DISSERTATION

Analysis of On-Chip Fault-Tolerant Distributed Algorithms

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften

unter der Leitung von

Univ.Prof. Dipl.-Ing. Dr. Ulrich Schmid
Institut für Technische Informatik
Embedded Computing Systems Group
Technische Universität Wien

eingereicht an der

Technischen Universität Wien
Fakultät für Informatik

von

Dipl.-Ing. Matthias Függer

fuegger@ecs.tuwien.ac.at

Matrikelnummer: 0100031
Friedenshöhegasse 36
1130 Wien, Österreich

Wien, Juni 2010

Kurzfassung

Die formale Spezifikation und Analyse von hochintegrierten Schaltungen (VLSI Circuits) ist für die Verwendung in hochzuverlässigen Anwendungen unerlässlich. Zwei Trends im VLSI Design sprechen für einen Modellierungsansatz analog zu dem verteilter Systeme: (i) merkliche Kommunikationszeiten zwischen Schaltungskomponenten und (ii) nicht vernachlässigbare Fehlerraten, verursacht durch Abnutzungseffekte und Teilchentreffer in Schaltungen mit immer kleiner werdenden Strukturgrößen. Trotz dieser vielversprechenden Gemeinsamkeiten können VLSI Circuits mit Modellen klassischer verteilter Algorithmen nur umständlich oder gar nicht spezifiziert und analysiert werden: Im Gegensatz zu klassischen verteilten Systemen, in denen Berechnungen von Prozessen in diskreten Schritten ausgeführt werden, benötigt die Modellierung von VLSI Circuits ein kontinuierliches Berechnungsmodell. Ein on-chip Algorithmus wird in diesem Kontext zu einer Menge von Schaltungskomponenten, die auf kontinuierlichen Berechnungsströmen operieren und via kontinuierlicher Nachrichtenströme kommunizieren.

In der Dissertation wird ein Spezifikations- und Analyse-Framework vorgestellt, welches auf die Besonderheiten von fehlertoleranten on-chip Algorithmen abgestimmt ist. Dieses basiert auf einer dreifachen Darstellung des Verhaltens der Komponenten über die Zeit. Die elementarste Darstellung ist ein “signal”, eine Folge von Ereignissen. Darauf folgt die abstraktere Repräsentation des “status”, der mehrere ähnliche signals umfasst und schließlich die “counting function”, die eine noch größere Menge von signals repräsentiert. Diese dreifache Darstellung erlaubt die Spezifikation von unterschiedlichen Schaltungskomponenten auf verschiedenen Abstraktionsniveaus.

Die Stärken des vorgestellten Frameworks werden beispielhaft an einem asynchronen on-chip Algorithmus, das ist ein Algorithmus, der nicht von einem zentralen Taktgenerator versorgt wird, illustriert. Die Erweiterung des Frameworks um eine Petrinetz-ähnliche Spezifikationssprache erlaubt es einige für das Design von asynchronen Schaltungen zentrale Komponenten kompakt zu spezifizieren. Unter diesen befindet sich auch das “General Join” Modul, welches das Zusammenführen von mehreren Datenpfaden von verschiedenen Quellen auf eine fehlertolerante Weise ermöglicht. Neben einer vollständigen Spezifikation werden für dieses Modul auch allgemeine Aussagen über das Zeitverhalten abgeleitet. Weiters wird eine Implementierung des General Join Moduls, bestehend aus einfachen Subkomponenten, präsentiert und deren Korrektheit bewiesen.

Im Gegensatz zu asynchronen werden synchrone Designs von einem zentralen Takt versorgt, welcher einen unvermeidbaren single-point of failure darstellt. Eine übliche Methode, um synchrone Designs fehlertolerant zu machen, ist die Replikation des Designs und der Taktquelle. Dies führt allerdings zum sogenannten “Tick Generation” Problem: alle einzelnen Komponenten mit fehlertoleranten, synchronisierten Taktsignalen, die nicht über

die Zeit divergieren, versorgen zu müssen. In der Dissertation wird eine Lösung mit Hilfe einer Schaltung aus interagierenden General Join Modulen präsentiert, deren zeitliches Verhalten charakterisiert und deren Korrektheit bewiesen.

Abschließend werden Grenzen einfacher Schaltungen aufgezeigt, die nur aus kombinatorischer Logik und deren Verbindungen bestehen. Es wird bewiesen, dass es keine derartige Schaltung gibt, die das "short-pulse-filter" Problem löst; ein Problem von großer Wichtigkeit für das Design von zustandsbehafteten Schaltungen, wie Arbitern, an deren Ausgängen keine beliebig kurzen, transienten Zustandswechsel erlaubt sind. Die Dissertation endet mit einem probabilistischen Ansatz zur Implementierung eines short-pulse-filters.

Abstract

For Very Large Scale Integrated (VLSI) Circuits intended to be used in highly reliable applications, formal specification and analysis is mandatory. Two trends in VLSI design favour a modeling approach analogous to that used for distributed systems: (i) noticeable communication delays between circuit components and (ii) increasing failure rates caused by wear-out and particle hits in circuits with ever decreasing feature sizes. Despite these striking similarities, specifying and analyzing circuits by means of classic distributed system models is either overly lengthy or not possible: In contrast to classic distributed systems, where computations are performed by processes in discrete steps in time, digital circuits in general adhere to a continuous computation model. An on-chip algorithm thus becomes a set of circuit components that compute on continuous streams of events and communicate by continuous streams of messages.

In this thesis a modeling and analysis framework tied to the peculiarities of fault-tolerant on-chip algorithms is presented. It is based on a three-fold representation of the behavior of a circuit's outputs over time. The most fundamental is the signal, a trace of events; next comes the more abstract status, grouping together similar signals, and finally the counting function, grouping together an even larger set of similar signals. The three-fold representation allows the specification of diverse circuit components at different levels of abstraction.

The capabilities of this framework are then illustrated by applying it to clockless on-chip algorithms, that is, circuits that are not driven by a central clock. The framework is extended by a Petri net like specification language, which is used to state pivotal circuit components for building clockless fault-tolerant on-chip algorithms. Among those is the General Join module, a module that allows to merge data provided by different sources in a fault-tolerant manner. In the thesis a complete specification is provided and generic timing properties are derived. Furthermore, an implementation of a General Join module in terms of simpler circuit components is given and proven correct.

In contrast to clockless circuits, synchronous circuits are driven by a central clock which inherently constitutes a single-point of failure. A common technique to make synchronous circuits fault-tolerant is by replication of the circuit and its clock source. Thereby, the problem arises to provide fault-tolerant, synchronized clock signals that do not diverge over time to each of the replicas. This problem is termed the "tick generation" problem. It is shown that an alternative to replicated synchronized clock sources is to let a set of General Join modules, forming an on-chip distributed algorithm, generate synchronized clock signals in the course of their interaction. A correctness proof and performance measures of this solution are derived.

Finally limitations of a restricted set of on-chip algorithms are established: It is shown that there is no circuit, only comprising combinatorial gates and wires with constant delay that solves the short-pulse-filter problem, a problem of great importance when building state-holding devices like arbiters that are not allowed to glitch at their output. The thesis concludes with a probabilistic by-pass of these limitations.

Acknowledgements

I would like to thank my adviser Ulrich Schmid. He did not only manage to create perfect conditions for scientific work in the Embedded Computing Systems Group, most notably by his engagement in qualitative lectures on distributed computing, but was always a great help through numerous discussions on general ideas and proof details. It was also him who had the very interesting idea to study fault-tolerant VLSI systems under the perspective of distributed systems, finally leading to the thesis' topic.

I further want to thank Andreas Steininger with whom discussions were always fruitful and led to many new insights. I especially appreciate his open-minded and creative way to approach VLSI systems.

Many thanks go to Gottfried Fuchs with whom I very much enjoyed working. We collaborated in the DARTS project[1] in which many of the thesis' fundamental ideas, that could be both refined and set into a more general context in the FATAL project, emerged.

I would like to thank Josef Widder for his fundamental discussions during my time at ECS and for his great PhD thesis.

I would like to thank Bernadette Charron-Bost for her insist to closely look at the very fundamentals of distributed computing, where she always discovered deep results on their nature.

Thanks, too, go to my co-supervisor Jennifer Lundelius Welch who wrote a wonderful book and with whom I recently had the pleasure to work.

For comments to improve the thesis I am grateful to Andreas Steininger, Josef Widder, Thomas Nowak, Gottfried Fuchs and Johanna Függer.

For her support in organizational issues, I would like to thank Traude Sommer.

Finally I would like to thank my family Michaela, Reinhold, Barbara and Tapas, Hanna, my grandparents and Christian as well as Dymfna, Ulrike, her family and my friends for their warm support through all the years.

Matthias Függer
Vienna, Austria, June 29, 2010

[1]The work received funding from the DARTS project of the FIT-IT program of the Austrian bm:vit (contract 809456-SCK/SAI) and from the FATAL project of the Austrian FWF Fond under project number P21694.

For Dymfna.

Contents

List of Figures

Chapter 1

Preface

THE AIM of this thesis is to approach the challenging area of on-chip fault-tolerant distributed algorithms in a way inspired by the formal analysis of classic distributed algorithms. As such, the thesis is neither the first step into this direction (for example, [17] is a special issue on hardware design) nor the last. However, the author hopes the reader will still find one or the other inspiring novel approach towards this interesting field. Classically a distributed system is a set of autonomous processes equipped with means to communicate with each other, e.g., by message passing. A distributed algorithm is an assignment of locally executed algorithms to the processes of a distributed system, where each local algorithm operates only on the process' local state (including its received messages). Interesting distributed algorithms are intended to solve a global problem, despite their lack of global knowledge. In this thesis the term "on-chip algorithm" denotes an algorithm that is intended not to be implemented in software, but rather in terms of (simple) hardware building blocks (modules) directly placed on a chip. In this thesis a distributed system is a set of modules residing on a chip that can communicate with each other by signalling, and an on-chip algorithm is an assignment of an implementation to each of the distributed system's modules. Implementations themselves comprise of modules that perform very simple computations only (here a multiplication of two 8-bit numbers is already a non-simple computation).

Due to the impossibility to cover all significant aspects in satisfactory depth, this thesis has to concentrate on a few particular issues that, however, are still representative. The following were chosen:

(i) A formal framework to model on-chip algorithms is introduced. We assume that the algorithm's hardware modules may fail during operation time due to wear-out, particle hits, etc., and behave in an arbitrary way after they have become faulty. Therefore major ingredients of the framework are the possibility to capture timing properties and fault tolerance.

(ii) A high-level Petri net like specification language is presented, together with building blocks that can be used to construct fault-tolerant on-chip algorithms.

(iii) An example non-toy algorithm is stated in terms of high-level building blocks and is proven correct; performance bounds are also derived.

(iv) The possibility of fault-containment of modules, that is, the possibility to build modules that meet their specification even if some of their inputs are driven by faulty modules, is investigated.

A brief description of the above issues follows.

Not to the least because of their great aesthetic attraction, this thesis only covers clockless algorithms, i.e., on-chip algorithms that do not make use of a central clock source. A second, more practical, reason in favor of clockless algorithms can be easily given: unlike their classical synchronous counterparts, clockless algorithms are not driven by a central clock which inherently constitutes a single point of failure. Clearly, though, synchronous algorithms can be made fault-tolerant by replicating the clocks. This, however, inevitably leads to metastability-prone[1] communication between functional units, clocked by different clocks, unless the clocks are synchronized. A method of how to obtain synchronized clocks is the topic of one of the chapters: interestingly the presented solution makes use of a clockless algorithm. Most of the ideas presented, however, can be easily applied to synchronous circuits as well.

Obviously, a thorough analysis of any system requires a formal modeling and analysis framework. It is first shown that classical modeling frameworks do not fit very well the peculiarities of on-chip algorithms: either they are not expressive enough, or a representation of important concepts in their language is lengthy. Thus a new modeling and analysis framework for on-chip algorithms is proposed, capable of expressing continuous time and fault-tolerance. This approach allows the specification of hardware components in terms of modules that communicate with each other and together form the system under consideration.

A more abstract, Petri-net-like, way of specifying clockless circuits is presented. The proposed language is then used to specify common building blocks of fault-tolerant clockless circuits. For example, a standard module used in non fault-tolerant circuit design is the Join module. It is used to merge data of multiple predecessor modules and forward it to a successor module. It does so by waiting to receive data from *all* its predecessor modules, before handing over the merged data to the successor module. Because of this "wait for all" semantics the join, however, is of limited use in fault-tolerant on-chip algorithms. Thus the set of modules is extended by a General Join module that does not wait for data from all predecessor modules. Rather, it only waits for a subset of its predecessor modules. In

[1] A hardware module that is bistable, i.e., has two stable internal states, may remain in an unstable state different from the two stable states for an unbounded time before it flips into one of its stable states. This problem is called the "metastability problem". See Section 6.2 for more details on metastability.

terms of the Petri net like specification language, this change boils down to introducing anti-tokens circulating in the Petri net.

A further point of interest is to measure the timing performance of a given algorithm. In the field of distributed computing this is classically done by giving best case and worst case bounds for the sought timing measures. Typically, those bounds are too conservative. We thus present a (simulation) relation between non fault-tolerant control structures of clockless algorithms and a well-known simple distributed algorithm, namely *Full Reversal* introduced by Gafni and Bertsekas [42], of which the detailed performance measures are known. While the technique is restricted to non fault-tolerant clockless algorithms with identical communication delays, it provides good estimates for real systems with homogeneous communication delays. An adaptation of this method to fault-tolerant algorithms and to non-homogeneous delays is planned future work.

The new framework, together with the abstract specification language, is then "tested" by applying it to a fault-tolerant tick generation algorithm: the latter comprises a set of tick generation units that locally generate ticks (0/1-transitions) in a distributed manner and establish synchronization, even in spite of a certain fraction of arbitrarily faulty tick generation units. Using the new modeling framework, the algorithm is proven correct and performance metrics are derived.

When proving a distributed system correct, one typically partitions it into isolated and autonomous parts, which may fail, and then proceeds to prove that the distributed system still solves a certain problem in spite of up to some number f of faulty parts. Such a partitioning only makes sense if a single faulty part cannot cause another part or even all other parts to fail, i.e., if *fault-containment* can be guaranteed. The thesis' final topic is a short investigation of fault-containment in terms of the new modeling framework.

