k} n_k$ such that $R(e_i) = 0$ for all edges in the path. The delay of the combinational path is*

$$\beta(P) = \sum_{n_i \in P} \beta(n_i).$$

The cycle time of RRG, τ, is the maximum delay of all combinational paths.

Now let us formally define the throughput of an RRG.

Definition 4.2.5 (Throughput) *Given an RRG, and a directed cycle c, the cycle ratio (ratio)* tokens/EBs *of c, Θ_c, is defined as*

$$\Theta_c = \frac{\displaystyle\sum_{e \in c} T(e)}{\displaystyle\sum_{e \in c} R(e)}$$

The throughput Θ of an RRG is the minimum ratio tokens/EBs over all cycles in the RRG, that is,

$$\Theta = \min_{\textit{for all cycles } c} \Theta_c.$$

At the first glance it is not evident why this definition corresponds to the real behavior of ESs. A couple of papers are devoted to the formal demonstration of this fact both in asynchronous [92] and synchronous [72] domains.

The throughput is the number of tokens that can be processed by a node per cycle. Since R&R modifies only RC of an RRG, next we will write $\tau(\mathrm{RC})$ and $\Theta(\mathrm{RC})$ to refer to the cycle time and the throughput of a given RC.

If an RC has no bubbles, then $\Theta(\mathrm{RC}) = 1$, as in Figure 4.1(a). The RRG depicted in Figure 4.3 has cycle time equal to 12 units of time. The cycle ratios tokens/EBs for the top and bottom cycles are equal to 1 and 4/5, respectively, therefore $\Theta(\mathrm{RC}) = 4/5$. In general, retiming always produces RCs with $\Theta(\mathrm{RC}) = 1$, whereas recycling produces RCs with $\Theta(\mathrm{RC}) < 1$. In case of c-slow retiming, the RCs with $\Theta(\mathrm{RC}) = 1/c$ are produced. We now define the main performance metrics for an RC.

Definition 4.2.6 (Effective cycle time) *The* effective cycle time *of an RC is defined as*

$$\xi(RC) = \tau(RC)/\Theta(RC).$$

The *effective cycle time* of an RC can be treated as a cycle time of the classical synchronous circuit that could process data at the same rate as its elastic version. The effective cycle time of the RRG depicted in Figure 4.3, is equal to $\xi = 12{\cdot}5/4 = 15$. The main contribution of this chapter is a method to find an RC with the minimal effective cycle time by using R&R. The next result reduces the margins for improvement for RRGs with only late evaluation nodes.

4.2.1 The lower bound on the effective cycle time

By using only the definition of the effective cycle time, the following lemma can be proved. It is a negative result since it reduces the margin for the performance improvement of R&R.

Lemma 4.2.1 (The lower bound on effective cycle time) *For each RC the following inequality holds:*

$$\xi(RC) \geq \max_{c \in C} \frac{\sum\limits_{v \in c} \beta(v)}{\sum\limits_{e \in c} T(e)},$$

where C is the set of all directed cycles of the corresponding RRG.

Proof: A known result from [84] indicates that for any cycle c

$$\tau(\mathrm{RC}) \geq \frac{\sum\limits_{v \in c} \beta(v)}{\sum\limits_{e \in c} R(e)} \tag{4.1}$$

To see this, assume that a directed cycle c contains k EBs which break the cycle into k combinational parts $p_1, p_2, \dots, p_k$. The local cycle time is provided by the longest combinational path, hence $\tau(\mathrm{RC}) \geq \beta(p_i), i = 1, \dots, k$ and consequently $\tau(\mathrm{RC}) \geq p_{max}$, where $p_{max} = \max_i \beta(p_i)$. Now for the cycle c we have the following chain of inequalities:

$$\frac{\sum_{v \in c} \beta(v)}{\sum_{e \in c} R(e)} = \frac{\sum_{i=1}^{k} \beta(p_i)}{k} \leq p_{max} \leq \tau(\mathrm{RC}).$$

which proves (4.1).

By Definition 4.2.5 of the throughput we have:

$$\Theta(\mathrm{RC}) \leq \Theta_c = \frac{\sum_{e \in c} T(e)}{\sum_{e \in c} R(e)} \tag{4.2}$$

Gathering (4.1) and (4.2) together we have:

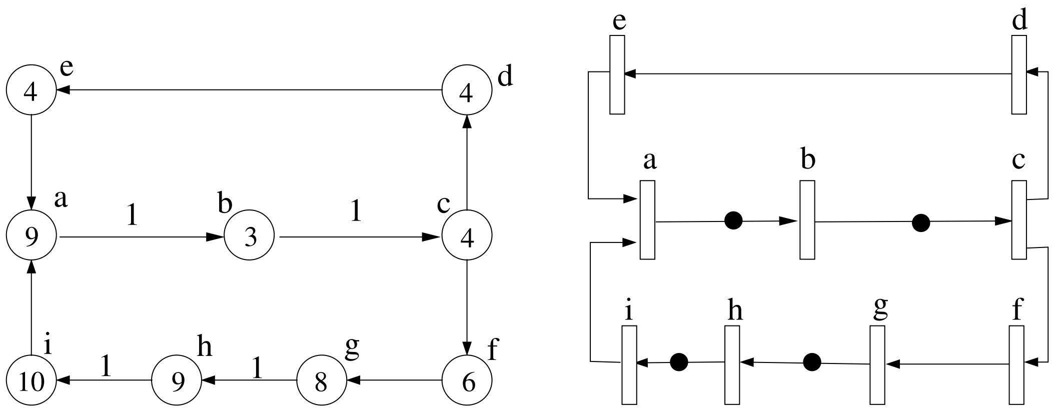
$$\xi(\mathrm{RC}) = \frac{\tau(\mathrm{RC})}{\Theta(\mathrm{RC})} \geq \frac{\sum_{v \in c} \beta(v)}{\sum_{e \in c} T(e)}$$

for each directed cycle c, which completes the proof. ∎

Thus, the lower bound for the effective cycle time coincides with the lower bound for *min-delay retiming* [84]. However, R&R offers more opportunities than retiming to approach this bound. For example, the lower bound on ξ in RRG depicted in Figure 4.1(a) is equal to 12.25 units of time. With retiming, the cycle time of 16 units of time can be achieved, while R&R can find an RC with effective cycle time equal to $\xi = 15$. In [84] an upper bound for min-delay retiming cycle time is also provided:

$$\tau \leq \max_{c \in C} \frac{\sum_{v \in c} \beta(v)}{\sum_{e \in c} T(e)} + \beta_{max}, \quad \beta_{max} = \max_{n \in N} \beta(n).$$

Thus, the margin for the improvement for R&R is within β_{max}. If an RRG has well-balanced delays, then for R&R it is difficult to improve the min-delay retiming cycle time. Also let us notice, without a formal proof, that the upper bound on the performance gain that can be obtained with R&R with respect to the min-delay retiming cycle time is equal to 50%. This is because by inserting a bubble the best we can do is to decrease the cycle time twice and preserve the throughput. In the experiments that were done to verify the proposed optimization technique, retiming always



(a) Retiming graph (b) Corresponding MG

Figure 4.4: Analogy between retiming graph and marked graph.

provided a configuration with minimal effective cycle time. Another observation that directly follows from the Lemma 4.2.1 is that clock skew optimization always provides the same or better performance than R&R. Fortunately, all these restrictions do not apply to RRGs with early evaluation nodes.

4.3 Retiming and marked graphs

Retiming has a strong analogy with the firing rule of *marked graphs* (MG). Please, refer to Section 2.4 if you are not familiar with MGs. Some of the results in the theory of MGs can be reused in the context of retiming, thus providing an essential support to make a new contribution in the area of the retiming.

The analogy between MG and retiming is the following:

- The retiming graph of a circuit is isomorphic to a marked graph: each combinational block corresponds to a node (transition); each connection corresponds to an edge (place).
- The registers in the retiming graph are represented by tokens in the MG.
- The firing rules of a MG coincide with the backward retiming rules: each time a node is retimed, registers are removed from the input edges and added to the output edges.

Figure 4.4(a) shows retiming graph from Figure 4.1(a); Figure 4.4(b) shows the corresponding MG. We next give a retiming interpretation of the liveness and reachability properties of MGs.

Retiming interpretation of liveness (Property 2.5.1). "An MG is live iff every directed cycle $\mathbf{c}$ is marked positively at given marking m_0". Every cycle should have at least one register to avoid combinational cycles in the circuit net-list.

Retiming interpretation of reachability (Property 2.5.2) . This property has two directions. One direction: "if a marking m is reachable then $m(\mathbf{c}) = m_0(\mathbf{c})$ for every directed cycle of the MG", corresponds to a well-known result in retiming: a valid retiming preserves the number of registers at each cycle. The other direction: "if for given marking m and every directed cycle c of MG it hold that $m(\mathbf{c}) = m_0(\mathbf{c})$ then m is a reachable marking" provides a new result for the theory of retiming: *if an assignment of registers has the same number of registers at each cycle as the initial circuit, then the assignment can be achieved by the retiming.*

Thus, using Property 2.5.2 and the analogy between MGs and retiming we obtain the following:

Theorem 4.3.1 *In a strongly connected retiming graph:*

- *a registers assignment T' can be achieved by retiming if and only if T' preserves the number of registers at each directed cycle of the retiming graph.*
- *any retiming configuration can be achieved with only forward retiming moves.*

Proof: The first part of the Theorem is a direct application of the reachability property. The demonstration of the second part follows from the observation that a firing vector is non-negative (see Definition 2.4.5). ■

For example, in the retiming graph from Figure 4.4(a) the backward retiming $r(a) = 1, r(v) = 0, v \neq a$ is equivalent to the forward retiming $r(a) = 0, r(v) = -1, v \neq a$.

The result of this section is standalone and does not have any relationship with the rest of this chapter. Next we will continue the developing of the algorithm to find an RC with the minimal effective cycle time for a given RRG.

4.4 Basic MIP model for retiming and recycling

First, we will show how the cycle time and throughput of an RC can be modeled with linear inequalities. This, in turn, allows us to derive a mixed integer non-linear programming (MIP) model to find an RC with the minimal effective cycle time that can be achieved with the R&R. The obtained MIP is difficult to solve with available solvers. Fortunately, the special structure of the problem allows to solve it by solving just a few MILP problems. Several variations and improvements of the proposed models are also discussed.

4.4.1 Combinational path constraints

In order for an RC to meet a cycle time τ, all combinational paths of the RRG must have delay less than or equal to τ. In the retiming problem, these constraints are formulated by using the matrices W (minimum latency) and D (maximum delay) [66]. Unfortunately, this formulation is not valid for the R&R problem, since the number of EBs between any pair of nodes can be changed by inserting arbitrary number of bubbles. Next, a different set of linear inequalities is proposed to formulate the combinational path constraints. These constraints are based on the concept of *arrival time* of each node of an RRG.

For every node n and every edge e of an RRG, the variables $t^{in}(n)$ and $t^{out}(e)$ are defined. The variable $t^{in}(n)$ represents the longest delay from any EB to the node n, including the delay of the node n. The variable $t^{out}(e)$ represents the longest delay from any EB to the exit of edge e. If e has no EBs, then $t^{out}(e) = t^{in}(u)$, where u is the source node of edge e, otherwise $t^{out}(e) = 0$, since the exit of edge e represents the beginning of a combinational path. This definition of t^{in} and t^{out} can be represented by the following constraints:

$$t^{in}(n) \geq t^{out}(e) + \beta(n) \qquad \forall e = (w, n) \tag{4.3}$$

$$t^{out}(e) \geq t^{in}(w) - \tau^* R(e) \qquad \forall e = (w, n) \tag{4.4}$$

$$t^{out}(e) \geq 0 \qquad \forall e, \tag{4.5}$$

$$t^{in}(n) \leq \tau \qquad \forall n. \tag{4.6}$$

The constraint (4.3) indicates that the delay of the longest path arriving at the node n is at least equal to the delay of the path terminating at the node w plus the delay of the node n. The constraint (4.4) transfers the length of the path to the exit of the edge. In case the edge has some EB ($R(e) > 0$), a new combinational path starts at edge e and $t^{out}(e) = 0$. τ^* is a constant large enough to guarantee its value to be larger than any possible value of τ. It is sufficient for τ^* to take the value of the cycle time of the original RC. For example, the node a of the RRG depicted in Figure 4.1(a) has the following cycle time constraints:

$$\begin{array}{l}
t^{in}(a) \geq t^{out}(e, a) + 9, \\
t^{out}(e, a) \geq t^{in}(e) - 21 \cdot 0, \\
t^{in}(a) \geq t^{out}(i, a) + 9, \\
t^{out}(i, a) \geq t^{in}(e) - 21 \cdot 0, \\
t^{out}(i, a) \geq 0, t^{out}(e, a) \geq 0, \\
t^{in}(a) \leq 21.
\end{array}$$

Here, the letter e refers to the node of the RRG, 21 is the cycle time of the RRG. With τ^* being constant, all the inequalities are linear. Inequalities (4.4) and (4.5) is a standard trick of mixed integer linear programming to model the OR relation between two groups of the constraints. In this case the following relation has been modeled:

$$t^{out}(e) \geq \begin{cases} t^{in}(u), & if R(e) \geq 1, \\ 0, & otherwise. \end{cases}$$

EB combinational delays. The previous constraints can be modified to account for the extra combinational delays that introduce EBs. If we call β_R the extra combinational delay of an EB, we can add β_R to the delay of each combinational path by assigning this delay to the beginning of the path. This is achieved by simply adding the constraint $t^{out}(e) \geq \beta_R$, for all edges of the RRG. Henceforth, the set of constraints (4.3)-(4.5) for a given RC and cycle time τ will be represented by a predicate $\texttt{Path\_Constr}(\mathrm{RC}, \tau)$.

Lemma 4.4.1 *For a strongly connected synchronous RRG the cycle time, $\tau(RC)$, is less than or equal to a given constant τ iff* $\texttt{Path\_Constr}(RC, \tau)$ *are feasible.*

Proof: $\Rightarrow$ Assume that $\tau(\mathrm{RC}) = \tau$. For each node n and each edge $e = (w, n) \in E$, we define $t^{in}(n)$ and $t^{out}(e)$ as follows:

$$\begin{aligned} t^{in}(n) &= \delta(n) + \text{delay of the longest combinational path arriving at } n, \\ t^{out}(e) &= \begin{cases} t^{in}(w) & \text{if } R(e) = 0, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

It is easy to check that the inequalities (4.3)-(4.5) hold.

$\Leftarrow$ Assume that the constraints (4.3)-(4.5) have a feasible solution

$$(t_1^{in}, \ldots, t_q^{in}, t_1^{out}, \ldots, t_m^{out})$$

for some τ, q and m are equal to the number of nodes and edges in the RRG respectively. Let $P = v_1, v_2, \ldots, v_{k-1}, v_k$ be a combinational path with total delay equal to $\beta(P)$. Then the following chain of inequalities holds:

$$\begin{array}{ll} t^{in}(v_1) \geq \beta(v_1) & \text{(constraint (4.3))} \\ t^{out}(v_1, v_2) \geq t^{in}(v_1) \geq \beta(v_1) & \text{(constraint (4.4))} \\ t^{in}(v_2) \geq t^{out}(v_1, v_2) + \beta(v_2) \geq \beta(v_1) + \beta(v_2) & \text{(constraint (4.3))} \\ \quad \ldots & \\ t^{in}(v_k) \geq \sum\limits_{i=1}^{k} \beta(v_i) = \beta(P). & \end{array}$$

To start the chain of inequalities we use the fact that in a strongly connected graph there are no vertices with input degree equal to zero (*source vertices*). Since $t^{in}(n) \leq \tau$, for any node n, we can conclude that $\beta(P) \leq \tau$ for any combinational path P. ■

To handle non-strongly connected RRGs, extra constraints should be added to $\texttt{Path\_Constr}(\mathrm{RC}, \tau)$ for each source vertex s: $t^{in}(s) \geq \beta(s)$.

4.4.2 Throughput constraints

We next present a set of constraints to guarantee that a specific RC has a throughput greater than or equal to a given constant Θ. The throughput constraints are the following:

$$R(e) \leq x \cdot R_0(e) + \sigma(u) - \sigma(v), \quad \forall e = (u, v) \tag{4.7}$$

Lemma 4.4.2 *There is a real vector σ that fulfills inequality* (4.7) *iff* $\Theta(RC) \geq 1/x$.

Proof:
$\Rightarrow$
Assume that $\Theta(\text{RC}) \geq 1/x$ then, from throughput definition we have

$$\min_{c \in C} \frac{\sum\limits_{e \in c} T(e)}{\sum\limits_{e \in c} R(e)} \geq 1/x,$$

or equivalently

$$\max_{c \in C} \frac{\sum\limits_{e \in c} T(e)}{\sum\limits_{e \in c} R(e)} \leq x.$$

Applying Theorem 2.3.1, the necessary σ is obtained. Similarly other direction can be proved. ■

In a similar way as for the cycle time constraints, let us introduce a predicate `Throughput_Constr`(RC, Θ) for a given RC and constant Θ. The constraints are verified if and only if the throughput of an RC is greater than or equal to Θ.