1.1 Motivation

This thesis is motivated by three important aspects of on-chip algorithm design, namely fault-tolerance, the distributed system's view, and the focus on clockless circuits which are discussed in detail below.

Fault-tolerance. The International Technology Roadmap of Semiconductors [48] projects that feature sizes and operating voltages will further decrease, such that product failures and parametric shifts of a chip's components will increase. While these trends result in a higher number of permanent faulty chips at production time, decreasing the manufacturer's yield [48], system failures that occur during operation will and have already increased, too [4, 18, 19]. The latter types of failures, clearly, play a crucial role when used in high-reliability applications. The reasons for failures that occur during system operation, and by this their effects on the system, differ. A possible classification of faults in terms of persistency is given in [18, 19]: faults can be permanent, intermittent or transient.

A *permanent fault* is one that results in an irreversible malfunction of the faulty device, e.g., a transistor, logic or memory cell. A malfunctioning cell with, say, one output is not necessarily silent, i.e., produces a constant signal after it has failed. The latter type of fault is called a *stuck-at fault*. Simulations carried out by Grahsl *et al.* [43] with a C-Element (a cell widely used in asynchronous circuit designs) where transistors were replaced by permanently faulty transistors, have shown that stuck-at faults only make up a minority of possible faults of a C-Element. In [33], Fuchs *et al.* discussed alternative fault models to the stuck-at fault. Unfortunately, it turned out that only arbitrary (in terms of distributed computing, Byzantine [56]) behavior of faulty cells can be assumed with sufficiently high coverage of the fault hypothesis.

An *intermittent fault* results from a device that behaves slightly off specification, e.g., due to a parameter shift or highly resistive interconnections between cells, which may result in (data dependent) delay faults [19]. While not permanently faulty, an intermittently faulty cell produces failures, e.g., glitches[2], with an increased rate. When being replaced by a correct device, the rate decreases.

Transient faults occur because of environmental conditions, like radiation, and thus their rate cannot be decreased by replacement of cells with equivalently designed (identical design, layout, etc.) cells. The main reasons for radiation-induced faults at sea level are (i) alpha particles emitted from package impurities, as well as neutrons (the most common hadrons at sea level) colliding with (ii) ^{10}B isotopes from the B-doping of Si and with (iii) Si atoms, which both produce ionized secondary particles [4, 5]. Ionized particles, while traversing the Si material, produce e^--h^+ (electron-hole) pairs along their track. The effect an ionized particle has on the circuit hit is typically given in terms of LET (linear energy transfer), $\frac{dE}{dx}/\rho$, where E is the energy deposited in the material and ρ the material density. Given the LET of a particle, the number of induced e^--h^+ pairs generated per traversed distance can be calculated. The pairs are typically collected at reverse biased junctions of transistors, thereby generating a current, respectively, voltage swing when moving in the material [65, 93]. Note that, particle movement is both due to drifting in the electric field and diffusion, initiated by the higher e^- and h^+ concentrations near the ion track. Baumann [4] states that typically the drifting process produces a large current spike, while the diffusion process becomes dominating later on, leading to a long tail of the induced charge pulse.

The induced current then may either (i) flip a memory bit stored in a DRAM, SRAM or flip-flop (we say a soft error (SE) and more specifically, a single-event-upset (SEU) has occurred), (ii) create a glitch (we say a single-event-transient (SET) has occurred), that is, either generate a new or mask an existing voltage pulse at a logic cell's output, or (iii) be masked by electrical means. Since masked electrical pulses are not necessarily well-formed, they may be perceived differently by successor cells. In terms of a purely digital system model, as our framework, the above failures map to spurious and missing 0-to-1 and 1-to-0 transitions that are received by a (possibly proper) subset of successor modules, leading to arbitrary and asymmetric failures.

[2] A cell's output is said to glitch, if it takes on an unintended, wrong state for a short time.

DRAM cells used to be more susceptible to radiation-induced faults compared to SRAM cells because of their large surface and dynamic update. However, this is not any longer the case [4]. Two trends can be observed: due to decreasing feature sizes, SRAM cells became smaller and thus less likely to be hit by a particle. However, for the same reason and because of the reduced operating voltage, the critical charge necessary to flip a bit became smaller [4, 20]. While mitigation techniques allowed to reduce the resulting soft error rate (SER) of a *single bit cell*, the *system* SER is nearly constant for DRAM and has increased for SRAM [4], because of the increasing system size, and for combinatorial logic [4, 18–20], due to dominating interconnects with non negligible and varying RC, reduced voltage swing and smaller drivers, which make electrical masking less likely.

Summarizing, according to Constantinescu [18–20], the rate of transient and intermittent system faults will further increase and affect combinatorial logic to a larger extent. Mitigation techniques are numerous and reach from material level to the system level.

Material level techniques comprise the use of different or highly pure materials. For example, it was observed that alpha particles result from radioactive impurities in the device material as well as solder bumps [5]. Baumann *et al.* [5] have identified ^{10}B isotopes, used in doping and to form dielectric layers, as a major contributor to the system SER and propose not to use ^{10}B. Advanced VLSI process technology like Silicon on insulator (SOI) has also been proven to effectively reduce radiation susceptibility: the idea is to have a buried insulator in the substrate that shortens the effective path an ion can generate e^--h^+ pairs along [65, 93].

Techniques on higher levels of abstraction include W/L sizing of transistors[3], where a larger W/L results in stronger drivers and thus less susceptible combinatorial logic. Zhou *et al.* [99] have proposed a model allowing fast estimation of the reliability gained by transistor sizing, enabling tradeoff-decisions between reliability, circuit size, and power consumption. Further mitigation techniques consider layout adaptations [86] and different cell-designs.

Techniques at even higher level of abstraction consider error detection and correction codes which are widely used to protect highly-regular circuit designs, like memory, against transient faults. Another example is the Dual Interlocked Storage Cell (DICE) [11] which is a storage cell made tolerant to SEUs by (i) replicating the internal inverter storage loop of a memory cell and (ii) interlocking both storage loops in a way such that transient upsets of one loop are corrected by the second loop within a short time.

Finally, *system level methods* use replication of functional units in space as well as calculations redundantly performed in time, and combinations of both. Clearly, redundancy in time is restricted to transiently faulty units, while redundancy in space allows to tolerate permanent, intermittent and transient faults, provided that a single fault occurring at a functional unit does not lead to faulty behavior of other replicated functional units, i.e., fault-containment is guaranteed. While replication of functional units within a single chip is expensive, it is expected to gain in importance because of the increase of system

[3] W denotes the width and L the length of the transistor.

SER [20] and the decrease in cost per transistor. Further, system-level techniques may use standard fabrication processes and cells and therefore, too, are attractive for industry.

Distributed Systems' View. Due to the replication of functional units, forming a system-on-chip (SoC), and the need to keep their state consistent, problems arise that are analogous to the problems encountered in the context of classic replicated state machines. Related issues have been studied for decades by the distributed algorithms community, both for non-fault prone systems [54] and systems where functional units may fail [30, 56, 83]. Indeed an SoC comprising a system of functional units that (i) may fail and (ii) communicate by signaling each other with non-negligible message delays, deceptively looks like a classical distributed system, formed by autonomous processes that communicate by message passing. It is thus promising to apply approaches used in the context of distributed computing to SoCs. A recent investigation of whether such approaches exist and can be successfully used to solve problems in fault-tolerant chip design, has been undertaken in a Dagstuhl seminary, organized by Charron-Bost, Dolev, Ebergen and Schmid [14].

Despite all similarities classical distributed systems share with systems-on-chip, there are major differences: the most striking ones being (i) the absence of a discrete zero-time step abstraction that performs complicated computational tasks in SoCs and (ii) the high degree of heterogeneity in SoCs. Chapter 2 is concerned with discussing the implications of (i). Here we will concentrate on (ii): most of the theoretical work assumes classical distributed systems to be highly regular with respect to the local algorithms run on the processes (the only difference of any two local algorithms run on distinct processes are often just their process identifiers) and the communication structure. While it could be argued that distributed algorithms for SoCs, too, comprise only a few distinct algorithmic components (consider, for example, a circuit that is only built from NAND, NOR and NOT gates), the heterogeneity of the communication structure, typically, does not match classically assumed homogeneous communication structures. Although there exists a large body of work on classical distributed algorithms that may be run in systems with irregular communication graphs, links in the communication graph are typically assumed to be equally constrained. For example, in case of synchronous systems, the lower and upper bounds of all timely links are identical. An interesting exception is the work by Halpern, Megiddo and Munshi [46] who consider the clock synchronization problem in distributed message passing systems with arbitrary communication graphs and not necessarily equal delay uncertainties on its links. Contrary, in a circuit of NAND, NOR and NOT gates interconnections are not necessarily equally long; rather, their actual delays depend on the outcome of the layout process.

We will next see why global, homogeneous constraints on the interconnect often are too restrictive: typically, a circuit $\mathcal{C}$ is intended to solve a problem, e.g., the adder problem, where inputs to the circuit are two binary numbers (in a certain range) and the output is the sum of the inputs. One would consider $\mathcal{C}$ a correct solution for the adder problem, if the output really is the sum for each possible input number pair, possibly subject to some additional properties on the inputs and output. Figure 1.1 shows an example solution for

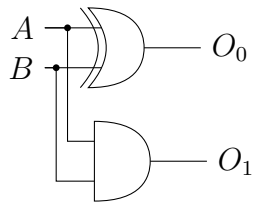

Figure 1.1: Adder Circuit

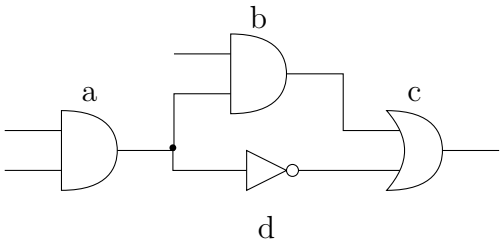

Figure 1.2: Relative Delay Constraints

the adder problem, where inputs A and B are restricted to 1 bit numbers and the output comprises two ports O_0 and O_1, whose state is the binary sum of the input's states after some time. Assume we further demand that when the inputs' states change at exactly the same time, the outputs' states must not glitch, i.e., produce arbitrarily short pulses. Then we must require that the delay from port A to the input port of the XOR gate and the delay from port B to the other input port of the XOR gate are equally long. Otherwise, O_0 might glitch[4]. Notice that not all the interconnect delays in $\mathcal{C}$ affect correctness: while some delays must adhere to restrictions, like relative delay constraints (e.g., say, in the circuit of Figure 1.2 the delay along the path from gate a via b to c must be at least twice the delay along the path from a via d to c) and absolute delay bounds, others can be chosen arbitrarily, without violating $\mathcal{C}$'s correctness and having only effects on the performance of $\mathcal{C}$. The approach taken in this thesis is to assume unknown, but finite delay bounds on sets of links, i.e., links that can be assumed to have common delay bounds, but allow each link to have an arbitrarily and typically varying delay within the bounds. The proofs then show which constraints have to hold for these bounds in order for a circuit to correctly solve a problem. Note that, the less timing constraints exist and the easier it is to fulfill those, the more robust the circuit will be.

Clockless Circuits. Consider a distributed system that communicates by message passing. In the context of distributed computing the term "asynchronous" is used to describe systems where no restrictions on the end-to-end message delays (from message send to message receive) exist and the relative speed by which processes perform computing steps is unbounded. That is, nothing but the finiteness is known on message and computing delays. By contrast, in VLSI the term "asynchronous" describes systems that are not clocked by a single oscillator. It does, however, not specify the degree, to which delay bounds are restricted: there, too, exist design styles for asynchronous hardware, in particular, *delay insensitive* (DI) circuits [16,91], where no assumption beyond the finiteness has to be put on the message (signal) delays between the components. Automatic compilation techniques that map languages to distributed SoCs were proposed, e.g., by Martin [62] and Ebergen [26]. Interestingly, both had to introduce additional constraints on some com-

[4]For example consider the inputs A, B to change from state $0, 0$ to $1, 1$.

munication forks[5], namely to have approximately the same delay on all the fork's paths, thereby forming an *isochronic fork*. A circuit, with some of its forks being isochronic is called *quasi delay insensitive* (QDI), whereas a circuit with all of its forks being isochronic is termed *speed-independent*. In [63], Martin proved that the set of problems solvable with circuits without any isochronic fork is very restricted. In that sense, QDI circuits make use of inhomogeneous timing assumptions. In this thesis, we will, however, only partially make use of the QDI design style: to prove complicated algorithms correct, like the one presented in Chapter 5, even more general timing assumptions than the isochronic fork have to be considered, allowing to express delay constraints on arbitrary circuit paths and not just forks. In terms of general constraints, often looser constraints can be found, that are easier to fulfill than isochronic fork assumptions.

Whenever confusion may arise which of the two meanings of "asynchronous" we refer to, we will replace the VLSI related term by the more descriptive "clockless".

It remains to motivate the use of clockless circuits in the context of fault-tolerant on-chip algorithms. From the above discussion it is already clear that clockless circuits allow for heterogeneous use of relative as well as absolute timing constraints, whereas synchronous circuits demand absolute timing constraints (introduced by the fixed clock period) on all its paths of combinatorial logic between two successive flip-flops. Therefore a larger robustness of asynchronous circuits to (slightly) varying communication delays is expected: in [18], Constantinescu performed a case study with commercial of-the-shelf (COTS) components under harsh environmental conditions (operating voltage and temperature), where it was shown that most failures were due to setup/hold-violations, i.e., violations of the absolute timing constraints of the combinatorial logic. The violations even lead to multi-bit errors and most of the time occurred in bursts.

There is a further concern about the appropriateness of synchronous systems for fault-tolerant on-chip algorithms: the clock together with the clock distribution network makes up a significant part of the chip and clearly is a single point of failure. Seifert *et al.* [84] have found the impact of radiation hits on the clock distribution network to be significant: simulations of particle-hits on a synchronous flip-flop-based design showed that bit failures in flip-flops due to current pulses in the clocking network made up 20% of the total system soft-errors.

Despite all the reasons given in favor of clockless circuits for fault-tolerant on-chip algorithms, some changes to classical clockless algorithms have to be made: ordering of computational events, respectively signal transitions, is achieved by two means in clockless algorithms: (i) explicit handshaking by acknowledging and (ii) implicit handshaking by timing constrains between two communicating components. Clearly, in case of fault-tolerance explicit handshaking cannot be used alone: consider a network of components, where each component receives data from its predecessor components, processes it and hands it over to all its successor components. Here, explicit handshaking can be used as a communication scheme between the components. However, as soon as one allows components to become faulty, a correct component that simply waits for all its predecessor and

[5] A component, providing data to two or more successor components does this by a communication fork.

successor components might stop making progress as soon as one of these is faulty. To overcome this limitation, implicit handshaking can be used, however, introducing additional timing constraints.

1.2 Structure of the thesis

In Chapter 2 the major difference between classical computation and a computation model that is tied to on-chip algorithms are discussed. It is followed by the introduction of a new formal framework in Chapter 3. The framework is then used to describe important components for building the control structures of fault-tolerant clockless algorithms in Chapter 4. The central component presented is the General Join Module. In Chapter 5 this component is used to solve the fault-tolerant tick generation problem. The solution provides the chip designer with a generic solution to replace the central clock signal by a set of synchronized, local clock signals, generated by a fault-tolerant distributed on-chip algorithm. Chapter 6 finally discusses the fault-containment assumptions made throughout the previous chapters from a theoretical point of view: it is investigated whether the modules previously used can be implemented with boolean gates and channels alone. Each chapter is ended by a "related work" section which both discusses existing related work as well as lists further, detailed, work by the author.

Chapter 2

Introduction

In THE previous chapter it has been motivated that the formal analysis of fault-tolerant clockless on-chip algorithms is of its own interest and cannot (very well) be performed by standard formal means to analyze classical distributed algorithms. The most significant difference between hardware-based and classical algorithms is the missing abstraction of discrete computing steps in the former. To shed light on this major difference, we will compare the causal structure of a classical asynchronous computation model and an asynchronous hardware-based computation model.

2.1 Classical Asynchronous Computations

Consider a computation model analogous to the one proposed by Fischer *et al.* in [30]: a *distributed system* comprises a finite set P of processes, with $|P| \geqslant 2$, where processes can communicate by message passing and each can run a local algorithm, resulting in a series of (receive-compute-send) steps happening at each $p \in P$. Each process p in P is equipped with a local memory (that holds p's state). A *distributed algorithm* is an assignment of local algorithms to processes in P. A *configuration* is a tuple of all the processes' state as well as the message layer state (containing the messages in transit). A *step* is a tuple comprising (i) the process $p \in P$ that makes the step, (ii) the messages received in the step and (iii) the messages sent in the step. A step is either applicable to a configuration or not. For example, a step involving the reception of message m is only applicable in a configuration where m is in transit. An application of a step to a configuration leads to a new configuration and the possible state transitions are specified by the distributed algorithm's transition relation ϕ: let Σ be the set of steps and Γ the set of configurations. Further, let relation $\phi \subseteq \Gamma \times \Sigma \times \Gamma$, where $\langle C, s, C' \rangle \in \phi$ iff s is applicable to configuration C and results in configuration C'. We further restrict ourselves to deterministic algorithms, i.e., ϕ is right-unique[1]. Without formally stating the details, transition relation ϕ is the

[1] That is for any two $\langle C, s, C'' \rangle$ and $\langle C, s, C''' \rangle$ in ϕ, $C'' = C'''$.

union of local transition relations of every process' algorithm: the application of a step of process $p \in P$ comprises (i) either p receiving a message which is in transit to p or p receiving a special $\emptyset$ message, and (ii) based on p's current local state and the message received: p changing its state according to its local algorithm and sending messages to a (possibly empty) subset of processes in P. In case $p \in P$ sends some message m (m is unique in each execution) in step s and $q \in P$ receives m in step s', s will be called a *send step* (of m) and s' the *corresponding receive step*. An *execution* of a distributed algorithm is an alternating sequence of *configurations* and steps made by processes in P, generated by the successive application of applicable steps to an initial configuration. For analysis purpose the existence of global Newtonian time is assumed and steps of an execution occur at instants in Newtonian time from $\mathbb{R}_0^+$.

The only restriction on the message communication is that a process performing infinitely many (receive-compute-send) steps during an execution will eventually receive any message sent to it. Since we do not assume any failures here, we demand that every process in P makes an infinite number of steps during an execution. Thus, every message sent is eventually received.

We will next recapitulate the means by which the causal structure of asynchronous distributed computations can be captured: for this purpose let us define an *event* of process $p \in P$ in an execution. Given an execution ϵ, we define the *set of events of* ϵ, denoted by $E(\epsilon)$ or simply by E, as the set of $\langle t, s\rangle$, such that step s occurs in ϵ at time t. For an *event* $e = \langle t, s\rangle \in E$, we denote its time of occurrence by $T(e) = t$.

For execution ϵ an event $e = \langle t, s\rangle \in E(\epsilon)$ is a *send event* (of message m) iff s is a send step (of m) that occurs at time t in ϵ. Event $e_r = \langle t_r, s_r\rangle \in E(\epsilon)$ is the *corresponding receive event* (of message m) iff s is a receive step $s_r \in \Sigma$ where m is received that occurs at time t_r in ϵ.

In his seminal work [54], Lamport introduced a general formalism to express the potential causal structure of some execution ϵ, which can be also applied to the computation model described above. The key components are events e from a set $E = E(\epsilon)$ and a binary relation $\longrightarrow$ on $E^2 = E \times E$, called "happened before" that is defined by:

Definition 2.1. $\longrightarrow$ *is the smallest (with respect to set inclusion) relation for which:*

(i) *If a and b are events of process p and a directly happens before b, then $a \longrightarrow b$.*

(ii) *If a is a send event and b is the corresponding receive event, then $a \longrightarrow b$.*

(iii) *If $a \longrightarrow b$ and $b \longrightarrow c$, then $a \longrightarrow c$ (transitivity).* □

The intuitive meaning of "$\longrightarrow$" is that if $a \longrightarrow b$, event a can potentially influence event b via a chain of local computations and messages sent between processes. If $\neg(a \longrightarrow b)$, there does not exist such a chain and a cannot influence b. Two events a and b are said to be concurrent iff both $\neg(a \longrightarrow b)$ and $\neg(b \longrightarrow a)$.

Mattern [64], too, investigated the potential causal structure of executions, with a slightly different notation: the direct successor relation on a process is denoted by $\prec_l$, i.e.,

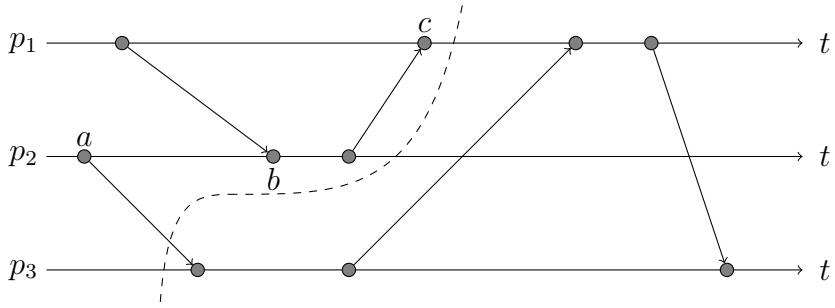

Figure 2.1: Classical asynchronous execution

$a \prec_l b$ iff both events a and b occur at the same process and a is a direct successor of b and $a \prec b$ is analogously defined to $a \longrightarrow b$. To avoid confusion of happened before with logical implication we will stick to Mattern's notation throughout the thesis. In [64] Mattern introduced consistent cuts of executions. A *consistent cut* S for a given execution with events E is a set $S \subseteq E$ that is left-closed with respect to $\prec$: for all e in S and e' in E, $e' \prec e \Rightarrow e' \in S$. A special consistent cut is the *causal past* of event e in E, defined by the set $\{e' \in E \mid e' \prec e\}$. The importance of the causal past of e is that in an asynchronous distributed system analogous to [30] (with a deterministic algorithm run on it), a process' local state after an event e is fully determined by the causal past of e, with the $\prec$ relation on it, together with the initial states of the processes. It is this latter observation that will play an important role in Chapter 6 for proving impossibility results.

Example 2.1. *An example execution of three processes is depicted in Figure 2.1. The asynchronous execution has been embedded in global Newtonian real-time which, however, is not visible to the processes themselves. Events of a process happening at a certain time are depicted as dots on the process' timeline. A message send event e with the corresponding receptive event e' is shown by an arrow from e to e'. Here $a \prec_l b$ and $a \prec c$ holds. It is important to note that the local state of process p_1 after application of c only depends on the causal past of c and the initial local states of processes p_1 and p_2. In Figure 2.1 the causal past of c is the set of events to the left of the dashed boundary line.*

We will next discuss asynchronous[2] executions in hardware and relate these to classical asynchronous computations.

2.2 Asynchronous Computations in Hardware

The most significant difference to classical asynchronous computations is the absence of discrete computing steps which are replaced by a continuous stream of computation. We discuss this difference by means of the example of the digital AND gate depicted in Figure 2.2. The gate has one output port c and two input ports that are connected to ports a

[2]Here "asynchronous" refers to the distributed computing term, i.e., unbounded but finite delays.

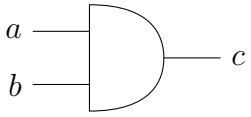

Figure 2.2: AND gate

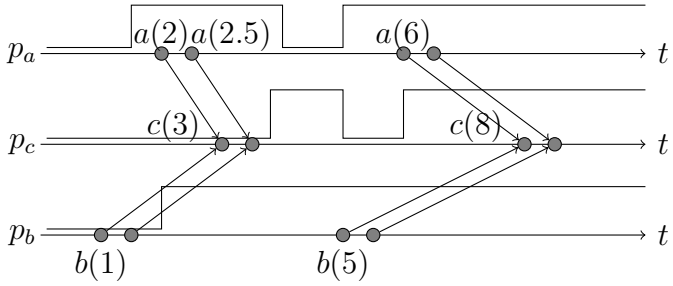

Figure 2.3: Asynchronous execution in hardware

and b via wires with varying, unbounded delay. The behavior of ports is described by signals, where a signal s is assumed to be binary; thereby $s : \mathbb{R}_0^+ \to \{0, 1\}$ is a function, where $s(t)$ is the state of signal s at time t. We can model this hardware setup as a distributed system comprising three processes p_a, p_b and p_c. Process p_a and process p_b continuously[3] send messages to process p_c, where the content of the message is the current binary state of signal a and b, respectively. Let us further assume that after some initial phase ending at time $T \geqslant 0$, at any point $t \geqslant T$ in time, process p_c receives two messages, one from p_a and one from p_b. This is true if a wire is modeled by a FIFO channel with a delay that is a continuous function in time, which is a realistic assumption: wire delays change because of chip temperature, voltage etc. It is adequate to assume that these changes are continuous over time. The FIFO property for channels results from the fact that binary signal transitions from 0 to 1 and vice versa do not overtake each other on a wire, if properly encoded (like 0 corresponds to low voltage and 1 to high voltage). If p_c receives two messages at time $t \geqslant 0$, it can easily determine $c(t)$, by computing the logical AND of both message contents received at time t. In case it does not receive a message from either p_a or p_b, we may model this by the reception of a special $\emptyset$ message from the respective sender process and define default values for the message $\emptyset$, depending on whether it comes from p_a or p_b. The latter models the initial state of the wire: e.g., to model the fact that the wire from p_a to p_c is initialized to 0 (respectively 1), the default value $\emptyset$ received from p_a is 0 (respectively 1).

Figure 2.3 shows an execution of the distributed system just described. The status of the three signals a, b and c is depicted above the timeline of the respective process. There have only been drawn 8 out of the continuum many messages and 12 out of the continuum many steps performed by the processes.[4] For example, process p_c at time 3 receives a message from p_a with content $a(2) = 1$, which is the value of signal a when p_a sent the message, and a message with content $b(1) = 0$ from p_b. p_c then computes $c(3) = a(2) \wedge b(1) = 0$. Similarly, $c(8)$ is computed by $c(8) = a(6) \wedge b(5) = 1$.

Before we investigate the causal structure of asynchronous hardware computations by

[3]Here, the term "continuously" does not mean forever periodically, but at every time t in $\mathbb{R}_0^+$. Thus an uncountable number of messages are sent.

[4]The messages drawn are those p_c received from p_a and p_b in the 4 steps depicted.

means analogous to [64], we have to take a look at the formal definition of an execution and a process making a step. Clearly a modeling approach as described in Section 2.1 is not possible here, since there is no immediately preceding step in a continuum of steps: a step thus cannot take a pre-state and transform it into a post-state. Further the concept of a transition function must be replaced. We thus define: A *distributed system* is a set P of processes. *Steps* are from a set Σ and are tuples that comprise (i) the process $p \in P$ that makes the step, (ii) the messages received from all other processes (possibly $\emptyset$ messages) and (iii) the messages sent to all other processes (possibly $\emptyset$ messages). Whether a process p's step is *applicable* at some time $t \in \mathbb{R}_0^+$ and the effect of the application of the step on the process' output state is specified by p's *output relation* ϕ_p: let $\mathcal{H}_p$ be the set of *histories* of p. Then $\phi_p \subseteq \mathcal{H}_p \times \Sigma \times \{0,1\}$ is a right-unique relation, $\langle H, s, x\rangle \in \phi_p$ is applicable at time t iff s happens on p, *p's history at time t*, $H_p(t)$ (defined later on), is equal to H and the result of applying s is that p's output state at time t is set to x. An *execution* $\epsilon : P \times \mathbb{R}_0^+ \to \Sigma \times \{0,1\}$ is a function that maps each process $p \in P$ and time $t \in \mathbb{R}_0^+$ to the tuple $\langle s, x\rangle$, where s is the step process p makes at time t and x is the resulting output state of process p at time t. It remains to specify $H_p(t)$, p's history at time t. There are two natural ways to do this, both of which will be needed throughout the thesis: (a) processes with empty history and (b) processes with full history. We will discuss one after the other. Before doing so, we adapt Definition 2.1 not to make use of the term "directly happens before". Consider an execution ϵ. The set of *events* of ε, denoted by $E = E(\epsilon)$ is analogously defined as for the classic asynchronous system from Section 2.1, namely, the set of all tuples $\langle t, s\rangle$, such that s occurs in ϵ at time t. Relation $\prec_l$ on E^2 is defined in a way specific to the two cases (a) and (b) and therefore detailed later on. We define relation $\prec$ on E^2 by:

Definition 2.2. *$\prec$ is the smallest (with respect to set inclusion) transitive relation for which:*

(i) *If a and b are events of process p and $a \prec_l b$, then $a \prec b$.*

(ii) *If a is a send event and b is the corresponding receive event, then $a \prec b$.* □

ad (a) In case of processes with empty history we define: for all processes $p \in P$ and times $t \in \mathbb{R}_0^+$, $H_p(t) = \emptyset$. By the definition of ϕ_p, a process thus becomes a device that sets its output state at time $t \in \mathbb{R}_0^+$ only according to the messages it receives at time t. If a process wants its current output state to influence its future output state it has to send a local message to itself which then influences its output state at the time of reception.