4.4.3 Basic MIP model

Having cycle time and throughput constraints one can find an RC with the minimal effective cycle time as a solution of the following quadratic programming optimization problem:

$$\begin{array}{ll} \textit{minimize}: \quad \tau \cdot x, & \\ \textbf{subject to:} & \\ \quad R'_0(e) = R_0(e) + r(v) - r(u), & \forall e = (u, v) \\ \quad R \geq R'_0 \geq 0, & \\ \quad \texttt{Path\_Constr}(\text{RC}, \tau), & \\ \quad \texttt{Throughput\_Constr}(\text{RC}, \Theta), & \\ \quad r(n) \in \mathbb{Z}, & \forall n \in N \\ \quad R(e) \in \mathbb{Z}, & \forall e \in E \\ \quad \Theta \geq 0. & \end{array} \tag{4.8}$$

In this formulation, r and R are vectors of integer variables. The model is not linear because of the objective function. The objective function is not convex, thus discarding a possibility to use convex optimization methods to solve it. The current formulation, to the best of our knowledge, can not be solved exactly by any existing solver. Next, it will be shown how the optimization problem (4.8) can be solved by solving just a few MILPs.

4.5 Iterative approach for R&R

Fixing the cycle time or throughput variable in (4.8) the two subproblems can be solved with mixed integer linear programming. Namely, maximizing throughput under a given cycle time constraint, that means the cycle time is upper bounded by a given constant or minimizing cycle time under given throughput constraint, that means the throughput is lower bounded by a given constant. We will construct an iterative procedure that makes use of these subproblems to solve (4.8). Henceforth, we will use $\texttt{RR}(\tau, \Theta)$ to denote all the constraints in (4.8). This notation will also be used to denote a function that returns an RC, RC $:= \texttt{RR}(\tau, \Theta)$.

4.5.1 Min-cycle-time R&R

The min-cycle-time R&R problem can be formulated as follows:

> Given an RRG and a throughput $\Theta > 0$, find an RC with the minimal cycle time such that $\Theta(\text{RC}) \geq \Theta$.

By fixing variable Θ in (4.8), it immediately follows that such an RC is the solution of the following MILP, where Θ is a constant:

$$\texttt{MIN\_CYC}(\Theta) \equiv \begin{array}{l} \textit{minimize}: \quad \tau, \\ \textbf{subject to: } \texttt{RR}(\tau, \Theta). \end{array} \tag{4.9}$$

Similarly as before, $\texttt{MIN\_CYC}(\Theta)$ denotes a function that returns the RC with the minimum cycle time from all those that have throughput greater than or equal to Θ.

4.5.2 Max-throughput R&R

The formulation of the problem is as follows:

> Given an RRG and a cycle time $\tau > 0$, find an RC with maximal throughput such that $\tau(\text{RC}) \leq \tau$.

Fixing variable τ in (4.8) the following mixed integer optimization problem is obtained to find an RC with the maximal throughput:

$$\texttt{MAX\_THR}(\tau) \equiv \begin{array}{l} \textit{minimize}: \quad x, \\ \textbf{subject to: } \texttt{RR}(\tau, \Theta). \end{array} \tag{4.10}$$

4.5.3 Min-area R&R

Fixing both cycle time and throughput variables in (4.8) an RC with the minimal number of the EBs can be found with the following MILP:

$$\begin{array}{l} \textit{minimize}: \sum\limits_{e \in E} R(e), \\ \textbf{subject to: } \texttt{RR}(\tau, \Theta). \end{array} \tag{4.11}$$

4.5.4 Minimum effective cycle time

Now we are ready to formulate the main problem. The formulation is as simple as follows:

> Given an RRG, find a configuration $\mathrm{RC_{min}}$ that can be achieved by R&R such that the effective cycle time $\xi(\mathrm{RC_{min}})$ is minimal.

We will show how the MILP models MIN_CYC(Θ) and MAX_THR(τ) can be combined to find $\mathrm{RC_{min}}$. We first start by a preliminary result required to make the search efficient.

Algorithm 1: function MIN_EFF_CYC_STEP(Θ)

```
1 $\mathrm{RC}_1 \leftarrow$ MIN_CYC($\Theta$)
2 $\mathrm{RC}_2 \leftarrow$ MAX_THR($\tau(\mathrm{RC}_1)$)
3 return $RC_2$
```

Lemma 4.5.1 *$\Theta(RC_{min}) \geq \frac{\beta_{max}}{\tau_{rt}}$ and $\tau(RC_{min}) \geq \beta_{max}$, where β_{max} is the maximum delay of a node and τ_{rt} is the cycle time obtained by min-delay retiming.*

Proof: Every retiming configuration is also an RC. Therefore $\xi(\mathrm{RC_{min}}) \leq \tau_{rt}$. By Definition 4.2.6 of ξ, and using the fact that $\tau(\mathrm{RC_{min}}) \geq \beta_{\max}$,

$$\Theta(\mathrm{RC_{min}}) = \frac{\tau(\mathrm{RC_{min}})}{\xi(\mathrm{RC_{min}})} \geq \frac{\beta_{\max}}{\tau_{rt}}.$$

■

Then, the search for the optimal ξ can be performed by interleaving MIN_CYC(Θ) and

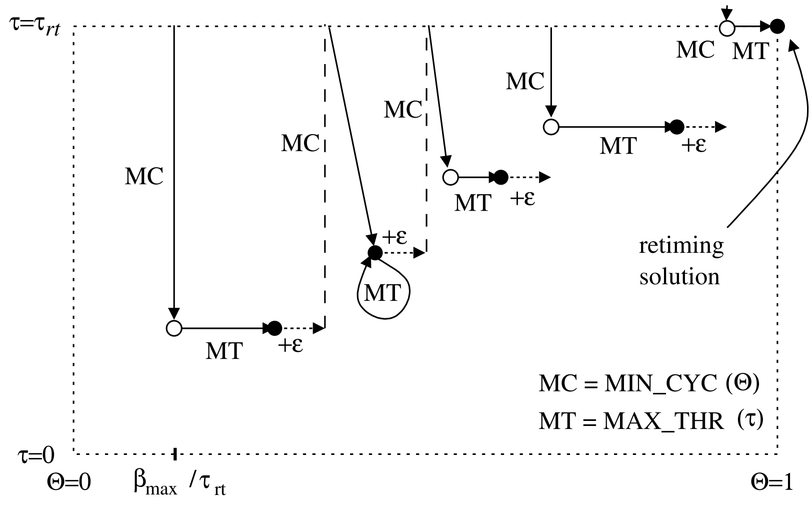


Figure 4.5: Search for minimum effective cycle time.

`MAX_THR`(τ) as Algorithm 1 does. The intuition behind this strategy is the following. Given a target Θ, an RC_1 with minimum cycle time is obtained by `MIN_CYC`(Θ). RC_1 has a throughput not smaller than Θ. After that, another RC_2 maximizing the throughput is obtained by `MAX_THR`$(\tau(\mathrm{RC}_1))$. RC_2 is guaranteed to have a cycle time not greater than $\tau(\mathrm{RC}_1)$. This process can be iteratively executed until the retiming solution is found ($\Theta = 1$, $\tau = \tau_{rt}$). For every explored RC, the effective cycle time $\xi(\mathrm{RC})$ can be calculated. At the end of the process, the solution with minimum ξ is returned. Formally, the procedure to find $\xi(\mathrm{RC}_{\min})$ is described by the Algorithm 2. The maximal number of iterations the function `MIN_EFF_CYC` may perform is equal to $(|T| + |E|)^2$. In our experiments this search never did more than ten iterations.

Figure 4.5 illustrates the search for the optimum ξ through a diagram that represents Θ in the x-axis and τ in the y-axis. The `MC` and `MT` labels indicate the progress performed by `MIN_CYC`(Θ) and `MAX_THR`(τ), respectively. The search terminates when the retiming solution is found. In some cases, `MAX_THR`(τ) does not make any progress (e.g., see the loop with label MT in the diagram). In those cases, the ε increase guarantees the termination.

Let us prove that the search does not miss any solution that could provide a better effective cycle time. The proof uses the notion of *non-dominated* RC. We will show that the proposed procedure finds all *non-dominated* RCs and the RC_{min} must be non-dominated. Let us start from the definition of non-dominated RC.

Definition 4.5.1 (Non-dominated RC) *Given an RRG and its two configurations RC_1 and RC_2, we say that RC_1 dominates RC_2 if $\Theta(RC_1) > \Theta(RC_2)$ and $\tau(RC_1) \leq \tau(RC_2)$. An RC is called* non-dominated *if there is no other RC that dominates it.*

Algorithm 2: function `MIN_EFF_CYC(RRG)`

```
    input  : an RRG
    output: $\mathrm{RC}_{min}$ an RC with the minimal $\xi$
 1  $\xi \leftarrow \tau_{rt}$
 2  $\varepsilon \leftarrow \frac{1}{(|T|+|E|)^2}$ /* Lemma 4.5.2 */
 3  $\Theta \leftarrow \frac{\beta_{max}}{\tau_{rt}}$ /* Lemma 4.5.1 */
 4  while $\Theta < 1$ do
 5      RC $\leftarrow$ MIN_EFF_CYC_STEP($\Theta$)
 6      store_nondominated_configuration(RC)
 7      if $\xi(RC) < \xi$ then
 8          $\xi \leftarrow \xi(\mathrm{RC})$
 9      end
10      $\Theta \leftarrow \Theta(R) + \varepsilon$
11  end
12  return $\xi$
```

Each non-dominated RC can be considered as a Pareto-point which explores different cycle time/throughput trade-off[1]. The following property directly follows from this definition:

Property 4.5.1 *If RC_1 dominates RC_2 then $\xi(RC_1) \le \xi(RC_2)$.*

Notice that RC_{min} must be non-dominated. Another non-dominated RC is min-delay retiming configuration. The `MIN_EFF_CYC_STEP` always returns a non-dominated RC. Let us formally prove this.

Theorem 4.5.1 *Let RC_1 be a solution of* `MIN_CYC`(Θ) *and RC_2 the configuration obtained by* `MAX_THR`$(\tau(RC_1))$*. Then, RC_2 is non-dominated.*

Proof: By contradiction. Assume that exists RC' that dominates RC_2. Then, $\tau(\mathrm{RC}') \le \tau(\mathrm{RC}_2) \le \tau(\mathrm{RC}_1)$. The first inequality is due to the definition of non-dominated RC, the second one is due to the fact that `MAX_THR`(τ) provides RCs with cycle time less than or equal to τ. Now, since $\tau(\mathrm{RC}') \le \tau(\mathrm{RC}_1)$ it follows that $\Theta(\mathrm{RC}') \le \Theta(\mathrm{RC}_2)$, because RC_2 has the maximal throughput from the all RCs with cycle time at most $\tau(\mathrm{RC}_1)$. Hence RC' does not dominate RC_2 that contradicts to the original assumption, hence, RC_2 is non-dominated. ■

Therefore, the RCs obtained by calling `MIN_EFF_CYC_STEP` are the candidates to be RC_{min}. They correspond to the black circles in Figure 4.5. The last thing to show is that by summing up ε to the throughput of a non-dominated RC in line 10 of

[1]Strictly speaking Pareto-points are formed by the pairs $(\Theta(\mathrm{RC}), -\tau(\mathrm{RC}))$

Algorithm 2 we do not lose any non-dominated RC. This can be done with the following:

Lemma 4.5.2 *Let Θ_L and Θ_U be two possible values of the throughput for two non-dominated RCs such that $\Theta_L \neq \Theta_U$. Then,*

$$|\Theta_L - \Theta_U| \geq \frac{1}{(|T| + |E|)^2}$$

where $|T|$ represents the number of tokens and $|E|$ is the number of edges in the RRG.

Proof: The throughput of an RC is always a rational number. This directly follows from Definition 4.2.5. Let $\frac{n_1}{m_1}$ and $\frac{n_2}{m_2}$ be two different rational numbers with denominators not greater than m. Then, the following inequality holds:

$$\left|\frac{n_1}{m_1} - \frac{n_2}{m_2}\right| = \left|\frac{n_1 m_2 - n_2 m_1}{m_1 m_2}\right| \geq \frac{1}{m^2}. \tag{4.12}$$

The maximum number of EBs a directed cycle can have is not greater than the number of tokens in this cycle plus the number of edges in this cycle. Since putting more than one bubble in an edge degrades the throughput without improving the cycle time. For this reason, non-dominated RCs can have at most one bubble per edge. Hence, the denominators of the throughput expression of such RCs are less than or equal to $|T| + |E|$. The proof of the lemma immediately follows from this observation and inequality (4.12). ■

A tighter bound could be used by observing that there is always a critical cycle without repeated vertices. Thus, the minimal difference between two different throughputs is lower bounded by the $\frac{1}{(|T|+|N|)^2}$. Finding the longest simple cycle in the graph might still improve this lower bound. However, this problem itself is NP-hard [83].

By combining Lemmas 4.5.2, 4.5.1 and Theorem 4.5.1 we have proved that Algorithm 2 finds all non-dominated RCs.

4.5.5 C-slow retiming

To demonstrate that the proposed model is general, let us show how c-slow retiming can be modeled with the developed framework. See Section 3.1 for the details about

C-slow retiming.

$$\begin{array}{ll} \textit{minimize}: \ \alpha c + \sum\limits_{e \in E} R(e) & \\ \textbf{subject to:} & \\ \quad R(e) = c \cdot T(e) + r(u) - r(v) \geq 0, \quad \forall e = (u, v) & \\ \quad \texttt{Path\_Constr}(\mathrm{RC}, \tau), & \\ \quad c \geq 1, & \\ \quad r(n) \in \mathbb{Z}, \quad \forall n \in N & \\ \quad c \in \mathbb{Z}. & \end{array} \tag{4.13}$$

The initial number of tokens is $T' = cT$. This is because each initialized EB in the original circuit is substituted by c EBs (one token and $c-1$ bubbles). The cost function minimizes c assuming that α is a large constant, thus maximizing the throughput. Therefore, the number of the EBs is minimized.

To conclude, let us show how the minimal effective cycle time can be found using only binary decision variables.

4.5.6 Mixed binary linear programming

The minimal effective cycle time R&R may achieve can be found using binary variables for R, instead of integers. Potentially this decreases the complexity of the model. The reduction is based on the observation that has been already used throughout this work. See, for example, the demonstration of Lemma 4.5.2. Namely, a non-dominated RC can have at most one bubble on each edge. Putting several bubbles on an edge does not break any combinational path and only degrades the throughput of the RRG. Thus, vector R can be represented as follows: $R(e) = T(e) + B(e), B(e) \in \{0, 1\}$. Then, the throughput constraints have the following form:

$$T(e) + B(e) \leq x \cdot T(e) + \sigma(u) - \sigma(v), \quad \forall e = (u, v).$$

The second inequality in cycle time constraints (4.4) is transformed into:

$$t^{out}(e) \geq t^{in}(u) - \tau^*(T'(e) + B(e)), \quad \forall e = (u, v).$$

Finally, the MIP (4.8) to find the minimal effective cycle time has the same form, except that the vector B is constrained to have only binary values. In the experimental results, this modification provided twice average speed up. But for some cases no significant improvement in CPU time was observed.

4.6 Experimental results

A set of experiments has been performed to verify the optimization power of R&R and analyze the performance/area advantages that can be obtained by combining both techniques.

The first consideration for R&R is that it will mostly be applied to coarse levels of granularity, e.g., at the level of 16-, 32- or 64-bit registers in medium and large systems having few dozens or hundreds of computational blocks. Unfortunately, there is no set of benchmarks usable by academia that keep hierarchical information and can be effectively used for realistic experiments. For this reason, we designed synthetic experiments based on the underlying graphs of the sequential ISCAS89 circuits. To obtain an RRG for every circuit, the following transformations were performed:

- The original latches were removed, and every gate was considered to be a large combinational block. Each combinational block was assigned a delay generated randomly from a uniform distribution in the interval $(0, 20]$.
- The largest strongly connected component of the circuit graph was kept in the RRG. The rest of nodes and edges were removed. Even though the proposed techniques do not require an RRG to be strongly connected this transformation reduced the number of edges in the graph, thus reducing the complexity of MILP models to solve.
- Each channel (edge) was assigned a token with a certain probability. The probability was chosen to be 0.5. Thus, about half of the edges were assigned a token, whereas the other half were just wires.

For each example, several solutions were obtained using the optimization techniques presented in Section 4.5.4. In the all test cases retiming always provided configurations with the minimal effective cycle time [2]. The reason for these results is easy to explain. The only potential margin for improvement is the difference $\tau_{rt} - \xi^*$ (see Table 4.1), which was usually small in our examples. Even with the existence of the margin, the degradation in throughput produced by the insertion of bubbles reduces the chances for improvement significantly.

Therefore, to compare simultaneous application of retiming and recycling with their separate application (first retiming and then recycling) a target cycle time was imposed for each RRG. The cycle time was set to $0.75 \cdot \tau_{rt}$, where τ_{rt} is the min-delay retiming cycle time. To meet this constraint bubbles must be inserted into the RRG. After satisfying the performance constraints of the model, the area of the all solutions was minimized.

Table 4.1 reports the results. The "Initial" columns report the parameters of the original RRG: the number of nodes, the number of edges, the initial number of the registers and the cycle time. The fifth column, ξ^*, shows the lower bound on the effective cycle time (see Lemma 4.2.1). The results obtained by min-delay retiming

[2]In paper [11] we failed to obtain exact solutions for some min-delay retiming problems within 5 minutes and the obtained approximations were used instead. Later, with a newer version of MILP solver, the exact solutions were obtained for the all test cases.

Table 4.1: Experimental results.