To model the absence of causal influence of locally performed steps on their successor states, we simply define $\prec_l = \emptyset$. The definitions of consistent cuts and causal past can be taken over from Section 2.1 without a change. Again, it holds that the causal past of event e at process p happening at time $t \geqslant 0$ determines p's output state at time t: the output state at time t is a function of the causal past of e, together with the $\prec$ relation, alone. In case of the example execution depicted in Figure 2.3, the casual past of the event happening at p_c at time 8 is the set comprising the events happening at p_a and p_b at times 6 and 5, respectively.

This approach is useful when modeling simple gates, like the AND gate. However, it has its limitations when describing more complex systems: first, there are systems whose specification in terms of this system model is lengthy; and secondly, in Chapter 6, it is shown that there exist important hardware systems that cannot be specified in terms of this system model.

ad (b) In case of processes with full history we define: for all processes $p \in P$ and times $t \in \mathbb{R}_0^+$, $H_p(t)$ is the set of $s \in \Sigma$, such that s happens at process p at least at time t, equipped with the happened before relation, i.e., tuples $\langle s, s' \rangle \in \Sigma^2$, for which holds that s and s' happen at process p, s happens before s' and s' occurs at latest at time t. To model the fact that each event is possibly causally influenced by all local predecessor events, we define $\prec_l$ to reflect exactly this dependence, by letting $\prec_l$ be the set of $\langle e, e' \rangle \in E^2$, where $T(e) < T(e')$. An example specification making use of a full history ϕ_p follows:

Example 2.2. *Consider the gate depicted in Figure 2.2. We will specify the behavior of the AND gate in a different way, making it act as an AND for 0-to-1 transitions:* p_c *sets its state to* 1 *in the event* e *happening at time* $t \geqslant 0$*, iff either (i) it has received a message with value* 1 *from both* p_a *and* p_b *at time* t*, or (ii) it has already set its state to* 1 *in an event locally happening before* e*. Otherwise, it sets it state to* 0*. Note that, as a result of (ii),* p_c *thus remains in state* 1 *once it has changed to* 1*.*

Again the definitions of consistent cuts and causal past can be taken over from [64] without any changes. In case of Example 2.2 and the execution depicted in Figure 2.3, the causal past of p_c's event at time 8 is the set comprising all events at p_c with time stamp less than 8, and all events at p_a, as well as p_b, with timestamps not greater than 6 and not greater than 5, respectively.

Specifying distributed systems with the latter kind of transition function often turns out to be more compact. In Chapter 6, we will show that there exist systems that can be specified by means of transition functions as defined in (b), but not by means of transition functions as specified in (a).

Unfortunately, there are problems that are neither expressible solely in terms of (a) nor in terms of (b), but rather by general constraints that a correct execution must fulfill. Here, the concepts of [64] cannot be applied. Since we do not restrict our attention to purely asynchronous computations, we will make also use of such augmented specifications in the thesis. Examples of the latter kind are systems with time constraints that are expressible by processes with a timed history: in contrast to (b) where $H_p(t)$ only includes an ordering on steps (by the happened before relation) we include timing information in $H_p(t)$: for this purpose let $H_p(t)$ be the set of events that happened at p at latest at time t. Note that some systems (e.g., non-causal systems) require even more general constraints. After this brief discussion of specifications of continuous executions, a detailed formal framework is presented in the following chapter.

Chapter 3

Modeling and Analysis Framework

THE KEY differences between classical and hardware based asynchronous computation models were discussed in the previous chapter. It has been shown that these differences lead to distinct forms of causal structures inside an execution. It is thus evident, that despite the many commonalities classical distributed systems have with hardware systems, a different formalism must be used to specify, analyze, and prove correct hardware systems. In this chapter we provide such a formal modeling and analysis framework, which is adapted to the peculiarities of computations in hardware and amenable to mathematical correctness proofs as well as worst-case and average case performance analysis. To handle the design complexity challenge, which arises when analyzing non-toy hardware systems specified at a very low level of abstraction, it also supports hierarchical modeling: The framework is based on *modules* that possess input and output *ports*, which themselves are modeled via binary signals whose values evolve over time. A module's behavior specifies how the input and output signals are related. There are two different techniques to specify a module's behavior analogous to the two classical VLSI specification types: either directly (termed "behavioral specification" in VLSI designs), or via a composition of sub-modules whose input/output ports are connected in an a priori defined way (termed "structural specification" in VLSI designs). The former kind of module is named *basic module*, while the latter kind of module is called *compound module*. Before we proceed to the definition of the basic concepts the framework relies on, namely *signals* and *executions*, we introduce some notation used throughout the thesis: both $\langle a, b \rangle$ and (a, b) denote a tuple with components a and b. The different styles of brackets are only used for better readability. To denote component x from tuple $y = \langle \dots, x, \dots \rangle$, we may write $y.x$. To specify that a lemma (respectively theorem) holds if a constraint C holds, we write "Lemma (C)" (respectively "Theorem (C)") when stating the lemma (respectively theorem). An interval $[\alpha^-, \alpha^+]$, with $\alpha^-, \alpha^+ \in \mathbb{R}$ is denoted by $\alpha^\pm$. For convenience we even allow the use of more complicated terms to specify intervals, e.g., $\alpha^\pm + \beta^\pm$ is short hand for $[\alpha^- + \beta^-, \alpha^+ + \beta^+]$. We denote the set of Boolean values by $\mathbb{B} = \{0, 1\}$. We further assume Newtonian real-time for analysis purpose and let the domain of time $\mathcal{T} = \mathbb{R}_0^+$ be the set of non-negative

reals, that is, we assume that the system *starts booting* at time $t = 0$. For some systems it is necessary to include the possibility of different *booting completion* times for different modules in the formal system model. Note that this is not prevented in the presented framework.

3.1 Signals

Signals are the key elements for specifying systems. Although signals are abstract mathematical concepts, they can be thought of as the behavior of measurement points in a digital circuit over Newtonian time. While signals are event traces, that is, sets of time-value tuples, more abstract representations can be given for a signal: (i) a status and (ii) a counting function. At first sight it might seem superfluous to represent a single signal by three different means, since a formal framework typically intends to unify concepts. During the analysis of more complex hardware systems [23, 41], however, the three-fold representation turned out to be of practical importance when writing specifications and proofs. While combinatorial logic like AND gates can be compactly specified in terms of how the input's events or status is mapped to the output's status, components from asynchronous circuit theory, like the Muller C-Element [72, 92] (later discussed in this chapter) can be easily specified in terms of the input's status and the output's events. Counting functions finally turn out to be of great help when specifying components with queueing effects, like micropipelines [92] (which will be extensively used in Chapter 4). On the other hand, trying to specify a queueing system via signal states or combinatorial logic via the event representations results in long behavioral definitions.

Certainly, a three-fold representation of signals, would be of no use unless different representations could be used interchangeably as appropriate, e.g., to specify different components inside a single system. We will thus show under what assumptions the representations can be converted into each other and how this can be done. In fact, conversion between different representations of a signal is not possible for arbitrary signals. While all signals are event traces by definition, a status representation can only be assigned to signals with certain properties. Finally, counting functions can be assigned only to a proper subset of those signals that have a status representation. This leads to a hierarchical ordering of the representation possibilities with respect to their expressiveness.

Since we focus on purely digital systems, except for Chapter 6, we consider signals which can take on Boolean values only. We thus restrict a signal's value to be from $\mathbb{B}$ only. We now present the formal definitions.

Event trace. A *signal* S is an *event trace*, i.e., a relation

$$S \subseteq \mathcal{T} \times \mathbb{B}.$$

We say that signal S *has value* v *at time* t iff $\langle t, v\rangle \in S$. We require non-simultaneity of contradicting events on a single signal by

$$\forall t \in \mathcal{T} : \ (\langle t, v\rangle \in S \wedge \langle t, v'\rangle \in S) \ \Rightarrow \ v = v', \tag{3.1}$$

and demand the existence of an initial event by

$$\exists I \in \mathbb{B} : \ (0, I) \in S, \tag{3.2}$$

stating that S must have some initial value I at time 0.

Example 3.1. *An example signal* S *is* $S = \{\langle 0, 0\rangle, \langle 5, 1\rangle, \langle 7, 0\rangle, \langle 8, 0\rangle\}$. *Here* S *initially has value* 0*, then at time* 5 *takes on value* 1 *and finally at both time* 7 *and* 8 *takes on value* 0 *again.*

From the above definitions and Example 3.1 it becomes evident that event traces can involve idempotent events (i.e., events with the same value and no event with a different value in between) and do not need to be dense in time $\mathcal{T}$.

We further define the prefix $\text{pre}(S, t)$ and suffix $\text{suff}(S, t)$ of signal S at time $t \in \mathcal{T}$ by

$$\text{pre}(S, t) := \{\langle t', v\rangle \in S \mid t' \leqslant t\}$$
$$\text{suff}(S, t) := \{\langle t', v\rangle \in S \mid t' \geqslant t\}$$

Clearly for all $t \in \mathcal{T}$, $S = \text{pre}(S, t) \cup \text{suff}(S, t)$.

Typically, when specifying and analyzing a signal, one is interested in the "last event" that happened until (and including) some time $t \in \mathcal{T}$. As has been motivated in Chapter 2, terms like "direct predecessor" and "last event" generally are not well defined on (potentially) continuous computations. Indeed, this is reflected in the definition of an event trace which is only restricted to adhere to (3.1) and (3.2). Consider for example the event trace

$$S = \{(t', 0) \mid t' < 2\}, \tag{3.3}$$

for which no last event of $\text{pre}(S, 2)$ is well-defined. Note that this is in contrast to classical distributed "event-by-event" computations, where for any process and time $t \in \mathcal{T}$ a last event locally executed at time not greater than t exists. Although we cannot speak of a "last event" before time t, it is sometimes possible to speak of the "last value" of an event trace before time t. We thus specify

$$\begin{aligned}&\text{last-val}(S, t) = v' \ \Leftrightarrow \\ &\exists (t', v') \in \text{pre}(S, t) : \forall (t'', v'') \in \text{pre}(S, t) : \ (t'' \geqslant t') \Rightarrow (v'' = v').\end{aligned} \tag{3.4}$$

The idea behind (3.4) is that a signal has a last value of v' at time t iff an event with value v' occurred until and including time t and no other-valued (non-idempotent) event occurred between this event and time t. For (3.3), we thus obtain $\text{last-val}(S, 2) = 0$. The question arises, whether the last value is always well-defined, that is: does either $\text{last-val}(S, t) = 0$ or $\text{last-val}(S, t) = 1$ hold for all $t \in \mathcal{T}$. The answer is no, as can be seen from the following example.

Example 3.2. *Let signal* $S = \{\langle 0, 0\rangle\} \cup \{\langle t_i, v_i\rangle \mid i \geqslant 1, t_{i+1} - t_i = \frac{1}{i^2} \text{ and } v_i = i \bmod 2\}$.

S of Example 3.2 comprises of events $\langle 0, 0\rangle$, $\langle 1, 1\rangle$, $\langle 5/4, 0\rangle$, etc., which alternate in value, approach time $\pi^2/6$, but never reach it. Thus, last-val$(S, \pi^2/6)$ is undefined. Following [58, p. 737f] such behavior will be called Zeno-behavior (formally defined later on). We will see that if an event trace is non-Zeno, we may safely apply the last-val function to it. For this purpose, we introduce the concept of *transitions*: a set of events A is said to *have at least* $n \in \mathbb{N}$ *transitions* iff there are events $a_i \in A$ with $1 \leqslant i \leqslant n+1$, where a_i occurs before a_{i+1} and consecutive a_i have alternating values. Note that the initial event $\langle 0, I\rangle$ is not counted in n. The set A is said to *have exactly* n *transitions* iff it has at least n, but not at least $n+1$ transitions. In case there is an $n \in \mathbb{N}$ such that A has exactly n transitions, A is said to *have finitely many transitions*, otherwise it is said to *have infinitely many transitions*. We say that an event trace S is *non-Zeno until time* $t \geqslant 0$ iff for all times $t' \leqslant t$, the prefix of S at time t' has only finitely many transitions. Event trace S is *non-Zeno* iff all finite time prefixes have only finitely many transitions. Interestingly, the non-Zeno property of S implies the existence of last-val(S, t) for all $t \in \mathcal{T}$, however, the existence of last-val(S, t) for all $t \in \mathcal{T}$, does not imply the non-Zeno property as shown by Example 3.3:

Example 3.3. *Let* S *be the set of events* $\{\langle t, v\rangle \mid (t \in \mathcal{T} \cap \mathbb{Q} \Rightarrow v = 1) \wedge (t \in \mathcal{T} \cap (\mathbb{R} \setminus \mathbb{Q}) \Rightarrow v = 0)\}$. *Clearly* S *is Zeno but has a well-defined last-val*(S, t) *for all* $t \in \mathcal{T}$*, because* S *is a "complete" trace with an event happening at all* $t \in \mathcal{T}$.

Zeno-behavior is not just a tedious property that one must take care of when arguing about signals, but provides us with a means to model metastable behavior of signals. Metastability is a physical effect, discussed in Section 6.2, that can result in a physical signal taking on an indefinite (non 0/1) value for a potentially unbounded time. Since our signals may only take on values in $\mathbb{B}$, a direct modeling of metastability is not possible. Fortunately, Zeno-behavior comes as a rescue here: an undefined last-val(S, t) for t in some interval $\mathcal{I}$ can be used to model metastability of S during $\mathcal{I}$.

If, however, last-val(S, t) is well defined for all $t \in \mathcal{T}$, we may make use of a different type of a signal's representation, namely its *status*.

Status. Sometimes it is more convenient to describe the behavior of signal S by means of its status. Formally, $\widetilde{\cdot}$ is a function that maps (in a way defined later on) a signal S to status $\widetilde{S}$, where

$$\widetilde{S} : \mathcal{T} \to \mathbb{B}.$$

We say that signal S *has value* v *at time* t iff $\widetilde{S}(t) = v$.

Example 3.4. *A possible example for the status of signal* S *from Example 3.1 is*

$$\widetilde{S}(t) = \begin{cases} 0 & \text{if } 0 \leqslant t < 5, \\ 1 & \text{if } 5 \leqslant t < 7, \\ 0 & \text{otherwise.} \end{cases}$$

It remains to specify function $\widetilde{\cdot}$. For this purpose we define the following transformation from S to $\widetilde{S}$: For all $t \in \mathcal{T}$,

$$\widetilde{S}(t) = \text{last-val}(S, t), \tag{3.5}$$

and say that S *has status (representation)* $\widetilde{S}$. Thus, provided that S is non-Zeno, (3.5) is well-defined and a status representation of S exists. The set of all signals that have the same status $\widetilde{S}$ forms an equivalence class $\mathit{Event}(\widetilde{S})$. Formally

$$\mathit{Event}(\widetilde{S}) := \{S \mid \forall t \in \mathcal{T} : \text{last-val}(S, t) = \widetilde{S}(t)\}.$$

For two signals $S \in \mathit{Event}(\widetilde{S})$ and $S' \in \mathit{Event}(\widetilde{S})$, we say that S and S' are equivalent with respect to their status. When transforming back from $\widetilde{S}$ to S, one loses information about the original event trace. However, for most systems, the exact event trace is not important, as long as it has the same status. If not otherwise specified, we transform back, by returning the "complete trace" (having many idempotent events) representative of the equivalence class, i.e., the graph of $\widetilde{S}(t)$:

$$S = \text{max-event}(\widetilde{S})$$

with

$$\text{max-event}(\widetilde{S}) := \{\langle t, \widetilde{S}(t)\rangle \mid t \in \mathcal{T}\}. \tag{3.6}$$

Example 3.5. *Reconsidering Example 3.1 and Example 3.4, by* (3.5)*, we obtain that* $S \in \mathit{Event}(\widetilde{S})$*, i.e., that* S *has status* $\widetilde{S}$*.*

Counting function. Finally, some signals can be represented by the number of non-idempotent events (excluding the initial event) that occur during $(0, t]$ for time t together with the initial status $I \in \mathbb{B}$. Formally function $\#\cdot$ maps (in a way defined later on) a signal S to a counting function $\#S$, where

$$\#S : \mathcal{T} \to \mathbb{N}.$$

Formally, function $\#S$ is defined by: For all $t \in \mathcal{T}$,

$$\#S(t) = \ell \text{ if } \text{pre}(S, t) \text{ has exactly } \ell \text{ transitions.}$$

The counting function exists iff the signal is non-Zeno. It immediately follows from the definition of the counting function that it is non decreasing, i.e., for any $t_1 \leqslant t_2$, $\#S(t_1) \leqslant \#S(t_2)$. Let $\nu \to t^-$ denote the fact that ν is approaching t from the left. We say S *makes transition* k *at time* t, for $k \geqslant 0$ and t in $\mathcal{T}$, iff $\#S(t) = \lim_{\nu \to t^-} \#S(\nu) + 1 = k$.

Example 3.6. *Let* $S = \{\langle 0, 0\rangle, \langle 1, 0\rangle, \langle 2, 1\rangle, \langle 3, 1\rangle\}$*. Then* $\#S(0) = 0$*,* $\#S(1.5) = 0$*,* $\#S(2) = 1$ *and* S *makes transition* 1 *(to value* 1*) at time* 2*.*

Sometimes we will also employ generalized counting functions, which are counting functions with an arbitrary but fixed offset. A generalized counting function $\#S'$ for signal S at time $t \in \mathcal{T}$ is defined by

$$\#S'(t) := S_0 + \#S(t), \tag{3.7}$$

where $\#S$ is the counting function of S and S_0 is a constant offset.

Vice versa, given signal S with counting function $\#S$ and initial value I, we say that it has status $\widetilde{S}$, iff for all t in $\mathcal{T}$,

$$\widetilde{S}(t) = (\#S(t) + I) \bmod 2.$$

3.2 Executions

We begin with the definition of a system. A *system* is a set of *ports* $\mathcal{P}$. Ports correspond to processes in classical distributed computing and can be thought of as measurement points on a chip. An *execution (of ports $\mathcal{P}$)* is a function $e_{\mathcal{P}}$ that maps each port $p \in \mathcal{P}$ to a signal $\widehat{p}$. To avoid cluttered notation we shortly write e instead of $e_{\mathcal{P}}$, when $\mathcal{P}$ is clear from the context. Further, for $p \in \mathcal{P}$, we write $\widetilde{p}$ for the status $\widetilde{\widehat{p}}$ as well as $\#p$ for the counting function $\#\widehat{p}$. When no confusion may arise we write "port p" instead of "the signal of port p". For example, we say "port p has status $\widetilde{p}$" instead of "the signal of port p has status $\widetilde{p}$".

Example 3.7. *A (very small) example execution $e_{\mathcal{P}}$ of ports $\mathcal{P} = \{i, o, x\}$ is $e(i) = \widehat{i}, e(o) = \widehat{o}$ and $e(x) = \widehat{x}$ with $\widehat{i} = \widehat{o} = \widehat{x} = \{\langle 0, 0\rangle\}$. It consists only of the initial events happening at the three ports.*

Given two non empty subsets of ports $\mathcal{P}'$ and $\mathcal{P}$ with $\mathcal{P}' \subseteq \mathcal{P}$, we define for any execution $e^{\mathcal{P}}$ the *abstraction* $e^{\mathcal{P}} \mid \mathcal{P}'$ of execution $e^{\mathcal{P}}$ to ports $\mathcal{P}'$ to be $e^{\mathcal{P}}$'s restriction to the domain $\mathcal{P}'$. Clearly an abstraction $e^{\mathcal{P}} \mid \mathcal{P}'$ is an execution, too.

Example 3.8. *Given the execution from Example 3.7, the abstraction $f^{\mathcal{P}'} := e^{\mathcal{P}} \mid \mathcal{P}'$ with $\mathcal{P}' = \{i, o\}$, has domain $\mathcal{P}' = \{i, o\}$ and value $f^{\mathcal{P}'}(i) = e^{\mathcal{P}}(i)$ and $f^{\mathcal{P}'}(o) = e^{\mathcal{P}}(o)$. Abstraction is used to strip down an execution of many ports to some ports only (here from $\{i, o, x\}$ to $\{i, o\}$).*

The inverse process of abstraction is combination. Given two disjoint, non-empty sets of ports $\mathcal{P}'$ and $\mathcal{P}''$, we say execution $e^{\mathcal{P}} := e_1^{\mathcal{P}'} \cup e_2^{\mathcal{P}''}$, with $\mathcal{P} = \mathcal{P}' \cup \mathcal{P}''$, is the *combination of* executions $e_1^{\mathcal{P}'}$ *and* $e_2^{\mathcal{P}''}$. Clearly one can separate the executions with, $e^{\mathcal{P}} \mid \mathcal{P}' = e_1^{\mathcal{P}'}$ and $e^{\mathcal{P}} \mid \mathcal{P}'' = e_2^{\mathcal{P}''}$ again.

A useful abbreviation, further, is $e^{\mathcal{P}}[\leqslant t]$, denoting the execution prefix of execution $e^{\mathcal{P}}$ until (and including) time $t \in \mathcal{T}$. Formally, for all $t \in \mathcal{T}$, we define $e^{\mathcal{P}}[\leqslant t]$ by

$$\forall p \in \mathcal{P} : e^{\mathcal{P}}[\leqslant t](p) := \mathrm{pre}(e^{\mathcal{P}}(p), t).$$

Finally, we will denote the set of all executions of a set of ports $\mathcal{P}$ by $\mathrm{Exec}(\mathcal{P})$.

3.3 Modules

Executions of ports $\mathcal{P}$ will be used to model the behavior of a hardware system comprising ports $\mathcal{P}$ over time $\mathcal{T}$. However, until now, we did not introduce any computational device corresponding to a process in classical systems that is capable of performing computations. The purpose of computational devices is to *restrict* the set of possible executions. As an example consider a zero-delay AND gate with input ports a, b and output port c: this gate (representing a hardware computing device) affects the execution of ports $\mathcal{P} = \{a, b, c\}$ by imposing the restriction that, for any time $t \in \mathcal{T}$, if the outcome of $\widetilde{a}(t) \wedge \widetilde{b}(t)$ is well-defined, then $\widetilde{c}(t) = \widetilde{a}(t) \wedge \widetilde{b}(t)$ has to hold. A hardware system thus can be modeled as a set of restrictions on the executions of its ports. Typically it is useful to group restrictions that result from a single hardware component together. For this purpose we introduce the concept of a module. A module $M = \langle I, O, E \rangle$ consists of

(i) a finite (possibly empty) set of input ports I,

(ii) a finite, non empty, set of output ports O, together with

(iii) a non empty set of possible executions E on the ports $I \cup O$ that module M produces.

E's form depends on whether I is empty or not. We therefore distinguish between

I is empty: Then E is a non empty set of executions of ports O.

I is non empty: For each $e_{in} \in \mathrm{Exec}(I)$, there is a non empty set of output executions $E_O(e_{in})$ of ports O. We refer to the executions in $E_O(e_{in})$ as *the output executions corresponding to input execution e_{in}*. $E_O(e_{in}) = \{e_{out}, e'_{out}, \dots\}$ means that M reacts to e_{in} with one of e_{out}, e'_{out}, etc. There are two extreme cases for the outcome of E_O: (i) $E_O(e_{in})$ is a singleton, in which case M reacts deterministically to e_{in} and (ii) $E_O(e_{in})$ is the set of all possible output executions, in which case M reacts arbitrarily. Let $\mathrm{React}(e_{in})$ be the set of all combinations of e_{in} and $e_{out} \in E_O(e_{in})$, i.e.,

$$\mathrm{React}(e_{in}) := \{e \mid e = e_{in} \cup e_{out} \text{ with } e_{out} \in E_O(e_{in})\}.$$

Then E is defined as

$$E = \bigcup_{e_{in} \in \mathrm{Exec}(I)} \mathrm{React}(e_{in}).$$

There are different ways to specify E_O, and by this, E: (i) By explicitly stating the restrictions on E in terms of formulas, which is done for the basic modules. (ii) By specifying E by means of *connected* submodules, which is done for compound modules. Hereby, the port x of a module M_1 is said to be *connected* to port y of a module M_2, iff $x = y$. A submodule M_1's output o_1 even can be another submodule M_2's and M_3's input. We, however, demand that a single input port must not be connected to more than one output port. The next paragraphs introduce two important basic modules that will be used throughout this thesis: the channel and the Boolean function modules.

Channel. A (FIFO) channel $C = \langle\{i\}, \{o\}, E\rangle = \mathrm{Ch}(i, o, \tau^-, \tau^+, v_0)$ is a module with one input port i and one output port o only. Its delay may vary during the execution in the range $[\tau^-, \tau^+]$. We define a *delivery time function* $d(t) : \mathcal{T} \to \mathcal{T}$ of the channel to be a continuous, strictly monotonically increasing ($t < t' \Rightarrow d(t) < d(t')$) function for which

$$d(t) - t \in \left[\tau^-, \tau^+\right]. \tag{3.8}$$

The channel's behavior is specified by the initial value conditions

$$\begin{aligned} &(0, v_0) \in \widehat{o} \text{ and} \\ &\nexists (t, v) \in \widehat{o} : \ 0 < t < d(0) \end{aligned} \tag{3.9}$$

as well as the existence of a delivery function d that fulfills the propagation condition

$$\forall t \in \mathcal{T} : \ \langle t, v\rangle \in \widehat{i} \Leftrightarrow \langle d(t), v\rangle \in \widehat{o}. \tag{3.10}$$

The delivery time function d may vary from execution to execution, i.e., for each execution of the channel, there must exist some d that correctly models the channel's delivery times. Because of d's properties (continuous and strictly increasing), d is a bijection from $\mathcal{T}$ to its codomain $d(\mathcal{T})$. More specifically, d maps every closed interval $[t_1, t_2]$ bijectively to the closed interval $[d(t_1), d(t_2)]$. Thus there exists an inverse function $d^{-1}(t)$ which again is continuous and strictly increasing. In the special case of a constant delay, i.e., $\tau^- = \tau^+$, we obtain $d(t) = t + \tau^-$ and $d^{-1}(t) = t - \tau^-$.

Since d carries over the total order of the events $\langle t, v\rangle$ in $\widehat{i}$ to the events $\langle d(t), v\rangle$ in $\widehat{o}$ [called *matching* events in the sequel], it follows immediately that $\widehat{o}$ is an event trace, if $\widehat{i}$ is an event trace.