	Initial					Retiming		Recycling ($\tau \leq 0.75\tau_{rt}$)				Retiming+Recycling ($\tau \leq 0.75\tau_{rt}$)				
	$\lvert N\rvert$	$\lvert E\rvert$	T	τ	ξ^*	T'	τ_{rt}	R	τ	Θ	ξ	R	τ	Θ	ξ	CPU
s27	14	24	14	58.20	58.20	5	58.20	12	30.32	0.50	60.64	12	30.32	0.50	60.64	00:00:01
s208	8	9	2	87.58	87.58	2	87.58	3	51.00	0.50	102.00	3	49.86	0.50	99.72	00:00:01
s344	135	176	76	72.52	39.75	72	41.53	106	26.15	0.50	52.30	101	23.90	0.50	47.80	00:00:16
s349	135	176	76	134.70	46.25	78	48.64	114	27.46	0.50	54.92	95	27.46	0.50	54.92	00:12:00
s382	42	60	23	66.00	38.14	29	41.32	41	24.39	0.50	48.78	34	24.39	0.50	48.78	00:00:02
s386	48	131	64	49.30	43.75	40	46.02	84	25.12	0.50	50.24	76	25.12	0.50	50.24	00:00:04
s400	46	66	29	66.24	50.42	34	52.43	42	33.24	0.50	66.48	39	24.88	0.43	58.05	00:21:26
s420	8	9	2	76.70	76.70	2	76.70	3	40.59	0.50	81.18	3	40.59	0.50	81.18	00:00:01
s444	58	82	41	80.47	52.69	35	55.02	44	31.37	0.50	62.74	38	39.73	0.67	59.60	00:00:09
s510	103	407	197	69.72	64.48	36	64.48	145	34.65	0.50	69.30	132	34.65	0.50	69.30	00:26:27
s526	50	71	31	80.66	80.66	23	80.66	29	45.38	0.50	90.76	30	41.44	0.50	82.88	00:00:01
s641	221	270	133	123.09	40.85	117	43.97	178	25.83	0.50	51.66	137	30.90	0.67	46.35	00:05:19
s713	256	341	158	171.65	40.21	143	47.82	252	23.57	0.42	56.57	240	36.72	0.75	48.96	00:18:25
s820	110	424	205	53.61	53.23	283	53.46	288	31.04	0.50	62.08	122	31.04	0.50	62.08	00:10:09
s832	117	462	226	61.84	49.04	329	50.39	354	30.45	0.50	60.90	231	27.40	0.50	54.80	00:21:42
s838	8	9	2	68.40	68.40	2	68.40	4	38.33	0.50	76.66	3	35.03	0.50	70.06	00:00:01
s953	268	371	172	90.93	50.69	131	53.26	234	25.99	0.40	64.98	158	35.88	0.67	53.82	01:29:40
s1488	133	572	284	72.52	58.05	123	63.39	194	37.05	0.50	74.10	196	36.01	0.50	72.02	00:03:58
s1494	136	572	275	71.10	58.22	306	60.29	393	33.13	0.50	66.26	231	33.13	0.50	66.26	01:48:08

are reported in the next two columns; first goes the number of registers and then the min-delay cycle time.

The configurations for "Recycling" and "Retiming and Recycling" were obtained by imposing the constraint $\tau \leq 0.75\tau_{rt}$. As an example, the target cycle time for `s27` was defined to be $\tau \leq 0.75 \cdot 58.20 = 43.65$. The obtained results had the optimum effective cycle time (ξ) satisfying the cycle time constraint.

The "Recycling" solution was obtained by only inserting bubbles in the RRG and not modifying the location of registers defined by the min-delay retiming[3]. You can observe that the cycle time is often smaller than the target cycle time $0.75\tau_{rt}$. The increase in number of EBs is strictly associated to the bubbles inserted by recycling.

The R&R solutions ("Retiming+Recycling" columns) were obtained by `MIN_EFF_CYC`, as explained in Section 4.5.4. The only modification is that the search was finished when the cycle time of a non-dominated RC became greater than $0.75 \cdot \tau_{rt}$. The results show the benefits in performance and area when combining retiming and recycling in the same model. In some cases, the reduction of the effective cycle time is significant e.g., `s526`, `s641` and `s953`. In other cases, the performance improvement is not so relevant but the reduction in area is important, e.g., `s820`, `s832` and `s1494`. An interesting case is `s400`, in which the throughput of the R&R solution (0.43) is smaller than the one of the recycling solution (0.50). However, the reduction in cycle time (from 33.24 to 24.88) results in a superior performance. This is a clear example of

[3]This was achieved by adding constraint $r = 0$ to the model (4.8)

the diversity of solutions that can be explored by R&R.

The CPU time (hh:mm:ss) is only reported for the R&R solutions, which corresponds to the most complex optimization model. CPLEX [58] was used as MILP solver. To make the computations affordable, a timeout of 600 seconds was defined for each MILP model. It is important to recall that in the iterative approach `MIN_EFF_CYC` generates two MILP models at each iteration of the *while* loop of the algorithm. Even with this limitation of CPU time, the results show that significant improvements in performance can still be obtained, even without guaranteeing optimality. In fact, in the most of the examples the exact solution was obtained. The maximal number of the iterations that was performed in the *while* loop of the function `MIN_EFF_CYC` (see Algorithm 2) is equal to ten.

Results from 2-slow retiming were also obtained, for completeness. All of them had worse performance than the R&R configurations. Given their irrelevance for the main conclusions of this chapter, they are not shown in the table.

4.7 Conclusions and future work

The main conclusions from this chapter is that R&R transformation will hardly improve the performance of ES with only late evaluation nodes with respect to min-delay retiming. Another result is that combination of both techniques allows to find better configurations than those obtained by applying retiming and recycling independently.

We do not know whether a polynomial time algorithm exist for the minimal effective cycle time problem. We believe that this problem is NP-hard, but this fact requires a formal proof.

Later, the same experiments were performed using the MBLP approach of Section 4.5.6. All experiments were completed within two hours and the exact solutions were obtained for all MBLPs. The run time speed up was observed only with the CPLEX solver. With the non-commercial MILP solvers, such as GLPK or LP SOLVE, binary programming hardly reduces the CPU time, sometimes it even significantly increases it.

As a positive thing to conclude, the developed framework can be extended to handle early evaluation nodes. The performance of RRGs with early evaluation nodes can be improved compared to min-delay retiming cycle time at least at this level of abstraction. Chapter 5 analyzes the performance advantages the early evaluation introduces and this analysis is based on the results of this chapter.

Chapter 5

Retiming and Recycling with Early Evaluation

This chapter extends the R&R model from the previous chapter to optimize ESs with early evaluation nodes. To handle early evaluation semantics of a node, a multi guarded marked graph model of an ES is constructed. Then, the throughput constraints in (4.8) can be modified for early evaluation nodes. This allows to use an approach based on non-dominated RCs, described in the previous chapter, to solve (4.8). The derived throughput constraints provide an upper bound for the actual throughput of an ES. Hence, the solution of (4.8) is not necessarily the one with the minimal effective cycle time but, as experimental results show, it is a good approximation.

Let us start with an example that demonstrates early evaluation increases the diversity of non-dominated RCs and consequently, the complexity of the application retiming and recycling (R&R) optimization technique.

5.1 Introductory example

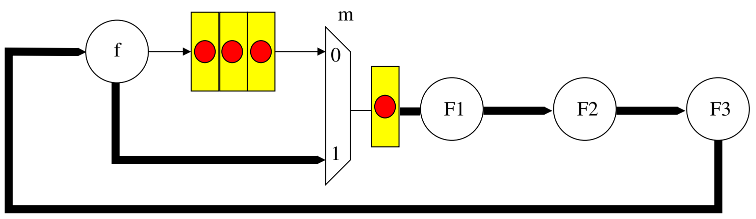
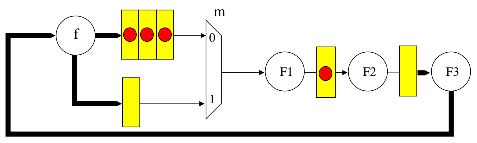
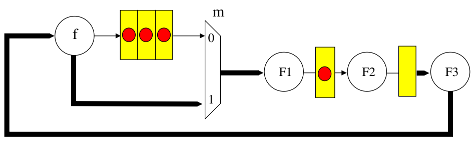


Figure 5.1: A retiming and recycling graph with a multiplexer

Let us show how early evaluation (EE) may help to R&R finding better RCs. Remember that an ES might implement passive or active anti-tokens. An anti-token



(a) First non-dominated RC



(b) Second non-dominated RC

Figure 5.2: Non-dominated RCs of the RRG from Figure 5.1.

is passive if it is waiting for a positive token to come and cancel it. The active anti-token may travel backward through the control logic, looking for a corresponding positive token. Let us, for the moment, focus only on the passive anti-tokens.

Consider the RRG depicted in Figure 5.1 with combinational nodes $F1, F2, F3, f$ and m. The node m is drawn differently to represent a multiplexer. The select signal of m is omitted to simplify the picture. Here we assume that the select signal is not critical. Assume that the delay of the nodes F_1, F_2, F_3 is equal to one unit of time and the delay of nodes f and m is equal to zero. Then, the cycle time of the RRG is equal to three units of time. The critical combinational path, $F_1 \to F_2 \to F_3 \to f \to m$, is marked with a bold line in the figure. The directed cycle $(F_1 \to F_2 \to F_3 \to f \to m \to F_1)$, has only one EB. Since retiming preserves the number of EBs at each directed cycle (see Section 3.1), three is the min-delay retiming cycle time and the depicted configuration is a min-delay retiming configuration.

For the moment let us ignore early evaluation and assume all nodes of the RRG depicted in the figure be late evaluation. R&R finds two more non-dominated RCs for this RRG. Remember that the min-delay retiming configuration is also non-dominated.[1] Figure 5.2(a) shows the non-dominated RC with the smallest throughput and cycle time. The cycle time is equal to one, the throughput is equal to 1/3. The effective cycle time is equal to three units of time, which is equal to the min-delay retiming cycle time. Figure 5.2(b) shows the second non-dominated RC. The cycle time is equal to two units of time, the throughput is equal to 1/2. The effective cycle time of this RC is equal to four units of time, which is worth than the min-delay retiming cycle time. Finally, we can conclude that three is the minimal effective cycle time the R&R achieves for this RRG.

Now let us assume that the select signal of the node m always chooses its zero branch with probability α, and the one branch with probability $(1-\alpha)$. The throughput of the first non-dominated RC in Figure 5.2(a) can be found by resolving the corresponding *Markov chain*, similarly, as it is done in Section 2.5.3. The interested reader may browse Section 5.2 for the details of using Markov chains for performance analysis. Here we just use the final results. The throughput of the first non-dominated RC is equal to 0.491 for $\alpha = 0.5$. Hence, the effective cycle time is

[1]See Section 4.5.4 for the definition of non-dominated configuration.

equal to $1/0.491 \approx 2.037$ units of time. For $\alpha = 0.9$ the throughput is higher and it is equal to 0.719 and the effective cycle time is approximately equal to 1.39 units of time, which is more than twice less than the effective cycle time of the min-delay retiming configuration. Similarly, the second non-dominated RC can be analyzed. The interested reader is welcomed to verify that its throughput is approximately equal to 0.774 when α is equal to 0.9.

It may appear that EE does not change the set of non-dominated RCs. If it were like this, then, to use EE in R&R we may first execute R&R treating all nodes as late evaluation. After this, we recalculate the throughput of each non-dominated RC using the EE information and select the one with the minimal effective cycle time. The RC depicted in the Figure 5.3 shows that this is not always the case. This RC has anti-tokens which are drawn as a rhombus with "-2" inside. The late

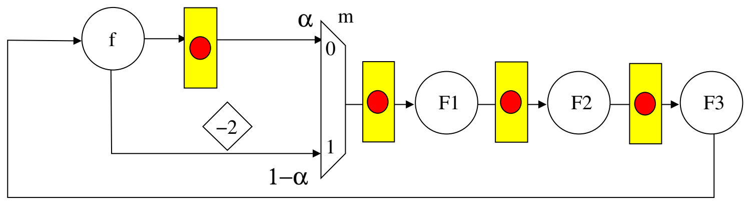


Figure 5.3: Configuration with minimal effective cycle time

evaluation throughput of this configuration is still equal to 1/3. But with m being EE the throughput is higher. It is even higher than the throughput of the RC with the EE mux in Figure 5.2(a). To see this, it is enough to note that the non-critical cycle has a ratio tokens/delay equal to 4/5 in the first case and 1 in the second case. The analysis of the Markov chain of this RC gives the following expression for the throughput: $1/(3 - 2\alpha)$. For $\alpha = 0.9$, the throughput is equal to $\frac{5}{6} \approx 0.833$ that is 16% better than the throughput of the RC in Figure 5.2(a) (0.719). Notice, that the Markov chain in Figure 5.4 has no states with two anti-tokens. This example shows that the trivial version of R&R with early evaluation does not always find the best possible solution.

5.2 Calculating throughput using Markov chains

This section shows how the throughput of the RC depicted in Figure 5.2(a) can be calculated exactly with Markov chain analysis[2]. This example illustrates the complexity of having an exact estimation of the throughput.

Figure 5.4 shows the Markov chain that corresponds to the behavior of the non-dominated RC depicted in Figure 5.2(a). Remember, that the probabilities for zero

[2]Assuming memoryless of branch selection. See also Section 3.2.3

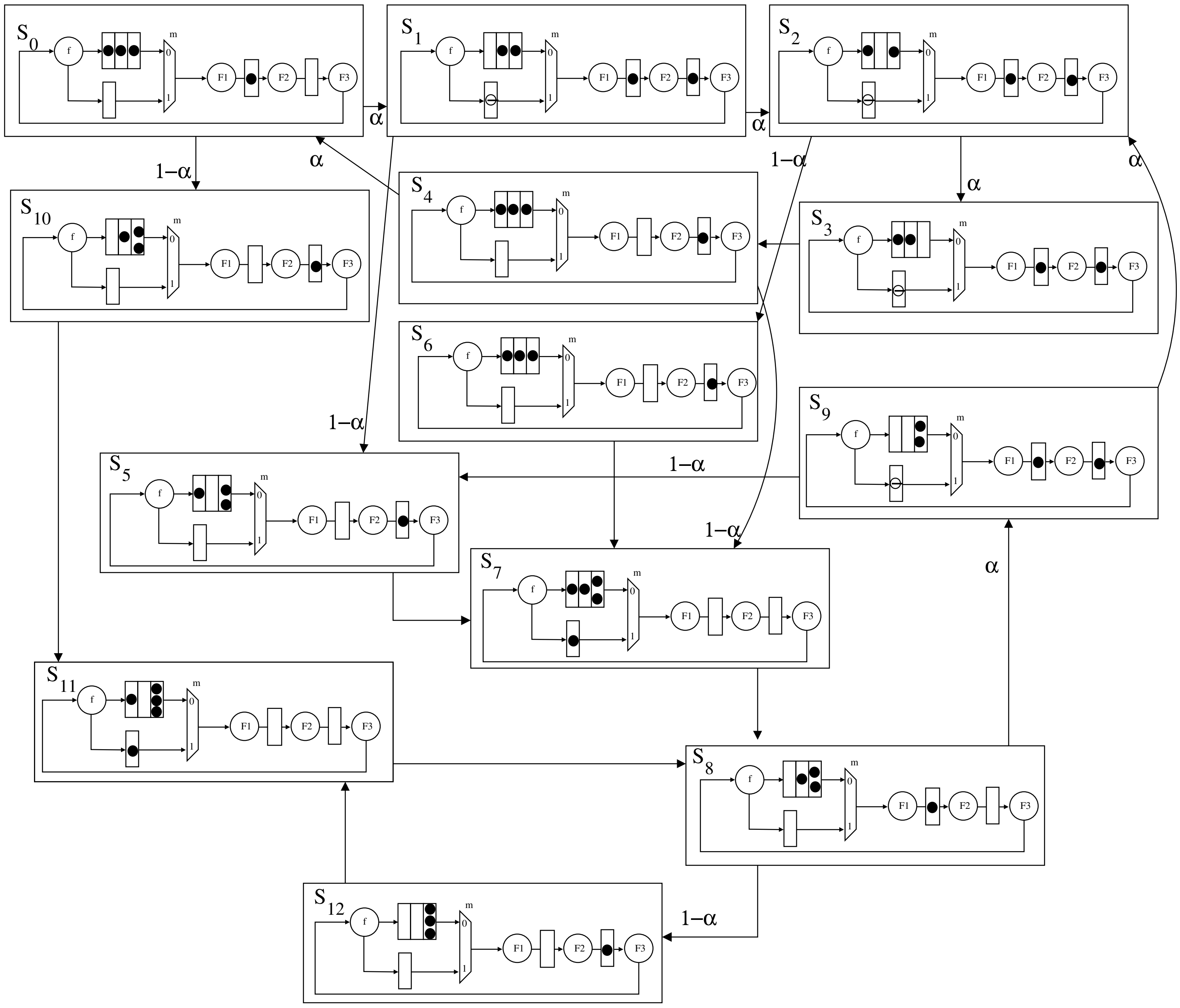


Figure 5.4: Markov chain for the RRG from Figure 5.2(a).

and one branches of the EE mux are equal to α and $1-\alpha$ respectively. Initially, the RC is in the state S_0 and no branch of the mux is selected. The duration of each state is supposed to be one clock cycle. If zero branch of m is selected, nodes F_2 and m produce tokens and RC goes from state S_0 to state S_1. Otherwise, the one branch is selected and only node F_2 produces a token. This corresponds to the arc $S_0 \rightarrow S_{10}$. Observe, that in the state S_{10} the last EB of the zero branch (f, m) stores two tokens. The second token is stored in the buffer of the control logic of the synchronous elastic design. In the state S_{11} this EB stores three tokens. Remember, that this chapter assumes that all EBs have infinite capacitances and hence, they never assert stop bit.