Boolean function module. A (zero-delay) Boolean function module $B = \langle I, \{o\}, E\rangle = \mathrm{Bfm}(I, o, f)$ is a module with a finite set of inputs ports $I = \{a, b, \dots\}$ and one output port o. f is an arbitrary finite Boolean function with propositional variables from the set I. The module's behavioral equation is: an execution of B is in E, iff it holds: for each t in $\mathcal{T}$, if $f(\widetilde{a}(t), \widetilde{b}(t), \dots)$ is well-defined, then

$$\widetilde{o}(t) = f\left(\widetilde{a}(t), \widetilde{b}(t), \dots\right), \tag{3.11}$$

where the status of the execution's ports at time $t \in \mathcal{T}$ is obtained by (3.5). Note that, in contrast to the channel, (3.11) defines a Boolean function module's behavior in terms of the status of its executions; here we do not care about the exact output event trace as long as it has the correct status. We thus observe:

Observation 3.1. *For any Boolean function module $B = \langle I, \{o\}, E\rangle = Bfm(I, o, f)$, any input execution e_{in} of ports I and output executions e_{out} and e'_{out} of port o, if $e'_{out}(o) \in \widetilde{Event(e_{out}(o))}$ and $e_{out} \in E(e_{in})$, then $e'_{out} \in E(e_{in})$.*

Compound module. A compound module $M = \langle I, O, E\rangle = \mathrm{Cm}(I, L, O, A)$ is a module specified by a composition of submodules: I is the set of input ports, L the set of local ports and O, the set of output ports. A, the architecture, is a finite set of submodules $A = \{M^1, M^2, \dots\}$ with the submodule's inputs and outputs from $I \cup L \cup O$. We further demand that:

(i) For all pairs of distinct submodules $M^i = \langle I^i, O^i, E^i\rangle$ and $M^j = \langle I^j, O^j, E^j\rangle$ with $i \neq j$, the sets of outputs are distinct, i.e., $O^i \cap O^j = \emptyset$. This reflects the designer's rule of not connecting two submodule's outputs.

(ii) For all submodules $M^i = \langle I^i, O^i, E^i\rangle$, $O^i \cap I = \emptyset$. That is, submodule outputs are never connected to module inputs.

(iii) For every $s \in (L \cup O \cup \bigcup_i I^i)$, there is a submodule $M^j = \langle I^j, O^j, E^j\rangle$ with $s \in (O^j \cup I)$. The meaning is that every submodule's non-input port is driven by a submodule output or is a module's input port.

We now specify the predicate that *e_{sys} is a valid system execution for input execution e_{in}*, denoted by $\mathit{valid}(e_{sys}, e_{in})$, where e_{sys} is an execution of ports $I \cup L \cup O$ and e_{in} an execution of ports I, by

$\mathit{valid}(e_{sys}, e_{in}) :\Leftrightarrow$

(i) $e_{sys} \mid I = e_{in}$ and

(ii) for all submodules $M^i = \langle I^i, O^i, E^i\rangle$: $e_{sys} \mid O^i \in E^i_O(e_{sys} \mid I^i)$.

With this in mind we define the set of possible output executions for an input execution e_{in} as

$$E_O(e_{in}) := \{(e_{sys} \mid O) \mid \mathit{valid}(e_{sys}, e_{in})\}.$$

Clearly, there exist compound modules with input ports I and $e_{in} \in \mathrm{Exec}(()I)$, for which $E_O(e_{in}) = \emptyset$, i.e., which do not have a valid definition of output executions corresponding to input e_{in}. As an example for such a system see Example 3.9. In this case the module's possible executions are not well defined. Thus, before using a compound module, one always has to prove, that its executions are well defined.

Example 3.9. *Consider a module comprising only a single output port o and a single submodule, namely an inverter with zero-time loopback from the output to the input. Such an inverter can be modeled by the Boolean function module $\mathit{Bfm}(\{o\}, \{o\}, \widetilde{o}(t) = \neg\widetilde{o}(t))$. This module has no valid system executions: assume by contradiction that the system has a valid execution. In this execution, signal $\widehat{o}$ must comprise of an initial event $\langle t, I\rangle$, for some $I \in \mathbb{B}$. However, by the specification of the module, $I = \neg I$ — a contradiction.*

3.4 Problems

A problem P is a module, i.e., it is specified by the triple $P = \langle I, O, E \rangle$, where I are the input ports, O the output ports and E the possible executions. When we intend to emphasize the problem character rather than the module character of P, we print it in bold face, e.g., **SPF** is the short-pulse-filter problem, introduced and investigated in Section 6.1.

3.5 Restricting problems

In the context of a module $M = \langle I, O, E \rangle$, we often use the term *(input) environment* for a set $Env \subseteq \mathrm{Exec}(I)$. Given problem $\mathbf{PROBLEM} = \langle I, O, E \rangle$ and environment Env, we define **PROBLEM** *restricted to environment* Env, in short $\mathbf{PROBLEM} \restriction Env$, as the problem $\mathbf{PROBLEM}' = \langle I, O, E' \rangle$, with

$$E' := E \cup \{e_{in} \cup e_{out} \mid e_{in} \in (\mathrm{Exec}(I) \setminus Env) \wedge e_{out} \in \mathrm{Exec}(O)\}.$$

Informally, this means, that for every e_{in} in the input environment Env, $\mathbf{PROBLEM}'$ behaves as **PROBLEM**, while for every e_{in} not in the input environment, $\mathbf{PROBLEM}'$ behaves arbitrarily.

3.6 Problem solving

Let M be a module and **PROBLEM** be a problem both with the same set of inputs I and outputs O. Again we distinguish between two cases for I:

In case I is empty: We say that M *solves* **PROBLEM** iff

$$M.E \subseteq \mathbf{PROBLEM}.E. \tag{3.12}$$

In case I is non empty: In this case demanding (3.12) is too weak to get the intuitive meaning of problem solving: M could possibly not react to any input executions e_{in} at all and still satisfy (3.12). What we want is that, for *any* possible input execution e_{in}, M reacts in a way allowed by **PROBLEM**. It thus may only *not* react to input execution e_{in} if **PROBLEM** allows it to do so. Formally, let Env be an input environment, that is, a non empty set of executions of ports I. Then we say that M *solves* **PROBLEM** *in environment* Env iff

$$\forall e_{in} \in Env : M.\mathrm{React}(e_{in}) \subseteq \mathbf{PROBLEM}.E$$

or, which is equivalent, in the syntax of restricted problems

$$\forall e_{in} \in \mathrm{Exec}(I) : M.\mathrm{React}(e_{in}) \subseteq (\mathbf{PROBLEM} \restriction Env).E.$$

3.7 Implementation

When formalizing hierarchical systems, two design approaches are of importance: (i) Modules are built out of submodules corresponding to a bottom-up approach. For this purpose it is necessary for a formal model to precisely define the behavior of a module that is built from a set of submodules. This is what has been done by the introduction of the *compound module* in Section 3.3, whose behavior is defined by means of the behavior of its submodules. (ii) However, hardware systems are typically not designed solely in a bottom-up fashion. For the top-down approach, the designer specifies an abstract (that is, not yet physically implemented) module M, which is implemented later on by module M_I during the design process. Hereby, the designer does not care about M_I as long as it behaves like M, that is, solves problem M. This process has not yet been reflected in the proposed model. We thus proceed with:

Definition 3.1. *Module M_I* implements *module M (in environment Env), iff M_I solves problem M (in environment Env).* □

One of the key properties of the "implements" relation is expressed by the following lemma. Intuitively, it justifies the correctness of approach (ii) in the above paragraph in our modeling framework.

Lemma 3.1. *If M implements M' (in environment Env) and M' solves problem $\boldsymbol{P}$ (in environment Env), then M solves problem $\boldsymbol{P}$ (in environment Env).*

Proof. We distinguish two cases:

(i) $M.I$ is empty: Then, by Definition 3.1, $M.E \subseteq M'.E$. Further, $M'.E \subseteq \mathbf{P}.E$, yielding $M.E \subseteq \mathbf{P}.E$. The lemma holds.

(ii) $M.I$ is non empty: Choose an arbitrary e_{in} in Env. By Definition 3.1, $M.\text{React}(e_{in}) \subseteq M'.E$, that is $M.E_O(e_{in}) \subseteq M'.E_O(e_{in})$. The combination with $M'.E_O(e_{in}) \subseteq \mathbf{P}.E_O(e_{in})$ yields the desired result $M.E_O(e_{in}) \subseteq \mathbf{P}.E_O(e_{in})$. The lemma holds. ■

Furthermore, if two modules implement each other, we say that these modules are indistinguishable. In terms of a hardware design, these modules can be replaced by each other without changing the behavior of the whole chip. In formal terms:

Definition 3.2. *Two modules M and M' are* indistinguishable *(in environment Env), iff M implements M' (in environment Env) and M' implements M (in environment Env).* □

3.8 Failures

Since the aim of this chapter is to provide a formal framework for analyzing fault-tolerant clockless algorithms, and we have only focused on specifying the correct behavior of modules up to now, it remains to specify the incorrect behavior of modules. In [43], it was

shown that the effects of faults in physical hardware cannot just be restricted to benign failures while maintaining a high assumption coverage. Thus, we have to assume Byzantine, that is unrestricted, behavior of faulty modules. While modules can often easily be designed to deal with missing transitions on ports, the adverse power of Byzantine failures typically lies in the ability of a faulty module to generate wrong transitions (early or even spurious transitions) that are perceived inconsistently at different successor modules. Such failures could be the consequence of manufacturing defects or electrostatic breakdown [51], particle hits [4, 76], or electromagnetic noise [66], which may affect any module in a circuit. Due to different wire lengths and signal-level detection thresholds, such faults typically propagate differently to different successor modules. Note that we allow faulty modules to create even metastability [55], but we must assume that metastability cannot propagate beyond so called fault-containment regions (see below); the latter assumption will be discussed in detail in Chapter 6. After this brief motivation on which failures we would like the framework to be capable of expressing, the formal definitions follow:

A *distributed system* is specified by a compound module comprising of multiple *nodes*, which are sub-modules, and their interconnect.

We partition our system into multiple *fault-containment regions* (FCRs), i.e., sets of (sub-)modules, such that, (i) modules in the same FCR are potentially affected by a single fault like a particle hit and thus cannot be assumed to fail independently, and (ii) modules in different FCRs are not affected by the same fault.

Throughout the thesis, let C be the set of correct FCRs, and F, with $f := |F|$, the set of faulty FCRs. Clearly $P = C \cup F$ and $C \cap F = \emptyset$, i.e., C and F partition P. The set of correct (respectively faulty) modules thus is given as $\bigcup C$ (respectively $\bigcup F$).

Given a distributed system together with a partitioning into FCRs, one typically needs to constrain its set F such that, the distributed system still solves a problem in spite the $\bigcup F$ faulty modules. A widely used approach is to simply constrain $|F|$ by an upper bound f, the maximum number of faults that can occur in the system.

3.9 Related Work

Further reading: Part of the proposed system model has already been presented in [40] as well as [37], where the model has been applied to analyze a fault-tolerant on-chip algorithm.

Existing work: There exist a number of formal frameworks both from the distributed computing community and from the VLSI community:

In [59], Lynch and Tuttle introduced I/O Automata. An I/O Automaton comprises shared input and output actions for communicating with the environment, a state and transition rules. The latter are specified by (i) a transition name, (ii) a pre-condition (on the automaton's state) and (iii) its effect on the automaton's state. Whenever the pre-condition of a transition is true, it is applicable to the current state; and being applied leads to a new state. If multiple transitions are applicable, the scheduler chooses any one among

those and applies it. Clearly, additional constraints have to be put on the scheduler to guarantee fairness. As such, I/O automata are well suited to model classical asynchronous computations.

Merritt *et al.* [69] extended I/O automata by introducing time-constrained automata, a simplified version of which is presented in Lynch's book [58] as MMT Timed Automata: Transitions are grouped into tasks. Tasks are assigned lower and upper time bounds, say Δ^- and Δ^+, respectively. When a transition of a task becomes enabled, say at time $t \geqslant 0$, it has to occur within $t + [\Delta^-, \Delta^+]$, unless it is disabled (by its pre-condition becoming false) in the meantime. The pre-condition of a transition and its effect can be written in form of a guarded command of the form $P \Rightarrow \mathit{action}$, where P is a predicate that is true iff the pre-condition is fulfilled and *action* the action taken by the transition. As such the behavior of an internal (that is, not externally triggered) transition can be specified in the thesis's system model by means of equations of the form[1]

$$\begin{aligned} \forall t \geqslant 0 : P_0(t) &\Leftrightarrow (0,t) \in \widehat{S} \text{ and} \\ P_1(t) &\Leftrightarrow (1,t) \in \widehat{S}, \end{aligned} \tag{3.13}$$

where $P_0(t)$ and $P_1(t)$ are predicates on the status of a module's signals at time t, that specify when the status of signal $\widehat{S}$ should be set to 0 and 1 respectively.[2] The delay $[\Delta^-, \Delta^+]$ can be modeled by a FIFO channel with delay $[\Delta^-, \Delta^+]$ and input port S. Clearly, this approach allows an infinite number of events to occur at S during a finite duration, a property, that is forbidden for executions of time-constrained automata. These are only allowed to have a finite number of events during any finite interval of time.

General Timed I/O Automata were introduced by Lynch and Vaandrager [58,60]. They generalize time-constrained automata in several ways: most notably, in (a) allowing Zeno-traces, where a Zeno-trace is an execution with an infinite number of events occurring in a finite interval of time, and (b) replacing the task's time bounds with a time-passage event $\nu(t)$, $t \geqslant 0$, that increments state *now* (the current time) by t, and allowing a transition to have access to *now*. For example a transition's pre-conditions may depend on *now*. While the specification of Timed I/O Automata does not exclude Zeno-traces, they are not capable of specifying an execution that has both (i) an infinite number of events occurring during a finite prefix and (ii) events with unbounded time stamps. Thus continuous computation streams cannot be captured.

In the VLSI community, both exist untimed and timed models of computation exist as well. Martin [62] and Ebergen [26] both proposed frameworks that allow (i) to specify a system's behavior in terms of a language on the occurrence of transitions of its ports' states and (ii) to systematically map, i.e., compile, this language into a set of basic modules that have known physical implementations.

[1] Note, that (3.13) does not allow disabling.

[2] The system model only allows binary state signals. More complicated state signals must be represented by a set of binary signals.