The process of the construction of the Markov chain for RC is similar to the one, described in Section 2.5.4. The arcs of the state transitions are marked with the probabilities of the corresponding transitions. If an arc is unmarked, as for example $S_5 \rightarrow S_7$, then its probability is equal to one. The unmarked arcs always go out from the states where all branches of the EE nodes are selected. State S_{10} is another example for such cases, since in S_{10} the one branch of the mux is already selected.

The states S_4 and S_6 have the same markings, the only difference between them is that in the state S_6 the one branch of the multiplexer has been selected, while in the state S_4 the select signal is unknown and both values are possible. This is why the state S_4 has two output edges, while S_6 has only one output edge. Some states contain an anti-token, for example, state S_1. The anti-token is represented as a cycle with the sign "minus" inside. Anti-tokens are produced when the zero branch of the mux is selected and there is no valid data in another input EB of the mux, as in the transition $S_0 \rightarrow S_1$. When an anti-token meets the token they are mutually canceled, as in the transition $S_3 \rightarrow S_4$. State S_3 has an anti-token which is canceled in the next state by the token that has been produced by node F_3.

If α is equal to one, then the only reachable states are represented by the cycle $(S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_0)$. The reader may verify that all computational nodes produce exactly four tokens during this cycle. Given that the duration of each state is equal to one clock cycle it can be concluded that the throughput of the RC is equal to 4/5 when $\alpha = 1$. Notice, that this value is equal to the tokens to EB ratio of the top most cycle of the RC. Another *corner* behavior corresponds to the case when α is equal to zero. In this case, the behavior of the RC is limited to the directed cycle $(S_{11} \rightarrow S_8 \rightarrow S_{12} \rightarrow S_{11})$. Notice, that for this case the initial state S_0 is not a part of steady-state behavior. All steady states must be at a directed cycle of the Markov chain. The throughput of this behavior is equal to 1/3. The cycle has three states and all the nodes produce exactly one token during this cycle.

Resolving Markov chain the following expression for the throughput is obtained:

$$\frac{1+\alpha+\alpha^2+\alpha^3}{3+\alpha+\alpha^2+\alpha^3-\alpha^4}.$$

To obtain this value it should be realized that the throughput of an ES is equal to the probability of any EB having a token. In the example, the EB between nodes F_1 and F_2 has a token in the states S_0, S_1, S_2, S_3, S_8 and S_9. Thus, the throughput of the RC is equal to the sum of the probabilities of all these states. Now let us start from the state S_2, the probability of the state S_3 is equal to αS_2 since the only possible way to go to S_3 is from the state S_2. Similarly,

$$S_4 = S_3 = \alpha S_2, \quad S_0 = \alpha S_4 = \alpha^2 S_2, \quad S_1 = \alpha^3 S_2,$$

etc. Doing in this way, the probabilities of all the states can be expressed in terms of the probability of the state S_2. After this, the probability S_2 can be found using the condition that the sum of the all probabilities is equal to one. Finally, summing up the probabilities of the states where the EB on the edge (F_1, F_2) has a token the expression for the throughput is obtained.

The Markov chain analysis is a general way to calculate the throughput of any RC with early evaluation nodes. But the number of states of a Markov chain increases exponentially with increasing the number of edges and the number of initial tokens in an RC. Now it will be shown how the throughput of RCs with EE nodes can be estimated in a more efficient way. This estimation is based on multi-guarded marked graph model of an ES. See Section 2.5.3 for more details about multi-guarded marked graphs.

5.3 RRG with early evaluation nodes

First of all, let us formally introduce early evaluation into the RRG. From this moment all RRGs are allowed to have EE nodes and we can reuse the abbreviation RRG for referring to RRGs with EE nodes.

Definition 5.3.1 (RRG with EE nodes) *A* Retiming and Recycling Graph with EE nodes *(RRG) is a tuple* (S, β, T, R, γ)*, where:*

- $S = (N, E)$ *is the underlying multi-graph of the ES,* N *is the set of nodes and* E *is the set of edges. The set* N *is partitioned into* N_L *and* N_E*:* N_L *includes the nodes with late evaluation and* N_E *the early evaluation nodes.*
- $\beta : N \to \mathbb{R}^+$ *assigns a combinational delay to each node.*
- $T : E \to \mathbb{Z}$ *is the number of tokens in each EB at the initial state. If negative, the absolute value of* $T, (|T|)$*, provides the number of anti-tokens. To ensure absence of dead-locks the sum of tokens on each directed cycle of* S *must be positive.*
- $R : E \to \mathbb{Z}^+$ *is the number of EBs on each edge. The condition* $R \geq T$ *must hold.*

- *$\gamma : E \to \mathbb{R}^+$ is the branch selection probability of the input edges of early evaluation nodes $n \in N_E$. The sum of the probabilities for all inputs of an early evaluation node $n \in N_E$ is equal to one:*

$$\sum_{e=(n_i,n)\in E} \gamma(e) = 1.$$

EE nodes produce tokens according to the following semantics. An EE node produces a new token if the selected branch has a token, otherwise the EE node waits until a token in the selected branch appears. Another branch can not be selected until the previous computation has been completed. In other words, EE nodes have finite-server semantics. If an EE node produces a token and some of its input edges have no tokens, then an anti-token is generated for each of these edge.

Figures 5.2 and 5.3 provide examples of RRGs, where the set N_E contains only one element, the multiplexer m. The definitions of the cycle time and throughput are inherited from Section 4.2. The cycle time constraints `Path_Constr`(RC, τ) (see Lemma 4.4.1) are unchanged. Similarly, the analog of the Lemma 4.2.1 for RRGs with EE nodes can be proved.

Lemma 5.3.1 (Lower bound for the effective cycle time) *For each RC the following inequality holds:*

$$\xi(RC) \geq \min_{c\in C} \frac{\sum\limits_{v\in c} \beta(v)}{\sum\limits_{e\in c} T(e)},$$

where C is the set of all directed cycles of the RRG.

Proof: Similarly to Lemma 4.2.1, the following inequality holds:

$$\tau(\mathrm{RC}) \geq \frac{\sum\limits_{n\in c} \beta(n)}{\sum\limits_{e\in c} R(e)},$$

for each $c \in C$.

Let $\Theta_{le}(c) = \frac{\sum\limits_{e\in c} T(e)}{\sum\limits_{e\in c} R(e)}$ be the late evaluation throughput of the directed cycle c. Then the following chain of inequalities holds:

$$\frac{\tau(\mathrm{RC})}{\Theta_{le}(c)} \geq \frac{\sum\limits_{n\in c} \beta(n)}{\sum\limits_{e\in c} R(e)} \cdot \frac{\sum\limits_{e\in c} R(e)}{\sum\limits_{e\in c} T(e)} = \frac{\sum\limits_{n\in c} \beta(n)}{\sum\limits_{e\in c} T(e)}.$$

This inequality can be rewritten as follows: $\Theta_{le}(c) \leq \tau(\mathrm{RC}) \cdot \frac{\sum_{e \in c} T(e)}{\sum_{n \in c} \beta(n)}$. The actual throughput of the RC is less than or equal to the maximal $\Theta_{le}(c)$ taken for all directed cycles of the RRG. Hence, we have the following chain of the inequalities:

$$\Theta(\mathrm{RC}) \leq \max_{c \in C} \Theta_{le}(c) \leq \tau(\mathrm{RC}) \cdot \max_{c \in C} \frac{\sum\limits_{e \in c} T(e)}{\sum\limits_{n \in c} \beta(n)}.$$

And finally for the effective cycle we have:

$$\xi(\mathrm{RC}) = \frac{\tau(\mathrm{RC})}{\Theta(\mathrm{RC})} \geq \frac{\tau(\mathrm{RC})}{\tau(\mathrm{RC}) \cdot \max \frac{\sum_{e \in c} T(e)}{\sum_{n \in c} \beta(n)}} = \min_{c \in C} \frac{\sum\limits_{v \in c} \beta(v)}{\sum\limits_{e \in c} T(e)}.$$

■

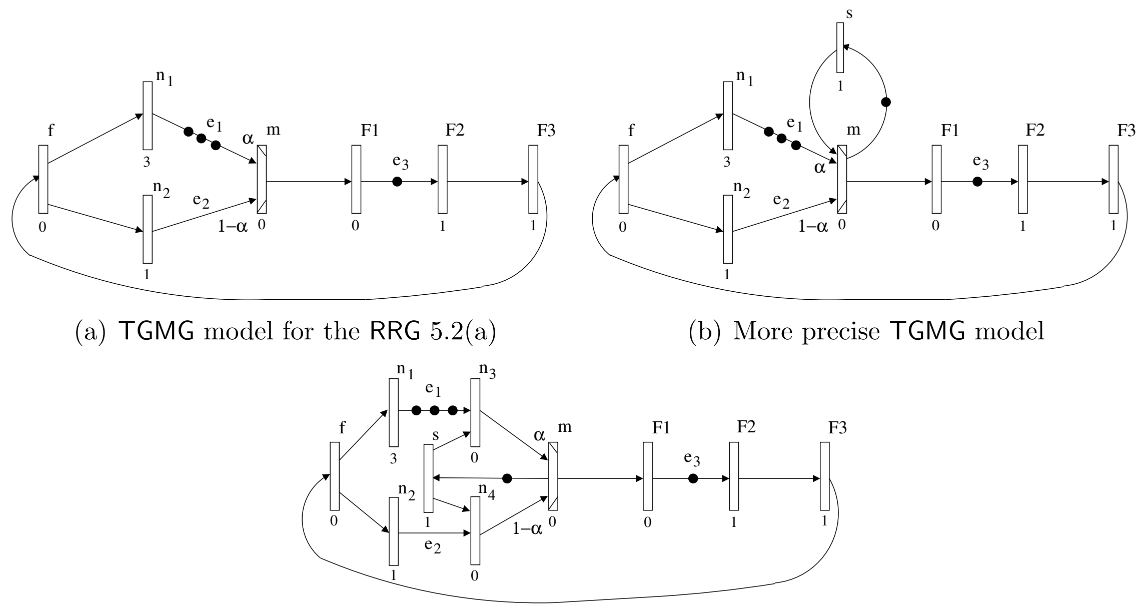
Comparing with the result of Lemma 4.2.1, this lower bound leaves more freedom for optimization. Still, it can be observed that for RRGs with well-balanced delays the chances to improve min-delay retiming cycle time are reduced, because for such RRGs the upper bound of Lemma 5.3.1 is close to the upper bound of Lemma 4.2.1, which, in turn, is close to the min-delay retiming cycle time.

5.4 Throughput constraints

To modify the throughput constraints (4.7) in (4.8) we construct a multi-guarded timed marked graph (TGMG) which models the behavior of an RC. The idea is to associate the number of EBs ($R(e)$) with the delay of transition and use initialized EBs ($T(e)$) as the number of tokens in corresponding place.

Figure 5.5(a) shows the TGMG model of the RC from Figure 5.2(a). Each transition represents a computational node of the RC. Node F_1 has only one input edge without EBs on it, thus the delay of the corresponding transition is equal to zero and the marking of the input edge also is equal to zero. Node m has two input edges (a join node), hence, the number of EBs on all input edges can not be associated with the transition m. To handle this, two new transitions are added for each input edge (n_1 and n_2) and their delays are set according to the number of EBs. Notice, that for each directed cycle of the RC the total number of EBs is equal to the total delay of the corresponding cycle of the TGMG.

The Algorithm 3 formalizes the construction of a TGMG model for an RC. The first line sets the graph structure of the TGMG to be the same as the one of the corresponding RRG. A node of the TGMG has early evaluation if its corresponding node in the RRG has. The branch selection probability function (γ) of the TGMG is



(a) TGMG model for the RRG 5.2(a)

(b) More precise TGMG model

(c) Singleton TGMG model

Figure 5.5: Multi-guarded marked graph models for RRGs with EE nodes.

Algorithm 3: Construction of TGMG model for an RRG

```
   input  : An RRG
   output: An TGMG model of the RRG
1  TGMG ← RRG
2  foreach $n \in N$ do
3      if $in\_degree(n) = 1$ then
4          $e \leftarrow (n_o, n)$
5          $m_0(e) = T(e)$
6          $\delta(n) = R(e)$
7      else
8          foreach $e = (n_i, n) \in E$ do
9              split $e$ into $(n_i, n_j)$ and $(n_j, n)$
10             $m_0(n_i, n_j) = 0$
11             $m_0(n_j, n) = T(e)$
12             $\delta(n_j) = R(e)$
13             $\delta(n) = 0$
14         end
15     end
16 end
```

equal to the corresponding function of the RRG, (see Figure 5.5(a)). The function *in_degree*(n) provides the input degree of node n. Lines $3-6$ applied to the node F_2 of the RRG from Figure 5.2(a) put one token on the edge e_3 and set the delay of node F_2 equal to one. Lines $8-14$ are applied to the node m.

If an RC has only simple nodes, its throughput is equal to the throughput of the TGMG obtained by Algorithm 3. However, if an RC has early evaluation nodes, then the throughput of the TGMG may be greater than the throughput of the RC. For example, if α is equal to 0.5 the throughput of the TGMG in Figure 5.5(a) is equal to 0.625 which is equal to the optimal solution of LP (2.7). From Section 5.2 we know that the throughput of the RC depicted in Figure 5.2(a) is equal to 0.491. The reason for this difference is that the EE transition of the TGMG can produce several tokens in one unit of a time, changing the order of the tokens. Even though such type of *out-of-order completion* could be implemented it is out of the scope of this work. This work assumes that all computational nodes of an ES preserve the flow of the data.

To forbid the EE nodes to change the order of the tokens, a special loop is introduced for each such a node. Figure 5.5(b) shows the result TGMG for the TGMG depicted in Figure 5.5(a). It can be observed, that the result TGMG is not singleton (see Section 2.5.3 for the definition of a singleton TGMG). The singleton TGMG with the same throughput is obtained by applying the transformation described in section 2.5.3 (Figure 2.12). For our example, Figure 5.5(c) shows the final TGMG model. The throughput of this TGMG is equal to 0.491 (for $\alpha = 0.5$) which exactly matches our expectations.

The final TGMG model for an RC is obtained first by applying Algorithm 3 and then by introducing a self-loop for each EE node as it shows Figure 5.5(b). Algorithm 4 formalizes the construction process.
The firing semantics of the EE transitions in the TGMG model coincides with the firing semantics of the EE nodes of an RC. Thus, the throughput of the RC is equal to the throughput of its TGMG model that is constructed by Algorithm 4. Then, using the TGMG model and LP (2.7) the following lemma can be derived.

Lemma 5.4.1 (Throughput constraints) *If the throughput of an RC is less than or equal to Θ, then the following system of inequalities is feasible:*

$$R(e) \le x \cdot T(e) + \sigma(v) - \sigma(u) \quad \forall e = (u,v),\ \textit{such that } v \in N_L, \tag{5.1}$$

$$R(e) \le \sigma(u) - aux_R(e) \quad \forall e = (u,v),\ \textit{such that } v \in N_E, \tag{5.2}$$

$$0 \le \sum_{e \in {}^\bullet v} \gamma(e) \cdot (aux_0(e) - \sigma(v)) \quad \forall e = (u,v),\ \textit{such that } v \in N_E, \tag{5.3}$$

$$x + \sigma(v) - \sigma(s) \ge 1 \quad \forall v \in N_E, \tag{5.4}$$

$$\sigma(s) - aux_0(e) \ge 0 \quad \forall e = (u,v),\ \textit{such that } v \in N_E, \tag{5.5}$$

$$x \cdot T(e) + aux_R(e) - aux_0(e) \ge 0 \quad \forall e = (u,v),\ \textit{such that } v \in N_E. \tag{5.6}$$

Algorithm 4: Refining a TGMG model

```
input  : An TGMG model produced by Algorithm 3
output: An TGMG model of the RRG
foreach $n \in N_E$ do
    add a new late evaluation node $s$
    $\delta(s) = 1$
    add an edge $(n, s)$
    $m_0(n, s) = 1$
    foreach $e = (n_i, n) \in E$ do
        split $e$ into $(n_i, n_k)$ and $(n_k, n)$
        add a new edge $(s, n_k)$
        $\delta(n_k) = 0$
        $m_0(n_i, n_k) = m_0(e)$
        $m_0(n_k, n) = 0$
        $m_0(s, n_k) = 0$
    end
end
```

Where $x = 1/\Theta$ is used instead of Θ to avoid quadratic terms in (5.1)*, similarly as it is done in Section 4.4.2. Inequalities* (5.2)-(5.6) *are applied for each* EE *node v. Variables aux_R are the firing count for the nodes introduced by the Algorithm 3. Variables aux_0 and s refer to the nodes introduced by Algorithm 4.*

The demonstration of this lemma is a direct application of LP (2.7) to the TGMG model of an RC.