In [62], the behavior of a system is stated in terms of guarded commands, as in (3.13), where guards are predicates on state variables and their effects transitions of the system's state. An example specification for an AND gate with input ports a and b and output port c is

$$\star[[a \wedge b \rightarrow c\uparrow \,|\neg(a \wedge b) \rightarrow c\downarrow]].$$

The proposed language allows to specify execution of guarded commands in sequence (";"), in parallel ("|") and their repetition ("$\star[[\ldots]]$"). The specification is then translated into a set of production-rules (guarded commands as in (3.13)), that are all executed in parallel. Analogously to [59, 69], a production rule's effect is only guaranteed to take place, if the rule's guard is enabled until the effect has occurred; otherwise the effect may not take place. *Stability* is the property that the rule's effect always take place before the guard is disabled again. The set of production rules, finally can be mapped to basic components, whose production rules are known. The method is time-free in the sense, that it does not contain any references to time, except for the *isochronic fork*, a forking wire with (nearly) equal propagation delays on both forks, that is added to the basic components. The resulting circuits are called quasi-delay insensitive (QDI) circuits.

In [26], Ebergen proposes a similar approach, however, in terms of transition logic: the system is not specified by a set of guarded commands (mapping state to events), but by *commands*, that specify a language over the alphabet of signal transitions: for each port a, symbol a (ended by either ? or ! depending on whether it is an input or output port) represents the occurrence of either $a\uparrow$ or $a\downarrow$. An example specification of a C-Element in an environment, where transitions on both a and b only occur after the effect (a transition of c) has taken place, is

$$\text{pref} \star [a?||b?; c!]. \tag{3.14}$$

Here, ";" denotes sequential, "||" parallel, "$\star[\ldots]$" repeated execution and "pref" is the prefix-closure operator. An execution in the language specified by (3.14) is $a; b; c; b; a$.

Both, [62] and [26], however, do not allow to introduce fine-grained timing constraints on paths between system components (besides isochronic forks), an important property of a formal framework when specifying fault-tolerant systems, since the behavior of faulty system components cannot be restricted, leading to violation of stability.

In [74, 75], Myers *et al.* proposed a system model that allows to capture detailed timing constraints: the order in which events occur is either specified by means analogous to production-rules (however, where guards are not on the system state, but on the occurrence of events) or by Petri net like graphs (orbital nets), which can be automatically derived from the production-rules. Both representations capture the causal structure of the occurrence of events together with numeric timing constraints on the maximum and minimum time between two causally related events. While this method allows to specify and verify a large class of clockless circuits, it has two drawbacks: (i) the large representation of state based logic (like combinatorial gates) in a purely event based specification language and

(ii) the explicit use of explicit numbers for timing constraints (instead of symbolic timing constraints).

An untimed framework that is capable of not only expressing AND causality between occurrences of events but also OR causality was proposed by Yakovlev *et al.* in [98]: they discuss existing, Petri net like, system models with respect to their ability to express two types of OR causality, namely *disjoint* and *joint* OR causality: constraints on rising and falling transitions of state variables are expressed via Petri net like graphs. In disjoint OR causality, causal predecessor events of an event e are not related: any occurrence of a predecessor event causes e to happen. That is, the number of occurrences of e until some time $t \geqslant 0$ is the *sum* of the numbers of occurrences of each predecessor event until t. Contrary, in joint OR causality, causal predecessor events of e are related: consider the example where the k^{th}, $k \geqslant 1$, occurrence of e depends on the k^{th} occurrence of any of its predecessor events by joint OR causality. Then, for all $k \geqslant 1$, only the first k^{th} occurrence of a predecessor event causes e to happen. All later k^{th} occurrences of predecessor events have no effect. Thus, the number of occurrences of e until some time $t \geqslant 0$ is the *minimum* over the number of occurrences of each predecessor event until t. Yakovlev *et al.* then introduce the *Causal Logic Net*, capable of capturing both joint and disjoint OR causality and represent it in an efficient way. The main idea is to assign each event in the Causal Logic Net an enabling function, that is, a Boolean predicate on the occurrences of the predecessor events. While joint OR causality is important when specifying fault-tolerant systems, the proposed framework does not allow for explicit use of time.

Chapter 4

Modeling Fault-tolerant Clockless Algorithms

We have presented a comprehensive modeling framework, which is expressive enough to specify the behavior of any fault-tolerant on-chip algorithm. In this chapter we try to identify key "building blocks" of fault-tolerant clockless algorithms and, step by step, build these building blocks in terms of our modeling framework.

We start this chapter by giving a very short introduction of how Petri nets can be used to model the control structure of clockless algorithms. Since Petri nets are too expressive for many clockless algorithms, we restrict our considerations to the subclass of event graphs. Starting from event graphs, we add more and more expressiveness to finally arrive at *threshold graphs* which turn out to be particularly well suited for modeling an important subclass of fault-tolerant clockless algorithms.

4.1 Petri Nets and Event Graphs

Since the author assumes that the reader is familiar with the basics of Petri nets, this section only very briefly summarizes definitions and results on Petri nets needed throughout the thesis. For an extensive introduction consult [73, 77], for example. In this thesis, we follow the notation of [15], since we will later in this chapter build on a generalization of Petri nets introduced therein.

A *Petri net* PN is a tuple $PN = \langle P, T, E, M_0 \rangle$, where P is a finite set of places, T a finite set of transitions, $E = E_{pre} \cup E_{succ}$ a relation, where for the predecessor set E_{pre} it holds $E_{pre} \subseteq P \times T$ and for the E_{succ} successor set it holds $E_{succ} \subseteq T \times P$. $M_0 : P \to \mathbb{N}$ is a function which assigns every place an initial number of tokens. An example Petri net with $P = \{a_w, b_w, c_w, a_p, b_p, c_p\}$ and $T = \{t_w, t_a, t_b, t_c\}$ is depicted in Figure 4.1. Relation E is visualized by arcs — whenever $\langle x, y \rangle \in E$ we draw an arrow from x to y. The initial

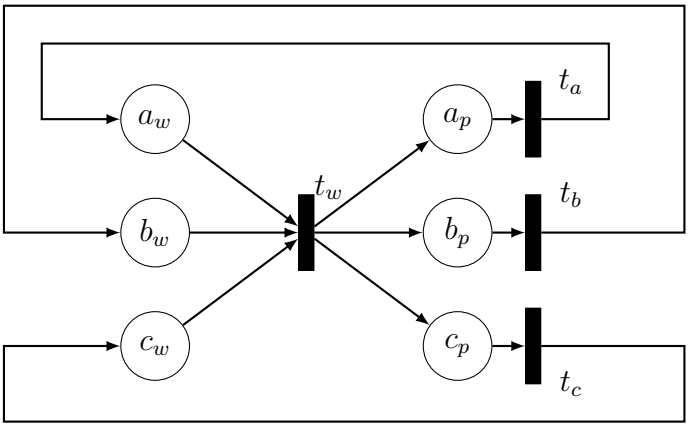

Figure 4.1: Example Petri net

configuration of the Petri net, M_0, is not drawn in Figure 4.1. Typically one depicts $M_0(p) = k$ for some $p \in P$ and $k \in \mathbb{N}$ by placing k black tokens in place p.

Due to its length, we do not give a formal definition of an execution of a Petri net here. Rather we will give (much shorter) definitions for executions of certain subclasses of Petri nets. Let us start with an important subclass, the so called *event graphs* [15][1]: Let the *predecessor places* (respectively, *successor places*) of transition t_x, denoted by $\bullet t_x$ (respectively, $t_x\bullet$), be

$$\bullet t_x := \{p \mid \langle p, t_x\rangle \in E_{pre}\} \text{ and}$$
$$t_x\bullet := \{p \mid \langle t_x, p\rangle \in E_{succ}\}.$$

Analogously the *predecessor transitions* $\bullet p$ (respectively, successor transitions $p\bullet$) of a place p, are defined as

$$\bullet p := \{t_x \mid \langle t_x, p\rangle \in E_{succ}\} \text{ and}$$
$$p\bullet := \{t_x \mid \langle p, t_x\rangle \in E_{pre}\}.$$

An *event graph* is a Petri net for which

$$\forall p \in P : |\bullet p| \leqslant 1 \wedge |p \bullet| \leqslant 1 \tag{4.1}$$

holds. Property (4.1) constraints a place p to have at most one predecessor and at most one successor transition. The Petri net depicted in Figure 4.1 thus is an event graph. The restriction comes at the price of reduced expressibility; its most important effect being the impossibility to model choice (a place has more than one successor transition) in event graphs. However, we will see that event graphs are expressive enough to model the behavior of many clockless algorithms.

[1] In [73] the same subclass is called *marked graph*.

Let a *state* of the event graph PN be a function $\mu : P \to \mathbb{N}$, where $\mu(p)$ is the number of tokens that reside in place p. A transition t_x is said to be *enabled in* μ, iff for all predecessor places $p \in \bullet t_x$, $\mu(p) \geqslant 1$ holds. A transition t_x is said to *fire in* μ *resulting in* μ', iff (i) t_x is enabled in μ and (ii)

$$\forall p \in \bullet t_x : \mu'(p) = \mu(p) - 1 \text{ and}$$
$$\forall p \in t_x \bullet : \mu'(p) = \mu(p) + 1.$$

Then an *execution* e *of* PN is a possibly infinite sequence of states $\mu_0, \mu_1, \ldots$, where initially

$$\forall p \in P : \mu_0(p) = M_0(p) \tag{4.2}$$

and for any μ_{i+1}, with $i \geqslant 0$, in the execution, μ_{i+1} is the result of a (possibly empty) subset of T firing in μ_i:

Example 4.1. *Consider the event graph PN given in Figure 4.1 with* $M_0(a_p) = M_0(b_p) = M_0(c_p) = 1$ *and* $M_0(a_w) = M_0(b_w) = M_0(c_w) = 0$. *PN can be interpreted to model the behavior of three computing devices* a, b *and* c *which initially all are in their processing state,* a_p, b_p *and* c_p *(modeled as tokens in the respective places). As soon as a process, say* a*, has finished processing, it changes into its waiting state, say* a_w*. When all processes reside in their waiting states, they interchange their processing results and afterwards start the next phase of processing, by changing into the processing state again.*

Example 4.1 is a very simple example for a set of collaborating processes. While its structure has commonalities with the triple-mode redundancy (TMR) technique used to make synchronous designs fault-tolerant, it is not fault-tolerant. A simple crash of one of the processes would result in a blocking of the other correct two processes. During this chapter, we will derive techniques which in fact are analogous to TMR in synchronous designs. While it is not possible to directly apply [30] or techniques therein, without clearly specifying the exact problem, there is a strong indication that untimed event graphs are not expressive enough for modeling hardware components that solve non-trivial fault-tolerant problems. Thus, a first feature that has to be added to event graphs to allow the modeling of fault-tolerant designs are timing properties.

4.1.1 Timed Event Graph

While event graphs can be used to model asynchronous behavior, they are not expressive enough to model some behavior that is restricted by timing constraints. To overcome this restriction, we introduce a function $D : E_{pre} \to (\mathcal{T} \times \mathcal{T})$, which assigns every edge in E_{pre} a delay interval $[\tau^-, \tau^+]$, with τ^- and τ^+ in $\mathbb{R}_0^+$. Further, we distinguish between two types of places in P: (i) binary places and (ii) unbounded places, the former of which are visualized as places with a double boundary line. In terms of the visualization of timed

event graphs, binary places can hold only 0 or 1 tokens. However, unlike 1-bounded places of standard Petri nets, where a place p's predecessor transition is forbidden to fire if the 1-bounded place p already holds one token, we allow firing of the predecessor transition, with the effect that the token in p is overwritten, resulting in 0 tokens in p. In contrast, unbounded places can store an arbitrary number of tokens. A timed event graph thus becomes a tuple $TEG = \langle P, \rho, T, E, M_0, D\rangle$, where ρ assigns every place p in P whether it is binary ($\rho(p) = 2$) or unbounded ($\rho(p) = \infty$).

Behavior of Timed Event Graph. We are now equipped to specify the behavior of timed event graphs in terms of the formal framework introduced in Chapter 3. Assume that $TEG = \langle P, \rho, T, E, M_0, D\rangle$ is given. Let the set of ports $\mathcal{P}$ be defined by

$$\mathcal{P} = P \cup \{\mathrm{rcvd}_{p,q} \mid p \in P,\ t_p \in {\bullet p} \text{ and } q \in {\bullet t_p}\}. \tag{4.3}$$

Intuitively, $\mathrm{rcvd}_{p,q}$ represents the output port of a channel from q to p and is formalized later on. Note that, for any $p \in P$, if there exists a t_p in ${\bullet p}$, then it must be unique by (4.1). We say that *e is an execution of TEG*, iff all of the following properties hold:

(i) e is an execution of the ports $\mathcal{P}$, where for all p in $\mathcal{P}$ and t in $\mathcal{T}$, $\#p(t)$ is well-defined.

(ii) For each $\mathrm{rcvd}_{p,q} \in \mathcal{P}$, port $\mathrm{rcvd}_{p,q}$ is the output port of a channel with input port q, initial value 0 and delay bounds $D(\langle q, t_p\rangle)$, where t_p is the unique transition in ${\bullet p}$. In terms of $\mathrm{rcvd}_{p,q}$'s status, we thus demand that there exists a delay function d with bounds $D(\langle q, t_p\rangle)$ such that,

$$\widetilde{\mathrm{rcvd}}_{p,q}(t) = \begin{cases} 0 & \text{if } d^{-1}(t) \text{ undefined} \\ \widetilde{q}(d^{-1}(t)) & \text{else} \end{cases} \tag{4.4}$$

We will also refer to d as the *outgoing delay function of q.*

(iii) For each $p \in \mathcal{P}$ with unique $t_p \in {\bullet p}$: let P_2 be the set of binary places in ${\bullet t_p}$ and P_∞ the set of unbounded places in ${\bullet t_p}$. Then for any $t \in \mathcal{T}$ and $v \in \mathbb{B}$, $\langle t, v\rangle$ in $\widehat{p}(t)$ iff,

(a) for all q in P_2, $\widetilde{\mathrm{rcvd}}_{p,q}(t) = v$ and

(b) $\#p(t) \leqslant M_0(p) + \min\{\#\mathrm{rcvd}_{p,q}(t) \mid q \in P_\infty\} \cup \{+\infty\}$ is not violated.