From this moment the $\texttt{Throughput\_Constr}(\text{RC}, \Theta)$ refers to the constraints (5.1)-(5.6). Using Lemma 5.4.1, an upper bound of the throughput for a given RC, can be computed with the following LP:

$$\begin{aligned} &\textit{minimize} \quad x : \\ &\quad \texttt{Throughput\_Constr}(\text{RC}, \Theta). \end{aligned} \tag{5.7}$$

By modifying throughput constraints in (4.8) we obtain a mathematical programming model to find an RC with the minimal effective cycle time. This problem can be solved in a similar way as in Chapter 4. Since, the LP (5.7) provides only an upper bound on the actual throughput of the RC, the exact solution of (4.8) does not necessary provides an RC with the minimal effective cycle time. To evaluate the quality of the provided approximation a lot of experiments were performed and the results were quite satisfactory.

Another observation is that Lemma 4.5.2 is not valid for RRGs with EE nodes. The demonstration of a similar lemma for the EE case is left as part of future work. For the moment a small enough value can be used to separate non-dominated configurations. For example, in the experiments that were performed for this chapter, epsilon in Algorithm 2 was set to 0.01.

What about the mixed binary linear program (MBLP) that was described in Section 4.5.6? The EE version of R&R can not be solved with binary programming, at least in the given framework. This is because the way the cycle path constraints are derived.

Another problem related to the EE nodes naturally arises. Given an ES with a set of EE nodes, which nodes can be converted to late evaluation while preserving the minimal effective cycle time R&R achieves? The paper [60] provides a heuristics to solve this problem. The *exact* version of this heuristics can be derived by using OR-relation in mixed linear programming (see Equation (2.3)). The two groups of constraints for each EE transition are introduced: the early evaluation and the late evaluation. Then, the number of the active early evaluation constraints is minimized, while preserving original throughput upper bound.

5.5 Active anti-tokens

The question to answer in this section is whether the active anti-tokens mechanism may further increase the performance of an ES. To answer this question let us consider the RC depicted in Figure 5.6. It has three EBs and two EE multiplexers, m_0 and m_1. The skid-buffer of each multiplexer is drawn as a rhombus with zero inside. Remember, that skid-buffers are special buffers of the control logic that can store tokens but do not increase the forward latency of the ES [37].

First, let us assume that the anti-tokens are passive. Figure 5.7(a) shows the Markov chain of the RC behavior. If the $(1-\alpha)$ branch of the mux m_0 is selected, then the initial state S_0 is preserved, which means that all nodes have computed new values, thus, the throughput of the RC is equal to one in this case. If both "slow" branches of the muxes are selected simultaneously then the initial state is changed to the state S_2; the probability of this transition is equal to $\alpha \cdot \beta$. The transition $S_0 \rightarrow S_1$ occurs when α and $(1-\beta)$ branches are selected. In that case, the multiplexer m_1 produces new token and generate an anti-token in the skid-buffer. The anti-token is waiting for a token from the mux m_0 which comes in the next time stamp. State S_1 is preserved if the $(1-\beta)$ branch of the m_1 is selected as this results that all nodes have produced token, otherwise, with probability equal to β the state changes to S_2. Resolving the Markov chain the following expression for the throughput of the RC is obtained

$$1 - \frac{\alpha\beta}{\alpha+\beta}.$$

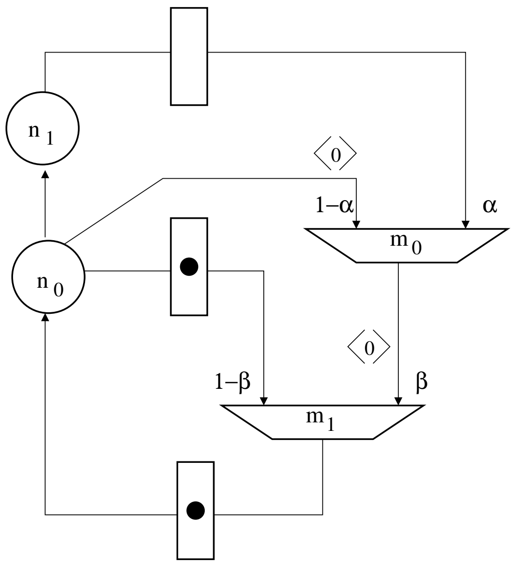


Figure 5.6: An RRG where active anti-tokens leads to a better throughput.

The Markov chain of the same RC with active anti-tokens mechanism is depicted in Figure 5.7(b). To derive this chain it is enough to notice that the state S_1 no longer exists. Since the anti-token in the skid buffer of the mux m_1 travels backward and cancels input tokens of the mux m_0. Hence, state S_1 can be merged with the state S_0. The probability to preserve state S_0 is higher now and it is equal to the sum of the probabilities of three possible branch selections:

$$(1-\alpha)\beta + (1-\alpha)(1-\beta) + (1-\beta)\alpha = 1 - \alpha\beta.$$

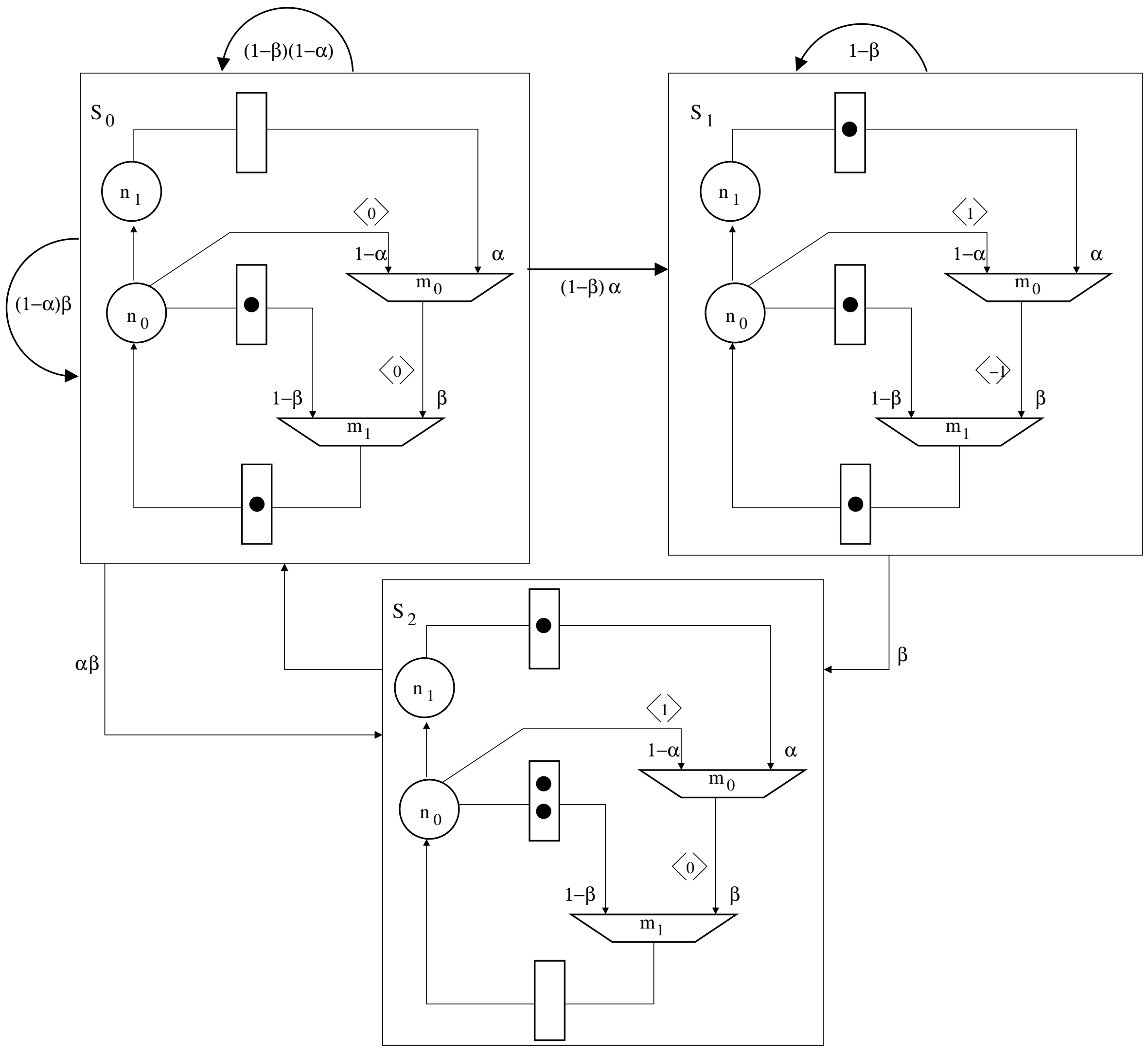
Resolving this Markov chain the following expression for the throughput is obtained:

$$1 - \frac{\alpha\beta}{1+\alpha\beta}.$$

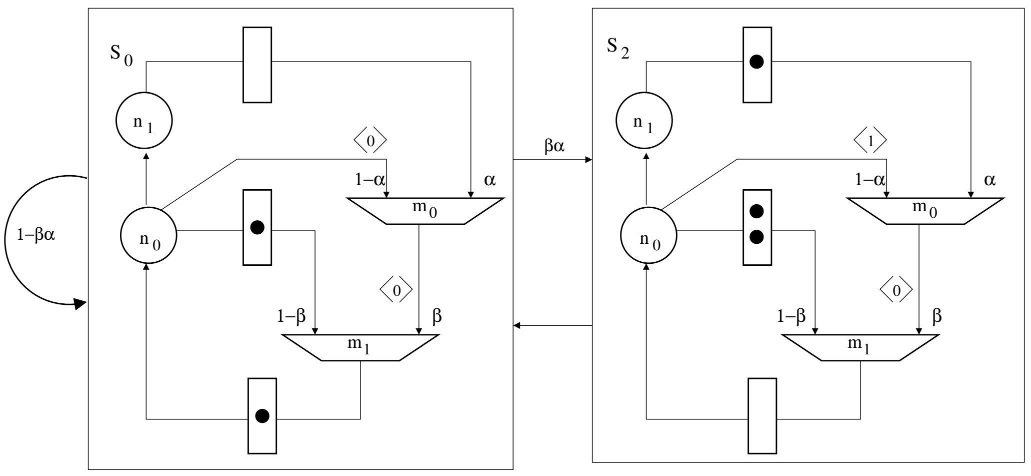
It can be proved, that for this example the throughput with active anti-tokens is always greater than the throughput with passive anti-tokens, independently of the values of α and β. Indeed:

$$[1 - \frac{\alpha\beta}{1+\alpha\beta}] - [1 - \frac{\alpha\beta}{\alpha+\beta}] = \frac{\alpha\beta(1-\alpha)(1-\beta)}{(1+\alpha\beta)(\alpha+\beta)} > 0,$$

when $0 < \alpha < 1$ and $0 < \beta < 1$. For example, if $\alpha = 0.25$ and $\beta = 0.2$, the throughput of the RC with passive anti-tokens mechanism is equal to $8/9 \approx 0.888$ while with active anti-tokens it is equal to $20/21 \approx 0.952$. Notice, that this throughput can or cannot be achieved in a particular implementation of elastic protocol. See, for



(a) Passive anti-tokens. Throughput: $1 - \frac{\alpha\beta}{\alpha+\beta}$



(b) Active anti-tokens. Throughput: $1 - \frac{\alpha\beta}{1+\alpha\beta}$

Figure 5.7: Markov chains for RC depicted in Figure 5.6 with passive and active anti-tokens

example, [26] for a discussion about details of different implementations. Our results of Verilog simulation for this example coincide with the theoretical prediction. MArEx tool [61] was used to generate control module.

It turns out that the TGMG model that was proposed in the previous section is correct only for passive anti-tokens. To get the TGMG which correctly handles active anti-tokens mechanism, in this example it is enough to remove self-loop from the mux m_0. To find out a general model to estimate the throughput of ESs with EE nodes and active anti-tokens more investigation is required. Remember, that in this chapter it is assumed that back-pressure never limits the throughput of an ES. For examples, when this is not the case the reader is referred to [22, 26].

5.6 Experimental results

A set of experiments was performed to verify the throughput model (5.1)-(5.6) and to demonstrate the optimization power of the R&R for ESs with EE nodes. A set of RRGs was generated as follows:

- The ISCAS89 circuits have been used to extract the underlying graph structures. The largest strongly connected component of the each ISCAS89 circuit was taken. The rest of nodes and edges were removed. Actually, they are the same graphs from the experiments of the previous chapter.
- Each edge was assigned a token with probability 0.25. Thus, about 25% of the edges were assigned an initialized EB, whereas the rest were just wires.
- Each node was assigned a combinational delay uniformly distributed in the interval $(0, 20]$.
- A node with more than one input was marked as early evaluation with probability of 0.4. The branch probabilities were selected randomly.

Table 5.1: All non-dominated RCs for the test case s526

Name	τ	Θ^{lp}	Θ	err(%)	ξ^{lp}	ξ	Δ(%)
s526	19.98	0.2500	0.2390	4.6025	79.9200	83.5983	
	24.10	0.3333	0.3050	9.2896	72.3000	79.0164	
	31.74	0.4936	0.4200	17.5219	**64.3041**	75.5714	
	56.54	0.8367	0.7910	5.7787	67.5742	**71.4791**	
	74.52	1.0000	1.0000	0.0000	74.5200	74.5200	5.4

Let x_0 be a solution of (5.7) for a given RC, then let us denote $\Theta^{lp}(\mathrm{RC}) = 1/x_0$. The corresponding effective cycle time will be denoted as $\xi^{lp}(\mathrm{RC})$, i.e., $\xi^{lp} = \tau(\mathrm{RC})/\Theta^{lp}(\mathrm{RC})$.

For each test case the RC_{min}^{lp} was found by calling the `MIN_EFF_CYC` procedure, described in Section 4.5. The Verilog representation of synchronous elastic controller was generated for each non-dominated RC. Then, the actual throughput was calculated by performing a Verilog simulation. MArEx [61] tool was used to generate Verilog code based on a high level circuit description.

As an example, Table 5.1 shows all the RCs that were found for the test case s526. Rows of the table correspond to different RCs. The column τ provides the cycle time of the RC. The columns Θ^{lp} and Θ provide the throughput upper bound and the actual throughput of the RC (obtained by simulation) respectively. The column $err(\%)$ provides the relative difference between the throughput upper bound Θ^{lp} and Θ. The effective cycle times of RC_{min}^{lp} and RC_{min} are marked in bold in the columns ξ^{lp} and ξ respectively. The last column $\Delta(\%)$ is the relative difference between $\xi(\mathrm{RC}_{min})$ and $\xi(\mathrm{RC}_{min}^{lp})$, e.g., for s526 it is equal to $(75.5714 - 71.4791)/71.4791 \cdot 100\% \approx 5.4\%$. It can be seen that the RC_{min}^{lp} and RC_{min} are different configurations in this case, however RC_{min}^{lp} has only 5.4% worse performance. Also the second best configuration returned by Algorithm 2 does correspond to RC_{min}.

Table 5.2 shows the obtained results. The first column is the name of the underlying ISCAS89 circuit. The next three columns are the number of simple nodes, early evaluated nodes and edges respectively. The column ξ^* provides the cycle time of test case before the optimization (it is equal to the effective cycle time because originally RRGs have no bubbles). The column ξ_{nee} provides the minimal effective cycle time of the RRG with all nodes being late evaluation. As in the experiments in the previous chapter, it always coincides with the min-delay retiming cycle time. The columns ξ_{min}^{lp} and ξ_{min}^{sim} show $\xi(\mathrm{RC}_{min}^{lp})$ and $\xi(\mathrm{RC}_{min})$, respectively. For example, for the RRG s526 the corresponding values are equal to 75.57 and 71.48 units of time. The column $\Delta(\%)$ shows the relative difference between ξ_{min}^{lp} and ξ_{min}^{sim}. The last column $I(\%)$ provides the performance improvement obtained by `MIN_EFF_CYC` using early evaluation. It is calculated as follows:

$$I = \frac{\xi_{nee} - \xi_{min}^{sim}}{\xi_{nee}} \cdot 100\%.$$

CPLEX [58] was used as an MILP solver. The timeout for integer optimization was set to 20 minutes in all experiments, but usually the optimal solution could be found before this timeout was reached.

Observation 1: The average effective cycle time improvement is equal to 14.5%. It has been calculated as the mean value of the column $I\%$. The improvement in the particular case strongly depends on the position of early evaluation nodes. In the RRG s953 the effective cycle time has been improved more than twice with respect to the min-delay retiming cycle time. Theoretically, even larger performance improvements can be achieved with EE (see Lemma 5.3.1). ξ_{nee} was not improved for s832, s1488,

Table 5.2: Experimental results.

Name	$\|N_L\|$	$\|N_E\|$	$\|E\|$	ξ^*	ξ_{nee}	ξ^{lp}_{min}	ξ^{sim}_{min}	$\Delta(\%)$	$I\%$
s208	7	1	9	87.58	87.58	85.54	85.54	0.00	2.3
s641	206	15	270	183.15	109.62	93.72	89.98	3.99	17.9
s27	9	5	24	43.73	43.73	32.31	32.31	0.00	26.1
s444	45	13	82	174.88	106.75	92.5	92.5	0.00	13.3
s838	7	1	9	68.4	68.4	59.99	59.99	0.00	12.3
s386	36	12	131	74.8	74.6	58.55	59.81	-2.15	21.5
s344	122	13	176	130.63	114.19	90.79	82.89	8.70	27.4
s400	37	9	66	149.29	79.5	80.1	77.63	3.08	2.3
s526	43	7	71	144.47	74.52	75.57	71.48	5.41	4.1
s382	35	7	60	84.65	68.47	66.07	66.07	0.00	3.5
s420	7	1	9	76.7	76.7	59.78	59.78	0.00	22.1
s832	76	41	462	62.11	50.39	50.39	50.39	0.00	0
s1488	85	48	572	64.28	59.52	59.52	59.52	0.00	0
s510	63	40	407	116.63	116.63	73.26	73.26	0.00	37.2
s953	232	36	371	354.86	292.28	125.92	119.53	5.07	59.1
s713	229	27	341	119.15	96.63	99.13	95.96	3.20	0.7
s1494	88	48	572	61.97	55.8	55.8	55.8	0.00	0
s820	72	38	424	55.64	53.23	46.9	46.9	0.00	13.5

s1494. This is because some critical directed cycles (the cycles where bubbles have to be inserted) have no early evaluation nodes. Early evaluation does not affect the performance of such ESs.

Observation 2: The RC^{lp}_{min} coincides with RC^{sim}_{min} in more than half of the examples. In s641, s386, s400, s526, s713, s953 the relative difference between $\xi(\text{RC}_{min})$ and $\xi(\text{RC}^{lp}_{min})$ ($\Delta(\%)$) is within 5%.

Observation 3: The average error $err(\%)$ of the throughput estimation is equal to 12.5%. It is calculated as the relative difference between Θ^{lp} and Θ. The error usually increases with the number of bubbles that were inserted in the RRG and achieves 35% for some configurations. Usually the error is proportional to the difference between the throughputs of an RRG with and without early evaluation nodes.

5.7 Conclusions and future work

The R&R optimization technique has been extended to ESs with EE nodes. This model is solved using a heuristics based on the exact method developed in Chapter 4 for R&R of ESs with late evaluation nodes. Experimental results show that in most cases this heuristics allows to solve the problem of R&R for ESs with EE nodes exactly. With the proposed approach the performace of an ES can be improved up to 50% with respect to the minimal cycle time achievable by the retiming technique.

The proposed MILPs are difficult to solve exactly for RRGs with more than one thousand edges. However, there are simple and efficient heuristics for solving MILP problems. Exploring such heuristics is a part of the future work. The proposed model can be extended to handle *telescopic* nodes (i.e., nodes with variable combinational delays). Recently, R&R was used as local optimization step to find an optimal number of bypasses for memory register files [48].

Chapter 6

Optimal buffer sizing

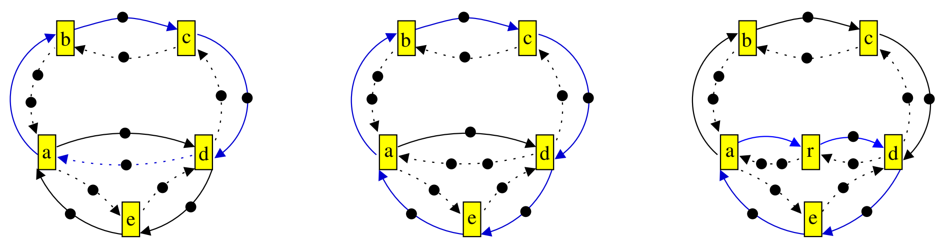
The optimization models that are presented in Chapters 4 and 5 assume that all EBs have infinite queues. If queue sizes of some EBs are not sufficiently large, the actual throughput of an ES may be significantly less than its estimation. This chapter shows how the EBs can be optimally resized to minimize the area the control logic of an ES occupies and to avoid throughput degradation. This chapter uses the elastic marked graph model to represent an ES. Notice, this model significantly differs from the retiming and recycling graph that is used in Chapters 4 and 5. The main contribution of this chapter was published in [12].

6.1 Introductory example

This section describes an example where the buffer sizing achieves higher throughput than the buffer insertion (a.k.a slack matching). Please, refer to Section 3.2.4 for the explanation of what the buffer sizing and slack matching is.

In the example that was presented in Section 3.2.4 (Figure 3.9) both transformations achieve the same throughput. This is not always the case. To construct an example, where buffer sizing does better than buffer insertion it is enough to remove vertex f from the graph 3.9(a) and put a token in the EB (c, d). Figure 6.1(a) shows the result graph, as before, vertices represent computational nodes. The pair of forward (solid line) and backward (dashed line) edges represent an EB. In the example we assume that the forward and the backward delay of each EB is equal to one. The minimal ratio tokens/delay of the graph is equal to 3/4; the critical cycle is composed by the vertices a, b, c, d. Buffer sizing removes the backward edge (d, a) from the critical cycle by incrementing the size of the EB (a, d). Figure 6.1(b) shows the result of the optimal buffer sizing; the throughput is equal to 4/5; the critical cycle $(a \to b \to c \to d \to e \to a)$ has only forward edges.

Inserting a bubble (a, r) to the "fast" branch of reconvergent paths produces a



(a) Original ES (b) Buffer sizing (c) Slack matching

Figure 6.1: ES, where buffer sizing achieves better throughput than buffer insertion.

new directed cycle ($a \to r \to d \to e \to a$) with the ratio tokens/delay is equal to 3/4. Figure 6.1(c) depicts the result of this transformation. Thus, buffer insertion cannot improve performance in this example.

The main contribution of this chapter is a general model that combines buffer sizing and buffer insertion to achieve maximal possible throughput of an ES and allows to give a priority to one of the transformations.

6.2 Elastic marked graph

This chapter models ESs with *elastic marked graphs*. The formal definition is as follows.

Definition 6.2.1 (EMG) *An elastic marked graph (EMG) is a directed connected graph $EMG = (V, E, E', T, \delta)$, where:*

- *V is the set of vertices (computational nodes of the ES)*
- *E is the set of forward edges*
- *E' is the set of backward edges*
- *$T : E \cup E' \to \mathbb{Z}^+$ assigns an initial number of tokens to each edge*
- *$\delta : E \cup E' \to \mathbb{R}^+$ is the delay of edge*

A bijection, $B : E \to E'$, exists such that for each forward edge $e = (u, v)$ there is a backward edge $e' = B(e)$ and $e' = (v, u)$. For each such a pair of forward and backward edges we define: $C_0(e) = T(e) + T(e')$. The value of $C_0(e)$ represents the EB capacity and must be greater than or equal to two, $C_0(e) \geq 2$.

Figures 6.1 and 3.9 provides examples of EMGs. The capacitance of all EBs of these EMGs is equal to two. It directly follows from the definition, that EMG have to be strongly connected. This is because if there is a directed path from vertex u to vertex v, then there is a complementary directed path from v to u. By complementary we mean the path that is composed with edges on the opposite direction. Therefore, there is a directed path in an EMG between any pair of vertices. An EB is modeled by a pair of forward and backward edges. Thus, each EB of an ES is identified by its forward edge in the EMG model. The number of tokens in the forward edge represents the number of valid data in the EB, while the number of tokens in the backward edge represents the number of free registers in the EB.

An EMG can be treated as a marked graph. Marked graphs is a convenient tool for modeling of concurrent discrete systems. They have been used in asynchronous design for several decades and also are used to model synchronous elastic systems. Section 3.2.4 provides more details about related work.

6.3 Throughput of an EMG

The performance of an ES can be estimated as the minimum cycle ratio tokens/delay of the corresponding EMG [91, 92]:

Definition 6.3.1 *The throughput, Θ, of an EMG is defined by the following equation:*

$$\Theta = \min\left\{ \min_{c \in C} \frac{\sum\limits_{e \in c} T(e)}{\sum\limits_{e \in c} \delta(e)}, \frac{1}{\delta_{max}} \right\} \tag{6.1}$$

where C is the set of directed cycles of EMG and $\delta_{max} = \max\limits_{e \in E \cup E'} \delta(e)$.

A cycle c is called *critical* if it satisfies $\Theta = \frac{\sum\limits_{e \in c} T(e)}{\sum\limits_{e \in c} \delta(e)}$. A critical cycle can contain both forward and backward edges. If a critical cycle contains backward edges the throughput of an EMG is being constrained by the capacity of the corresponding EB and therefore, it can be improved with buffer sizing and/or buffer insertion.

The maximum throughput, Θ^*, that can be achieved by buffer sizing and buffer insertion for a given EMG is upper bounded by the throughput, Θ^F, of the corresponding forward EMG [72]:

Proposition 6.3.1 *The maximum throughput, Θ^*, that can be achieved by buffer sizing and buffer insertion satisfies:*

$$\Theta^* \leq \Theta^F = \min\left\{ \min_{c \in C^F} \frac{\sum\limits_{e \in c} T(e)}{\sum\limits_{e \in c} \delta(e)}, \frac{1}{\delta_{max}} \right\} \tag{6.2}$$

where C^F is the set of directed cycles containing only forward edges of EMG.

The following theorem allows to combine buffer sizing with buffer insertion in one MILP model. The constraints that apper in the formulation of this theorem are similar to the constraints (2.4), with $W_1 = T$ and $W_2 = \delta$. The deomnstration of this theorem is a direct application of Definition 6.3.1 and Theorem 2.3.2.

Theorem 6.3.1 *The throughput of an* EMG *can be calculated with the following LP:*

$$\begin{array}{ll} maximize: \ \Theta, & \\ \textbf{\textit{subject to:}} & \\ \quad T(e) - \Theta \cdot \delta(e) \geq d(v) - d(u), & \forall e = (u,v) \in E, \\ \quad T(e') - \Theta \cdot \delta(e') \geq d(u) - d(v), & \forall e' = (v,u) \in E', \\ \quad \Theta \leq \dfrac{1}{\delta_{max}}. & \end{array} \tag{6.3}$$

Here $d \in \mathbb{R}^{|V|}$ is a $|V|$-dimensional vector of real variables.

6.4 MILP formulations

Firstly this section reviews MILP formulations for buffer sizing and buffer insertion, both models can be derived from Theorem 6.3.1. These formulations are not the contribution of this chapter. Then, a single MILP formulation is proposed to combine both techniques. Finally, a method based on a binary search is described to search for the maximum achievable throughput.

6.4.1 MILP: Buffer sizing

The capacity of an EB is equal to the sum of the tokens on the corresponding forward and backward edges. By increasing the capacity of an EB we increase the number of tokens on the corresponding backward edge e', since the number of valid data in each EB should be unchanged to preserve the original behavior. For instance, the EB in Figure 6.2(a) has capacity 2 and the addition of 1 token to $e' = (v, u)$ increases its capacity to 3, see Figure 6.2(b). The capacity of an EB after sizing is denoted by $C(e)$.

The following model for buffer sizing of ESs takes into account that an increase of a given EB capacity entails a more complex control logic, and in turn an increase of the corresponding forward delay. Such an increase of the delay is usually logarithmic with respect to the increase of the capacity, however, linear approximations work also reasonably well for small values of the capacity and EBs with big capacity have no

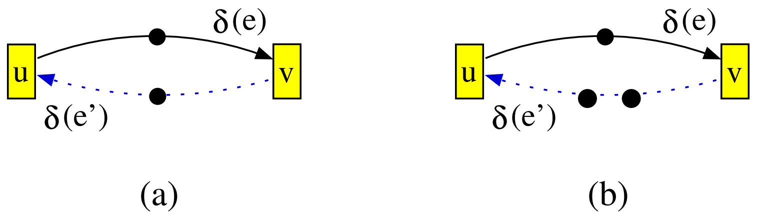


Figure 6.2: (a) Original buffer with capacity 2, i.e., $C_0(e) = 2$; (b) Buffer is resized to capacity 3, i.e., $C(e) = 3$.

practical interest. Let us assume that the EB corresponding to edge e is sized from capacity $C_0(e)$ to $C(e)$, then its new forward delay, $\delta_c(e)$, can be calculated as follows:

$$\delta_c(e) = \delta(e) + h \cdot (C(e) - C_0(e)) \tag{6.4}$$

where h is a real parameter to adjust the linear approximation of the increase of delay with respect to the increase of capacity. Obviously, if $h = 0$ the new delay is independent of the new capacity.

Capacity-dependent delays, δ_c, can be included in Theorem 6.3.1 just by replacing $\delta(e)$ with $\delta_c(e)$. Then, for a given $\Theta_g \leq \dfrac{1}{\delta_{max}}$ the minimum sum of capacities that can achieve throughput Θ_g can be obtained by solving the following MILP:

$$\begin{array}{lll}
\multicolumn{3}{l}{minimize: \ \sum\limits_{e \in E} C(e),} \\
\multicolumn{3}{l}{\textbf{subject to:}} \\
\quad T(e) - \Theta_g \cdot \delta_c(e) \geq d(v) - d(u), & \forall e = (u,v) \in E, & \\
\quad C(e) - T(e) - \Theta_g \cdot \delta_c(e') \geq d(u) - d(v), & \forall e = (u,v) \in E, & (6.5) \\
\quad C(e) \in \mathbb{Z}^+, C(e) \geq C_0(e), & \forall e \in E. &
\end{array}$$

where $C_0(e)$ is the initial capacity of the EB, and $C(e)$ is a variable determining the new capacity of the EB. The MILP (6.5) is feasible iff Θ_g is achievable by buffer sizing. If all delays in (6.5) are equal to one and $h = 0$ then (6.5) is equivalent to the model for buffer sizing of latency insensitive systems proposed in [71]. Also notice, that if h is equal to zero, then MILP (6.5) is feasible for $\Theta_g = \Theta^F$, where Θ^F is equal to the throughput of the forward EMG (see Proposition 6.3.1).

6.4.2 MILP: Buffer insertion

In order to describe the insertion of a bubble to the EMG, one more edge weight function should be introduced to it: $N : E \to \mathbb{Z}^+$. This function specifies the number of bubbles, that are added after main EB. For example, all the edges of the EMG in Figure 6.1(c) have no extra EBs except (a, d), which is pipelined with one bubble

(a, r), that is $N((a, d)) = 1$. Figure 6.3 exemplifies how the EMG is modified when two bubbles are inserted.

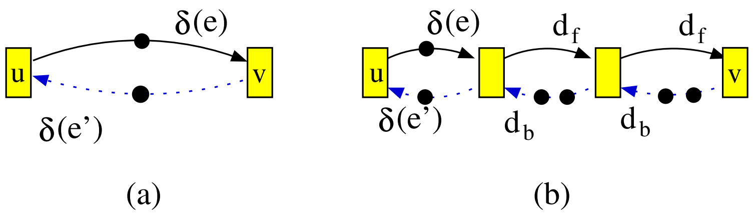


Figure 6.3: (a) Original EB with capacity 2; (b) Result of inserting 2 bubble, i.e., $N(e) = 2$.

After buffer insertion, the total number of tokens on the backward edges from v to u is equal to $C(e) - T(e) + C_b \cdot N(e)$, where C_b is the capacity of the inserted EBs. For example, $C_b = 2$ for latency insensitive systems. Let d_f and d_b be the forward and backward delays of the inserted bubbles. Then, the total delay of the forward edges from u to v is equal to $\delta(e) + d_f \cdot N(e)$, and the overall delay of the backward edges from v to u is $\delta(e') + d_b \cdot N(e)$. The use of Theorem 6.3.1 allows to design an MILP that minimizes the number of bubbles that must be inserted to achieve a given throughput $\Theta_g \le \dfrac{1}{\delta_{max}}$:

$$\begin{array}{ll}
\textit{minimize}: \sum_{e \in E} N(e) & \\
\textbf{subject to:} & \\
\quad T(e) - \Theta_g \cdot (\delta(e) + d_f \cdot N(e)) \ge d(v) - d(u), & \forall e = (u, v) \in E, \\
\quad C_0(e) - T(e) + C_b \cdot N(e) & \\
\quad -\Theta_g \cdot (\delta(e') + d_b \cdot N(e)) \ge d(u) - d(v), & \forall e = (u, v) \in E, \\
\quad N(e) \in \mathbb{Z}^+, & \forall e = (u, v) \in E.
\end{array} \tag{6.6}$$

where $N(e)$ is the number of bubbles inserted in edge e. Capacity dependent delays, δ_c, are not considered in (6.6) because capacities are kept to their initial values C_0. The MILP (6.6) is feasible iff Θ_g is achievable by buffer insertion. The formulation in (6.6) is equivalent to the one obtained in [4] for slack matching of asynchronous design.

6.4.3 MILP: buffer resizing and buffer insertion

The buffer insertion and buffer sizing transformations have their own advantages and disadvantages: buffer sizing might achieve a higher throughput than buffer insertion but the complexity of the required logic increases with the size of the computed capacities; buffer insertion does not increase the complexity of the control logic but it might achieve a lower throughput and also increases the latency of the ES. It is always possible to construct an example of an EMG where the value of the objective function of the optimal solution of MILP (6.5) is equal to one, while the value of the objective function of the optimal solution of MILP (6.6) is equal to K, for any integer K. Thus, both transformations provide the same throughput, but buffer insertion may lead to large area overhead.

This section proposes an MILP that aims at combining both techniques to minimize the cost involved by the implementation of the solution. The cost can refer to any index related to the sizing and insertion of EBs, for example, area required to resize an EB (insert a new EB), complexity of the control logic required to resize an EB (insert a new EB), etc.

In order to combine both techniques, we will make use of a real parameter $\alpha > 0$ that represents the ratio of the cost required to increase in one unit the capacity of an EB to the cost required to insert a bubble between two nodes. If $\alpha = 1$ ($\alpha < 1$) ($\alpha > 1$) the cost required to increase in one the EB capacity is assumed to be equal to (less than) (greater than) the cost to insert one bubble. For instance, if one desires to minimize the area of the implementation, and the area required to insert one bubble is half the area required to increase the capacity of an EB in one unit, then α must be set to 2. For a given $\Theta_g \leq \frac{1}{\delta_{max}}$ the solution of the following MILP provides simultaneously a sizing and an insertion transformation that achieves Θ_g and minimizes the overall cost.

$$
\begin{array}{lll}
\multicolumn{3}{l}{\textit{minimize}: \ \alpha \cdot \sum\limits_{e \in E} C(e) + \sum\limits_{e \in E} N(e)} \\
\multicolumn{3}{l}{\textbf{subject to:}} \\
\quad T(e) - \Theta_g \cdot (\delta_c(e) + d_f \cdot N(e)) \geq d(v) - d(u), & \forall e = (u,v) \in E, & \\
\quad C(e) - T(e) + C_b \cdot N(e) & & \\
\quad -\Theta_g \cdot (\delta_c(e') + d_b \cdot N(e)) \geq d(u) - d(v), & \forall e = (u,v) \in E, & \\
\quad N(e), C(e) \in \mathbb{Z}^+, C(e) \geq C_0, & \forall e = (u,v) \in E. &
\end{array}
\tag{6.7}
$$

where $C(e)$ is the capacity of the EB e, and $N(e)$ is the number of bubbles inserted after main EB e. The MILP (6.7) is feasible iff Θ_g is achievable by buffer sizing and buffer insertion. Notice that even a very high α is chosen, it might not be possible to achieve Θ_g just by inserting bubbles. In general, the throughput of forward EMG can not be always achieved with both transformations. This might happen if buffer sizing significantly increases the delay of the backward edge, i.e., the value of the parameter h in the equation (6.4) is large. In circuits this is not a real problem, but to be rigorous let us show how the maximal achievable throughput can be found.

6.4.4 Search for the maximum throughput

The use of MILP (6.7) greatly simplifies the search for the maximum throughput, Θ^*, achievable with buffer sizing and buffer insertion. By Proposition 6.3.1, Θ^* is upper-bounded by Θ^F, i.e., $\Theta^* \leq \Theta^F$. On the other hand, Θ^* is obviously greater than or equal to the throughput, Θ, of the original EMG which is given by (6.1), then $\Theta \leq \Theta^* \leq \Theta^F$.

In this way, the interval $[\Theta, \Theta^F]$ can be taken to perform a binary search for Θ^*: if (6.7) is feasible for a given $\Theta_1 \in [\Theta, \Theta^F]$ then $\Theta_1 \leq \Theta^* \leq \Theta^F$ and the interval $[\Theta_1, \Theta^F]$ is taken for the next search. If (6.7) is not feasible for Θ_1 then $\Theta \leq \Theta^* \leq \Theta_1$ and the interval to be considered is $[\Theta, \Theta_1]$. This procedure can be repeated until a satisfactory precision for Θ^* is obtained.

It is always a good idea to try first $\Theta^* = \Theta^F$ and if the problem is infeasible then to apply the binary search. In synchronous elastic systems the delays of forward and backward edge represent the minimal forward and backward latency of the EB. Hence, the delay cannot be increased by buffer sizing and the proposed binary search is never needed if an EMG models a synchronous elastic circuit.

The proposed MILP might be difficult to solve for some cases, since it is well known that MILP is an NP-complete problem [51]. Next we prove that the decision version of buffer sizing problem is NP-hard and hence, the reduction to MILP is one of the most efficient ways to solve it.

6.5 Buffer sizing is NP-complete

The minimal buffer sizing is minimization problem. The decision version of buffer sizing is as follows:
Given an EMG and an integer constant $K > 0$. Is it possible to achieve the maximal throughput with total buffer capacity increment of at most K?

Theorem 6.5.1 *The decision version of buffer sizing is NP-complete.*

Proof: To demonstrate the statement of the theorem it must be shown that:

- The problem is in NP.
- The problem is in NP-hard.

To see that the problem is in NP it is enough to notice that the throughput of an EMG can be calculated in polynomial time with a minimal cycle ratio algorithm [42].

To show that the decision version of buffer sizing is NP-hard we will reduce the *minimal feedback edges set* problem, which is known to be NP-complete [51], to it.

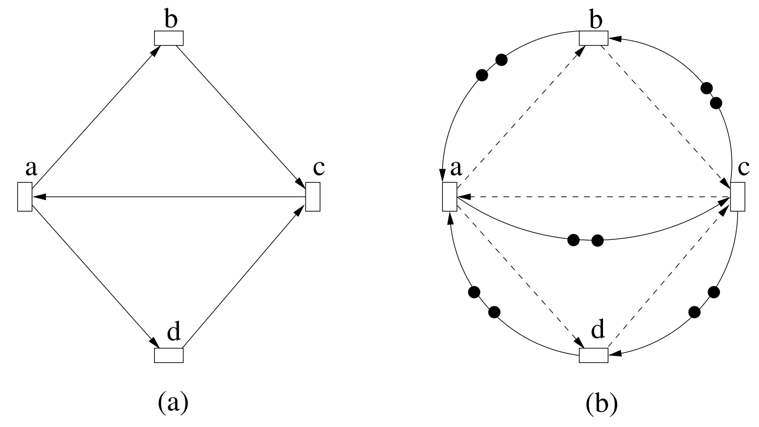


Figure 6.4: (a) Original directed graph; (b) Corresponding EMG.

A feedback edges set is a set of edges which, when removed from the graph, leave a directed acyclic graph (DAG). The decision version of the problem is as follows:
Given directed graph $G = (V, A)$, where V is a set of vertices and A is a set of edges (arcs) and integer number $K > 0$. Is there $A_f \subset A, |A_f| \leq K$ such that subgraph $G' = (V, A - A_f)$ is a DAG?

The reduction is as follows: for each edge $a \in A, a = (u, v)$ of the graph G we add a forward edge $e = (v, u)$ and set $T(e) = 2, T(a) = 0, \delta(e) = 1, \delta(a) = 1$. Figure 6.4(a) shows a directed graph, and Figure 6.4(b) shows the corresponding EMG.

The following observations conclude the demonstration:

- The reduction has a linear time complexity with respect to the number of the edges in G.
- The initial throughput of the EMG is equal to zero. If it is not, then G has no directed cycles.
- If for some cycle of the EMG the tokens to delay ratio is greater than zero, then it is greater than $\frac{1}{2|A|}$ as well.
- A minimal buffer sizing of the result EMG that yields the throughput of at least $\frac{1}{2|A|}$ provides minimal feedback edges set of the G.

■

The provided reduction in fact allows to show that a typical case of buffer sizing is not a difficult to solve problem. Since, the combinatorial optimization should be applied only to the *critical subgraph* of the EMG. A critical subgraph is a subgraph such that any cycle in it has a ratio tokens/delay equal to the throughput of the EMG. The most common structure of a critical subgraph is a graph with one simple cycle. For such a graph the minimal feedback edges problem is trivial to solve.

The related problem was studied in the world of timed marked graphs. It is called *minimal initial marking* [94] or *optimal token allocation* [53] problem. The minimal feedback edges set problem can also be reduced to these problems in a similar manner.

6.6 Experimental results

6.6.1 Generation of EMGs

Two sets of EMGs have been used to evaluate the method presented in this chapter. On the one hand, the ISCAS89 circuits have been used to extract the underlying graphs. On other hand, some random EMGs have also been generated (lines from ge1 to ge10 in the table of results). The tokens, EB capacities and delays have been generated as follows:

Initial tokens: Each forward edge e is assigned a token ($T(e) = 1$) with probability 0.9.

Capacities: The initial capacity of each edge e is 2 ($C_0(e) = 2$). The capacity of the inserted bubbles is 2 as well ($C_b = 2$).

Delays: Each forward delay e is generated randomly, more precisely, $\delta(e)$ is a real number obtained from a random uniform variable in the interval $[1,\ 3]$. The forward delay of the inserted bubbles is 1 ($d_f = 1$). All backward delays are set to 1 ($\delta(e') = 1$) for each backward edge e', and $d_b = 1$ for the inserted bubbles. The dependence of the delay on the capacity is modeled using Equation (6.4) with $h = 0.5$.

In order to test buffer sizing and buffer insertion, every EMG in Table 6.1 satisfies $\Theta < \Theta^F$, what implies that its throughput can be improved. We were generating EMGs until this constraint is verified.

6.6.2 Results

For each test case, buffer sizing and buffer insertion solutions that provide a maximal throughput were obtained by using the procedure described in Subsection 6.4.4. Two different values of α were used, 6/5 and 5/6, in order to test different priorities for buffer sizing and buffer insertion. In most of the tests the actual values of α do not change the obtained results. Examples, where the value of α changes the results, are explicitly discussed.

Table 6.1 reports the obtained results. Columns $|V|$ and $|E|$ are the number of vertices and forward edges in an EMG respectively. Column "Θ" is the throughput of the EMG computed with (6.1). Column "Θ^*" provides the maximal throughput achieved with buffer sizing and buffer insertion. Column "$\Sigma\Delta N$" reports the number of inserted bubbles. Column "$\Sigma\Delta C$" reports the overall increase of capacities, $\sum\limits_{e\in E}(C(e) - C_0(e))$.

Observation 1: In every case, except ge5 and ge6, the throughput upper bound, Θ^F, provided by (6.2) is achievable, i.e., $\Theta^* = \Theta^F$. For ge5 $\Theta^F = 0.4037$ and $\Theta^* = 0.4025$; for ge6 $\Theta^F = 0.4653$ and $\Theta^* = 0.4643$, the precision of Θ^* obtained by the binary search was set to 0.01%. That is, the increase of delays due to buffer sizing avoids achieving Θ^F only in ge5 and ge6 (notice, however, that $\Theta^F - \Theta^*$ is very small in both cases) .

Observation 2: In most cases Θ^* can be achieved only by buffer insertion, see columns "$\Sigma\Delta N$" and "$\Sigma\Delta C$" below $\alpha = 6/5$. Only ge1, ge2, ge3, and ge4 require buffer sizing to achieve Θ^* (this happens even if arbitrarily high values of α are considered).

Observation 3: With $\alpha = 5/6$, Θ^* is achieved only by buffer sizing in all cases except in s27, s400 and ge7. Nevertheless, if α is set to 0.5, i.e., higher cost is given to buffer insertion, the maximum throughput of s27, s400 and ge7 is achieved only by buffer sizing with $\Sigma\Delta C$ equal to 12, 9 and 10 respectively.

Observation 4: Although MILP problems are NP-complete, the CPU times spent for the largest test cases ge7, ge8, ge9 and ge10 were 10, 15, 45 and 540 seconds respectively. This because only the variables corresponding to cycles with minimum cycle ratio equal to Θ are relevant in the objective function, the rest of variables are set to 0 in the first steps performed by the solver. The CPU time for the rest of cases was less than one second.

All the experiments were run on Linux system with XEON 3.8GHz CPU and 2Gb of memory per process. CPLEX [58] was used as MILP solver.

6.7 Conclustions and future work

The provided solution does not handle early evaluation. MILP (6.7) should be revised to handle early evaluation nodes. Again, guarded marked graphs presented in [59] may help to deal with this problem, as it was done in the second contribution [10].

This work does not show how the EMG model can be derived from an RRG. This is because the EMG model depends on the details of the implementation of the ES.

Table 6.1: Experimental results

	$\|V\|$	$\|E\|$	Θ	Θ^*	Buffer sizing & buffer insertion			
					$\alpha = 6/5$		$\alpha = 5/6$	
					$\Sigma\Delta N$	$\Sigma\Delta C$	$\Sigma\Delta N$	$\Sigma\Delta C$
s27	31	78	0.2634	0.3399	11	0	11	0
s298	823	7154	0.0341	0.0383	37	0	0	37
s349	139	241	0.2110	0.2663	2	0	0	2
s400	119	273	0.1963	0.2892	8	0	1	7
s526	145	382	0.1215	0.1835	2	0	0	2
s641	182	298	0.2062	0.2855	2	0	0	2
s713	208	350	0.2935	0.3099	1	0	0	1
s820	183	919	0.1094	0.1320	10	0	0	10
s832	191	972	0.1181	0.1356	7	0	0	7
s953	373	704	0.2887	0.3233	15	0	0	15
s1423	484	942	0.1529	0.1784	1	0	0	1
s1488	321	1662	0.0716	0.0867	1	0	0	1
s1494	341	1775	0.0639	0.0911	22	0	0	22
s5378	1138	2484	0.2204	0.2527	5	0	0	5
s9234	1023	1992	0.1813	0.2133	3	0	0	3
ge1	10	30	0.5288	0.5897	3	1	0	4
ge2	20	50	0.5484	0.5916	2	1	0	3
ge3	70	100	0.5467	0.5785	2	1	0	3
ge4	30	100	0.5317	0.6470	6	1	0	7
ge5	20	100	0.3373	0.4025	1	0	0	1
ge6	10	30	0.4390	0.4642	1	0	0	1
ge7	5000	20000	0.0942	0.1250	9	0	1	8
ge8	10000	50000	0.0662	0.0834	8	0	0	8
ge9	20000	100000	0.0675	0.0857	10	0	0	10
ge10	50000	500000	0.0146	0.0220	8	0	0	8

Chapter 7

Conclusions and future work

This chapter draws general conclusions about the contributions of this dissertation. Next, two directions for the future investigation are described: elastization of a synchronous design and performance evaluation of complex ESs.

This work provides formal methods for performance evaluation and optimization of elastic systems. All the contributions are general and can be useful both in synchronous and asynchronous domains.

In this work elastic systems (ES) are represented with simple abstract models based on graphs. Such models can be derived at different stages of the design flow. They allow to use simple and yet powerful formal techniques for the analysis and optimization of ESs.

Retiming and recycling hardly improves the performance of an ES with only late evaluation nodes. The insertion of bubbles, indeed, breaks the long interconnections, but also decreases the throughput of the ES. If a critical interconnection belongs to a directed cycle with a small latency, the insertion of one bubble may decrease the throughput by 50%. Fortunately, it turns out that early evaluation may help in this situation. Speculative execution is another technique to increase the throughput of ESs [49].

Buffer sizing is a nice, efficient and easy to implement technique which allows to remove extra performance lost in ESs. As we have already mentioned, this technique should be adapted to ESs with early evaluation and variable latency nodes. Optimal capacitances of EBs can be greater than two. An efficient implementation of such buffers are welcome. The paper [70] defines the interface for buffers of capacity $N \geq 2$. Combining this interface with the design of a synchronization queue proposed in [28] the desired implementation can be derived. Probably, it can still be optimized.

Elastic design looks promising for the deep nano-technologies. This work can be treated as a preliminary step toward making it widely accepted. More investigation on this topic is required to bring it to the industry. It is not an easy task to change a well known and well supported synchronous methodology.

As a future work, the theory presented in this document should be applied for a real design. This seems to be quite a big project. Let us outline some issues that should be addressed in this sense:

1. Design partitioning. To build retiming and recycling graph from the high level description (HDL) of a circuit it is necessary to know which part of this description can be represented as a vertex in the graph. This process might be difficult to automate. The information about candidates on early evaluation and variable latency nodes resides mostly in the mind of the designer. The best approach would be to start from a well structured HDL code. In such a code the nodes of retiming and recycling graph will be the modules of the system, e.g, ALUs, memory register file, multiplexers, etc.

2. Input parameters for the optimization such as: delays of the nodes, probabilities for taking branches or several cycles for multi cycle operations. The timing of the design at the behavioral level may be significantly different from the timing of the layout. All contemporary synchronous designs face with this problem and the elastic methodology is not an exception. To obtain the probabilities one can use the numbers that were derived from the profiling of a non-elastic implementation or just some realistic numbers.

Recently, a progress in this direction was made by the Intel office in Moscow [55].

Another interesting direction to explore is *ES performance evaluation*. The performance evaluation of ESs can not be closed at this step. Marked graphs are only capable of describing simple ESs. There are new challenges arise when modeling ESs with early evaluation and variable latency nodes and active anti-tokens. For example, in [50] a method based on the combination of unfolding and symbolic max-plus algebra to calculate throughput bounds of ES with early evaluation and variable latency nodes is provided.

Simulation should not be discarded as a powerful tool for ESs performance estimation. For example, the performance of an ES with early evaluation, variable latency nodes and active anti-tokens propagation mechanism can be calculated using its definition, $\lim\limits_{t\to\infty} \sigma(t)/t$, where $\sigma(t)$ is the number of times the given node has been fired at the time stamp t. In [33] (Chapter 3) a simple recurrence for the $\sigma(t)$ is described. This recurrence allows to calculate exactly the throughput of a stochastic guarded marked graph with several thousands of places within a second. Each iteration of the simulation of a marked graph has linear complexity with respect to the number of edges.

The optimization models proposed in Chapters 4 and 5 can be extended by incorporating variable latency nodes and memory register file bypass transformation. This can be easily achieved by modeling the *OR* relation between different groups of constraints in mixed integer linear programming. A question to answer is how to estimate the throughput of an ES with both early evaluation and variable latency nodes. It can be shown that marked graphs can not model a general ES of such type. Simulation can be used to estimate the throughput but it does not provide a way to optimize the performance. Thus, formal methods for performance estimation of ESs of this kind are welcomed.

Bibliography

[1] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger. Clock rate versus IPC: the end of the road for conventional microarchitectures. *SIGARCH Computer Architecture News*, 28(2):248–259, 2000.

[2] M. Ampalam and M. Singh. Counterflow pipelining: Architectural support for preemption in asynchronous systems using anti-tokens. In *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pages 611–618, Nov. 2006.

[3] D. Baneres, J. Cortadella, and M. Kishinevsky. Variable-latency design by function speculation. In *Proceedings of Design, Automation and Test in Europe (DATE)*, pages 1704–1709, Mar. 2009.

[4] P. A. Beerel, N.-H. Kim, A. Lines, and M. Davies. Slack matching asynchronous designs. In *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 184–194, Mar. 2006.

[5] P. A. Beerel, R. O. Ozdag, and M. Ferretti. *A Designer's Guide to Asynchronous VLSI.* Cambridge University Press, 2010.

[6] L. Benini, E. Macii, and M. Poncino. Telescopic units: increasing the average throughput of pipelined designs by adaptive latency control. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 22–27, June 1997.

[7] J. Boucaron, R. de Simone, and J.-V. Millo. Latency-insensitive design and central repetitive scheduling. In *4th ACM & IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE 2006)*, pages 175–183, July 2006.

[8] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[9] C. F. Brej. *Early Output Logic and Anti-Tokens.* PhD thesis, University of Manchester, 2005.

[10] D. Bufistov, J. Cortadella, M. Galceran-Oms, J. Júlvez, and M. Kishinevsky. Retiming and recycling for elastic systems with early evaluation. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 288–291, July 2009.

[11] D. Bufistov, J. Cortadella, M. Kishinevsky, and S. Sapatnekar. A general model for performance optimization of sequential systems. In *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pages 362–369, Nov. 2007.

[12] D. Bufistov, J. Júlvez, and J. Cortadella. Performance optimization of elastic systems using buffer resizing and buffer insertion. In *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pages 442–448, Nov. 2008.

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

[14] S. M. Burns and A. J. Martin. Performance analysis and optimization of asynchronous circuits. In *Advanced Research in VLSI*, pages 71–86, 1991.

[15] J. Campos, G. Chiola, J. M. Colom, and M. Silva. Properties and performance bounds for timed marked graphs. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 39(5):386–401, 1992.

[16] L. P. Carloni, K. L. McMillan, A. Saldanha, and A. L. Sangiovanni-Vincentelli. A methodology for correct-by-construction latency-insensitive design. In *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pages 309–315, Nov. 1999.

[17] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Transactions on Computer-Aided Design*, 20(9):1059–1076, Sept. 2001.

[18] L. P. Carloni and A. L. Sangiovanni-Vincentelli. Coping with latency in SoC design. *IEEE Micro, Special Issue on Systems on Chip*, 22(5):24–35, Oct. 2002.

[19] L. P. Carloni and A. L. Sangiovanni-Vincentelli. Combining retiming and recycling to optimize the performance of synchronous circuits. In *16th Symposium on Integrated Circuits and System Design (SBCCI)*, pages 47–52, Sept. 2003.

[20] J. Carmona, J. Cortadella, M. Kishinevsky, and A. Taubin. Elastic circuits. *IEEE Transactions on Computer-Aided Design*, 28(10):1437–1455, Oct. 2009.

[21] J. Carmona, J. Júlvez, J. Cortadella, and M. Kishinevsky. Scheduling synchronous elastic designs. In *Proceedings of the 2009 Application of Concurrency to System Design conference (ACSD 2009)*, pages 52 – 59, July 2009.

[22] M. R. Casu. Improving synchronous elastic circuits: Token cages and half-buffer retiming. In *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 128–137, May 2010.

[23] M. R. Casu and L. Macchiarulo. A new approach to latency insensitive design. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 576–581, June 2004.

[24] M. R. Casu and L. Macchiarulo. A new system design methodology for wire pipelined soc. In *Proceedings of Design, Automation and Test in Europe (DATE)*, volume 2, pages 944–945, Mar. 2005.

[25] M. R. Casu and L. Macchiarulo. Adaptive latency-insensitive protocols. *IEEE Design and Test of Computers*, 24(5):442–452, 2007.

[26] M. R. Casu and L. Macchiarulo. Adaptive latency insensitive protocols and elastic circuits with early evaluation: A comparative analysis. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 245:35–50, 2009.

[27] T. Chelcea and S. M. Nowick. Robust interfaces for mixed-timing systems with application to latency-insensitive protocols. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 21–26, June 2001.

[28] T. Chelcea and S. M. Nowick. Robust interfaces for mixed-timing systems. *IEEE Transactions on VLSI Systems*, 12(8):857–873, 2004.

[29] C.-H. Chen and C.-Y. Tsui. Timing optimization of logic network using gate duplication. In *Proceedings of Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 233–236, Jan. 1999.

[30] C.-P. Chen, C. C. N. Chu, and D. F. Wong. Fast and exact simultaneous gate and wire sizing by lagrangian relaxation. In *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pages 617–624, Nov. 1998.

[31] Y. Chen, H. Li, J. Li, and C.-K. Koh. Variable-latency adder (vl-adder): new arithmetic circuit design practice to overcome nbti. In *International Symposium on Low Power Electronics and Design*, pages 195–200, Aug. 2007.

[32] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. M. Gettrick, and J.-P. Quadrat. Numerical computation of spectral elements in max-plus algebra. *Proceedings of the IFAC Conference on System Structure and Control*, July 1998.

[33] G. Cohen. *Analisis Y Control De Sistemas De Eventos Discretos : De Redes De Petri Temporizadas Al Algebra.* Rosario, Repuˊblica Argentina: Universidad Nacional de Rosario, Facultad de Ciencias Exactas, Ingenieriá y Agrimensura (en espanõl), 2001.

[34] R. Collins and L. P. Carloni. Topology-based optimization of maximal sustainable throughput in a latency-insensitive system. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 410–416, June 2007.

[35] J. Cong. Challenges and opportunities for design innovations in nanometer technologies. *Invited Semiconductor Research Corporation Design Sciences Concept Paper*, pages 1–15, may 1998.

[36] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* The MIT Press, 2001.

[37] J. Cortadella and M. Kishinevsky. Synchronous elastic circuits with early evaluation and token counterflow. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 416–419, June 2007.

[38] J. Cortadella, M. Kishinevsky, and B. Grundmann. Synthesis of synchronous elastic architectures. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 657–662, July 2006.

[39] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, Mar. 1997.

[40] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic synthesis of asynchronous controllers and interfaces.* Springer-Verlag, 2002.

[41] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou. Desynchronization: Synthesis of asynchronous circuits from synchronous specifications. *IEEE Transactions on Computer-Aided Design*, 25(10):1904–1921, Oct. 2006.

[42] A. Dasdan, S. Irani, and R. Gupta. Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 37–42, June 1999.

[43] A. Davis and S. M. Nowick. An Introduction to Asynchronous Circuit Design. Technical Report UUCS-97-013, Computer Science Department, University of Utah, September 1997. http://www1.cs.columbia.edu/async/publications/davis-nowick-intro-tr.pdf.

[44] M. P. Desai, R. Cvijetic, and J. Jensen. Sizing of clock distribution networks for high performance cpu chips. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 389–394, 1996.

[45] K. Fazel, L. Li, M. Thornton, R. B. Reese, and C. Traver. Performance enhancement in phased logic circuits using automatic slack-matching buffer insertion. In *Proceedings of eat Lakes Symposium on VLSI*, pages 413–416, 2004.

[46] J. P. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, 39(7):945–951, 1990.

[47] S. Furber. Computing without clocks: Micropipelining the ARM processor. In G. Birtwistle and A. Davis, editors, *Asynchronous Digital Circuit Design*, Workshops in Computing, pages 211–262, 1995.

[48] M. Galceran-Oms, J. Cortadella, D. Bufistov, and M. Kishinevsky. Automatic microarchitectural pipelining. In *Proceedings of Design, Automation and Test in Europe (DATE)*, pages 961–965, Mar. 2010.

[49] M. Galceran-Oms, J. Cortadella, and M. Kishinevsky. Speculation in elastic systems. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 292–295, July 2009.

[50] M. Galceran-Oms, J. Cortadella, and M. Kishinevsky. Symbolic Performance Analysis of Elastic Systems. In *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, Nov. 2010. (To appear).

[51] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[52] S. H. Gerez. *Algorithms for VLSI Design Automation*. Addison-Wesley, 1998.

[53] A. Giua, A. Piccaluga, and C. Seatzu. Optimal token allocation in timed cyclic event-graphs. In *Proceedings of h International Workshop on Discrete Event Systems*, pages 209–218, 2000.

[54] GNU Linear Programming Kit. Available from http://www.gnu.org/software/glpk/glpk.html.

[55] A. Gotmanov, M. Kishinevsky, and M. Galceran-Oms. Evaluation of flexible latencies: designing synchronous elastic H.264 CABAC decoder. In *Proceedings of the Problems in design of micro- and nano-electronic systems (in Russian)*, Oct. 2010.

[56] J. L. Hennessy and D. Patterson. *Computer Architecture: a Quantitative Approach*. Morgan Kaufmann Publisher Inc., 1990.

[57] A. P. Hurst, A. Mishchenko, and R. K. Brayton. Scalable min-register retiming under timing and initializability constraints. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 534–539, June 2008.

[58] ILOG. CPLEX. Available from http://www.ilog.com.

[59] J. Júlvez, J. Cortadella, and M. Kishinevsky. Performance analysis of concurrent systems with early evaluation. In *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pages 448 – 455, Nov. 2006.

[60] J. Júlvez, J. Cortadella, and M. Kishinevsky. On the performance evaluation of multi-guarded marked graphs with single-server semantics. *Discrete Event Dynamic Systems*, 20(3):377–407, 2010.

[61] T. Kam, M. Kishinevsky, J. Cortadella, and M. Galceran-Oms. Correct-by-construction microarchitectural pipelining. In *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pages 434–441, Nov. 2008.

[62] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984.

[63] R. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[64] L. Khachiyan. A polynomial algorithm for linear programming (Russian). In *Doklady Akademii Nauk SSSR 244*, pages 1093–1096, 1979.

[65] C. E. Leiserson and J. B. Saxe. Optimizing synchronous systems. In *22nd Annual Symposium on Foundations of Computer Science*, pages 23–36, 1981.

[66] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, 1991.

[67] C.-H. Li and L. P. Carloni. Using functional independence conditions to optimize the performance of latency-insensitive systems. In *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pages 32–39, Nov. 2007.

[68] D. J. Lilja and S. S. Sapatnekar. *Design Digital Computer Sysyems with Verilog.* Cambridge University Press, 2005.

[69] Y. Liu, S. R. Nassif, L. T. Pileggi, and A. J. Strojwas. Impact of interconnect variations on the clock skew of a gigahertz microprocessor. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 168–171, June 2000.

[70] R. Lu and C.-K. Koh. Performance analysis and efficient implementation of latency insensitive systems. ECE Technical Reports, Mar. 2003.

[71] R. Lu and C.-K. Koh. Performance optimization of latency insensitive systems through buffer queue sizing of communication channels. In *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, pages 227–231, Nov. 2003.

[72] R. Lu and C.-K. Koh. Performance analysis of latency-insensitive systems. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 25(3):469 – 483, 2006.

[73] R. Manohar and A. J. Martin. Slack elasticity in concurrent computing. In J. Jeuring, editor, *Proceedings of h International Conference on the Mathematics of Program Construction*, volume 1422 of *Lecture Notes in Computer Science*, pages 272–285, 1998.

[74] A. J. Martin, A. Lines, R. Manohar, M. Nystrom, P. Penzes, R. Southworth, U. Cummings, and T. K. Lee. The design of an asynchronous mips r3000 microprocessor. In *Advanced Research in VLSI*, pages 164–181, 1997.

[75] D. Matzke. Will physical scalability sabotage performance gains? *Computer*, 30(9):37–39, 1997.

[76] Little, J.D.C. A proof of the queuing formula $L = \lambda W$. *Operations Research*, 9:383–387, 1961.

[77] G. D. Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw Hill, 1994.

[78] D. E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switching*, pages 204–243, Apr. 1959.

[79] T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, pages 541–580, Apr. 1989. NewsletterInfo: 33Published as Proceedings of the IEEE, volume 77, number 4.

[80] R. Murgai. Efficient global fanout optimization algorithms. In *Proceedings of Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 571–576, Jan. 2001.

[81] K. G. Murty. *Linear Programming.* Wiley and Sons, 1983.

[82] S. R. Nassif. Design for variability in dsm technologies. In *Proceedings of the 1st International Symposium on Quality of Electronic Design (ISQED)*, pages 451–454, 2000.

[83] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1995.

[84] M. C. Papaefthymiou. Understanding retiming through maximum average-delay cycles. *Mathematical Systems Theory*, 27(1):65–84, 1994.

[85] T. Pering, T. Burd, and R. Brodersen. Dynamic voltage scaling and the design of a low-power microprocessor system. In *In Power Driven Microarchitecture Workshop, attached to ISCA98*, 1998.

[86] C. A. Petri. *Kommunikation mit Automaten.* Bonn: Institut fur Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

[87] C. A. Petri. Kommunikation mit automaten. *New York: Griffiss Air Force Base, Technical Report RADC-TR-65–377*, 1:1–Suppl. 1, 1966. English translation.

[88] M. D. Powell, A. Biswas, J. S. Emer, S. S. Mukherjee, B. R. Sheikh, and S. Yardi. CAMP: A technique to estimate per-structure power at run-time using a few simple parameters. In *IEEE 15th International Symposium on High Performance Computer Architecture (HPCA)*, pages 289–300, Feb. 2009.

[89] H. J. Prabhakar, H. M. Jacobson, P. N. Kudva, P. Bose, P. W. Cook, S. E. Schuster, E. G. Mercer, and C. J. Myers. Synchronous interlocked pipelines. In *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 3–12, Mar. 2001.

[90] P. Prakash and A. J. Martin. Slack matching quasi delay-insensitive circuits. In *Proceedings of International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 195–204, Mar. 2006.

[91] C. V. Ramamoorthy and G. S. Ho. Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets. *IEEE Transactions on Software Engineering*, 6(5):440–449, 1980.

[92] C. Ramchandani. Analysis of asynchronous concurrent systems by timed Petri nets. Technical Report Project MAC Tech. Rep. 120, Massachusetts Inst. of Tech., Feb. 1974.

[93] R. Reese, M. Thornton, C. Traver, and D. Hemmendinger. Early evaluation for performance enhancement in phased logic. *IEEE Transactions on Computer-Aided Design*, 24(4):532–550, Apr. 2005.

[94] J. Rodriquez-Beltran and A. Ramirez-Trevino. Minimum initial marking in timed marked graphs. In *Proceedings of EE International Conference on Systems, Man, and Cybernetics (SMC'2000)*, volume 4, pages 3004–3008, Oct. 2000.

[95] S. S. Sapatnekar. *Timing.* Kluwer Academic Publishers, 2004.

[96] S. S. Sapatnekar and R. B. Deokar. Utilizing the timing skew equivalence in a practical algorithm for retiming large circuits. *IEEE Transactions on Computer-Aided Design*, 15(10):1237–1248, Oct. 1996.

[97] N. V. Shenoy. Retiming: Theory and practice. *Integration, the VLSI Journal*, 22(1):1–21, 1997.

[98] J. Sifakis. Use of petri nets for performance evaluation. In *Proceedings of the Third International Symposium on Measuring, Modelling and Evaluating Computer Systems*, pages 75–93, 1977.

[99] M. Silva. Introducing petri nets. *Practice of Petri Nets in Manufacturing*, pages 1–62, 1993.

[100] K. Singh and A. Sangiovanni-Vincentelli. A heuristic algorithm for the fanout problem. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 357–360, June 1990.

[101] M. Singh and M. Theobald. Generalized latency-insensitive systems for single-clock and multi-clock architectures. In *Proceedings of Design, Automation and Test in Europe (DATE)*, volume 2, pages 1008 – 1013, Feb. 2004.

[102] J. Sparsø and S. Furber, editors. *Principles of asynchronous circuit design: a systems perspective.* Kluwer Academic Publishers, 2001.

[103] A. Srivastava, R. Kastner, C. Chen, and M. Sarrafzadeh. Timing driven gate duplication. *IEEE Transactions on VLSI Systems*, 12(1):42–51, Jan. 2004.

[104] S. Suhaib, D. Mathaikutty, S. Shukla, and D. Berner. Validating families of latency insensitive protocols. *High-Level Design, Validation, and Test Workshop. Tenth IEEE International*, 0:127–134, Nov. 2005.

[105] G. Thorsten. *System Design with SystemC.* Kluwer Academic Publishers, 2002.

[106] H. J. Touati and R. K. Brayton. Computing the initial states of retimed circuits. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 12(1):157–162, 1993.

[107] S. H. Unger. *Asynchronous Sequential Switching Circuits.* Wiley-Interscience, John Wiley & Sons, Inc., 1969.

[108] T. E. Williams. Performance of iterative computation in self-timed rings. *Journal of VLSI Signal Processing*, 7(1/2):17–31, Feb. 1994.

[109] R. W. Wolff. *Stochastic modeling and the theory of queues.* Prentice Hall, 1989.