Note, that we do not constrain $\#p$ in case there exists no preceding transition t_p. Places with no preceding transition are input places and will play an important role in the next section.

There are two special cases, where condition (iii) can be simplified significantly:
In case ${\bullet t_p}$ comprises binary places only: then condition (b) is trivially fulfilled and signal $\widehat{p}$ is defined as: for any t in $\mathcal{T}$ and v in $\mathbb{B}$, $\langle t, v\rangle$ in $\widehat{p}(t)$ iff, for all q in P_2, $\widetilde{\mathrm{rcvd}}_{p,q}(t) = v$.
In case ${\bullet t_p}$ comprises unbounded places only: here, condition (a) is trivially fulfilled and we thus obtain: for all t in $\mathcal{T}$,

$$\#p(t) = M_0(p) + \min\{\#\mathrm{rcvd}_{p,q}(t) \mid q \in {\bullet t_p}\}. \tag{4.5}$$

Timed Event Graph Module. Since the behavior of timed event graphs can be expressed within the formal framework of Chapter 3, one can specify the behavior of a module via a timed event graph. Let $TEG = \langle P, \rho, T, E, M_0, D\rangle$ be a timed event graph, where $P = I \cup O \cup L$ is partitioned into the three sets I (the input places), O (the output places) and L (the local places). We demand two properties to hold on the *TEG*:

(P1) Input places have no preceding transitions and exactly one succeeding transition. Output places have no succeeding transitions and exactly one preceding transition, while all local places have exactly one preceding and succeeding transition.

(P2) Input places and output places are binary places.

We are now equipped to define the set of executions E of module $M = \langle I, O, E\rangle = \mathrm{TEGm}(P, \rho, T, E', M_0, D)$: consider an arbitrary execution e of ports $\mathcal{P}$ as defined in (4.3). In case $e \mid I$ has a well defined counting representation, $e \mid P$ in E if e is an execution of *TEG*. In case $e \mid I$ does not have a well defined counting representation, e in E, i.e., the module's possible reactions are unconstrained.

Example 4.2. *The Muller C-Element is a frequently used basic building block for clockless algorithm designs [92]. A zero-latency Muller C-Element is a module with two input ports a and b and one output port c, whose behavior is specified by: for all t in $\mathcal{T}$ and v in $\mathbb{B}$, in case $\#a$ and $\#b$ are well-defined for all times $t \in \mathcal{T}$,*

$$\langle t, v\rangle \in \widehat{c} \;:\Leftrightarrow\; \widetilde{a}(t) = \widetilde{b}(t) = v.$$

An equivalent behavioral definition of a Muller C-Element in terms of a timed event graph is given in Figure 4.2.

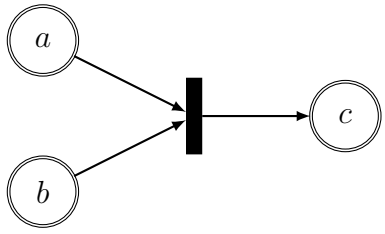

Figure 4.2: Muller C-Element

4.2 And-or Event Graph

Adding time to event graphs alone does not add the capability to model fault-tolerant systems. Again, consider Example 4.1. Intuitively the crash of process a, i.e., t_a not firing anymore, still results in an execution where no process makes a step.

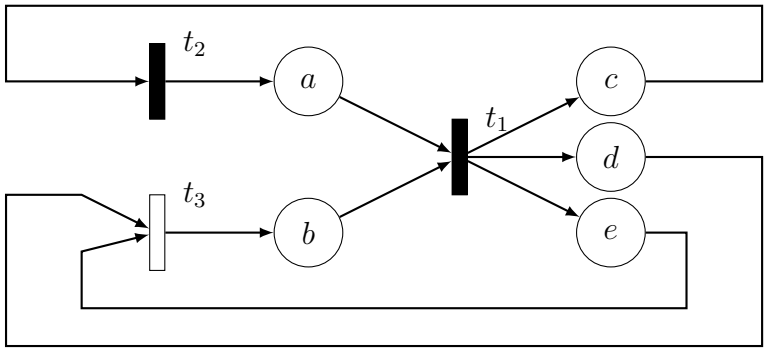

Figure 4.3: Timed and-or Event Graph

To overcome this restriction we would need to replace t_x by a different kind of transition, one that does not have "wait-for-all" semantics. There exists work form both the VLSI community and the control theory community [15, 44, 98] which have in common that they add a transition with "wait-for-first" semantics, also called Or-Transition, in contrast to the And-Transitions with the "wait-for-all" semantics. During this work we will follow [15] and name the resulting graph an and-or event graph. Figure 4.3 depicts an example graph, where And-transitions are depicted as filled bars, while Or-transitions are depicted as unfilled bars.

Formally a timed and-or event graph (or short and-or graph) AOG is defined as a tuple $\text{AOG} = \langle P, \rho, T, E, M_0, D, \theta \rangle$ where all variables are defined analogously as for the timed event graph, except for the additional $\theta : T \to \{\min, \max\}$ which maps a transition to a type. With the intention to keep the behavioral specification of an and-or graph simple, we demand: for p in P with preceding transition t_p, where $\theta(t_p) = \max$, all places in $\bullet t_p$ are unbounded. In analogy to (4.5) the behavior for those signals $\widehat{p}$ is then specified by: for all t in $\mathcal{T}$,

$$\#p(t) = M_0(p) + \max\{\#\text{rcvd}_{p,q}(t) \mid q \in \bullet t_p\}.$$

The behavior of all other ports is specified analogously to the ports in timed event graphs.

Executions of and-or event graphs can be visualized by tokens and anti-tokens traversing through the graph. For this purpose let the number of tokens residing at place p at time t be $k = \#p(t) - \#q(t)$ if p is an unbounded place and $k = (\#p(t) - \#q(t)) \bmod 2$ if p is a binary place, where q is an arbitrary "successor" place ($q \in (p\bullet)\bullet$).[2] In case $k \geqslant 0$, we draw k full tokens at place p. Otherwise, if the number of tokens is negative, we draw $|k|$ anti-tokens, that is, unfilled tokens at place p.

[2]Note that for any two successor places q and r of p, $\#q(t) = \#r(t)$ which justifies the choice to use an arbitrary one among them.

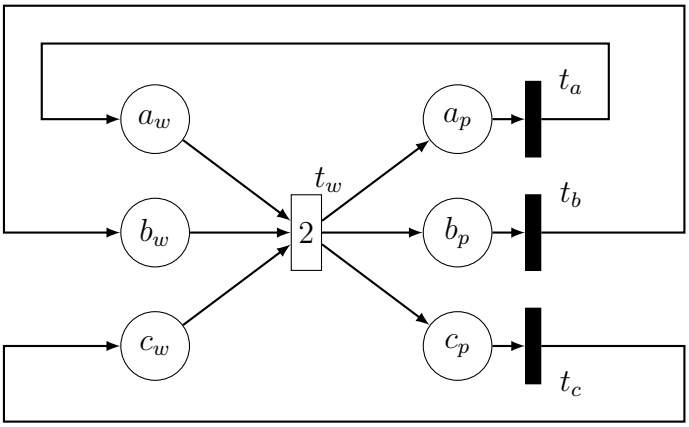

Figure 4.4: Example Threshold graph

4.3 Threshold Graph

We will next generalize and-or graphs to threshold graphs. While this generalization does not add any extra capability in terms of expressiveness, it provides means to replace large sets of and-or transitions and places, which are interconnected in some regular way, with new kinds of transitions, namely *threshold transitions*.

Timed event graphs can be modeled as and-or graphs whose range of θ is restricted to $\{\min\}$, while and-or graphs have a θ with range $\{\min, \max\}$. We can extend the range even more. Consider a transition t_p and let the number of the input places of t_p be n. Then $\theta(t_p)$ may be of any type out of the set $\{1^{\text{st}}, 2^{\text{nd}}, \ldots, n^{\text{th}}\}$, where k^{th} applied to a multi-set S, with $|S| \geqslant k$, returns the k smallest element. Thus as special cases $1^{\text{st}} = \min$ and $n^{\text{th}} = \max$. A transition t_p with $\theta(t_p) = k^{\text{th}}$ will be called threshold transition. A threshold transition is depicted as an unfilled bar with the number k inside the bar. The special cases for min respectively max still are depicted as filled respectively unfilled bars. An example threshold graph is depicted in Figure 4.4.

It remains to define the set of possible executions of a threshold graph. The behavioral specification of a threshold graph is analogous to the specification of an and-or graph: given place p in P with preceding transition t_p, where $\theta(t_p) \neq \min$, we require all places in $\bullet t_p$ to be unbounded. The behavior of signal $\widehat{p}$ is then specified by: for all t in $\mathcal{T}$,

$$\#p(t) = M_0(p) + \theta(t_p)\{\#\text{rcvd}_{p,q}(t) \mid q \in \bullet t_p\}.$$

The behavior of all other ports is defined analogously to those in and-or graphs.

Example 4.3. *Analogous to Example 4.1, consider a system comprising three nodes a, b and c. Each process p either is in the waiting state p_w or in the processing state p_p. However, unlike in Example 4.1, a process in the waiting state does not wait for all processes to reach the waiting state, before it enters the processing state, but only for a majority of (two) processes. This behavior can be modeled via the threshold graph depicted in Figure 4.4.*

Although a formal definition of the failure of a process has not been given yet, the example system above intuitively tolerates the crash failure of a single process.

4.4 Modeling Control Circuits

Clockless circuits can be modeled at different levels of abstraction. In Chapter 3 we have introduced modules and ways to specify the behavior of modules. During the design process, the circuit designer typically uses even higher levels of description than available for modules. One such abstraction, not possible with modules, is to group ports together to buses. For example a typical asynchronous interface, when considering bundled data communication, comprises: (i) a binary request port (short req), (ii) a data bus and (iii) a binary acknowledge port (short ack). The communication interface between two such modules is depicted in Figure 4.5. Here module A pushes data to module B by sending

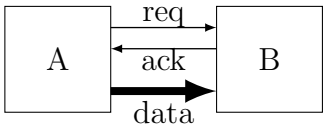

Figure 4.5: Asynchronous communication

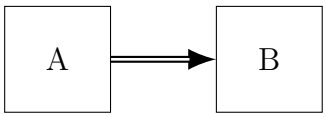

Figure 4.6: High level view

the data value over the data bus and issuing a request over the req line. After B has successfully captured A's data, it issues an acknowledge back to A over the ack line. As soon as A receives the acknowledgement, it may push the next data item to B.

Sparso describes in [87] a high level graphical notation, where he abstracts from the specific communication interface, by drawing only a single "⇒" between the modules, as done in Figure 4.6. This has two benefits: (i) the graphical notation of a system becomes smaller and less tedious to paint and (ii) the abstract communication interface can be substituted by any asynchronous communication interface. The decision of using, e.g., four-state logic communication instead of bundled data communication is thus deferred to a later design stage. In this thesis, however, we restrict ourselves to *2-phase bundled data communication*, that is explained in the following. The protocol of data exchange between two modules communicating by 2-phase bundled data communication is depicted in Figure 4.7. It shows the state of ports ack, req and data at their respective origin, i.e., req and data at A's output ports and ack at B's output port. For convenience, we say that A (respectively B) *makes a step at time* t, $t \geqslant 0$, iff A produces a transition on its req (ack) port at time t.

The protocol may be started from one of two possible initial states, depending on whether initially (at time 0) (i) B waits to receive data from A or (ii) A waits to receive an acknowledge from B:

(i) Assume that module A initially is waiting to receive an acknowledgement from B. In this case let the ports' status be $\widetilde{req}(0) = \widetilde{ack}(0) = 0$. As soon as B is ready to receive data

from A, it makes a step, thereby producing a transition on ack. This instant is depicted in Figure 4.7 by vertical line ❶. As soon as A receives the transition and is ready to provide data, it applies the data on the bus and makes a step, thereby producing a transition on req. As soon as B receives the req transition, B can be sure that the data on the bus is valid (assuming that the propagation delays of data and req are matched), and makes a step, thereby again producing a transition on ack, as soon as it is ready to receive new data (e.g., when it has safely stored the data). The protocol repeats.

(ii) Otherwise, assume that module B initially is waiting to receive data from A. In this case we let $\widetilde{req}(0) = 0$ and $\widetilde{ack}(0) = 1$. The initial state is depicted by the vertical line ❷ in Figure 4.7. As soon as A is ready to provide data, it makes a step and thereby issues a request and additionally applies its data on the data bus. As soon as B receives the req transition, B can be sure that the data on the bus is valid and makes a step, thereby producing a transition on ack, as soon as it is ready to receive new data. When A receives the ack transition, it may repeat as before.

We thus observe for any two modules A and B communicating by the 2-phase bundled data protocol:

Observation 4.1. *During an execution of two modules A and B communicating by the 2-phase bundled data protocol, A and B alternate in making steps, where, depending on the protocol's initial state either B (in case of (i)) or A (in case of (ii)) makes the first step.*

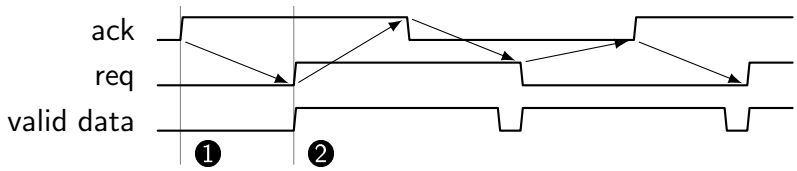

Figure 4.7: 2-phase bundled data communication

A simple, but central, module is the *register*. A register R has one input communication interface and one output communication interface. The input communication interface is connected via channels to a predecessor module that provides data to R. Likewise the output communication interface is connected via channels to a successor module that receives data from R. A register R can only store one data word at a time. It behaves as follows: as soon as R has received both (i) a request (and by this, valid data) from its predecessor module over the input communication interface and (ii) an acknowledge from its successor module over its output communication interface, it *makes a step*, that is, (i) takes over the data applied to the input communication interface (storing it in its local one word memory) and issues an acknowledge over the input communication interface as well as (ii) provides the newly captured data to the output interface and issues a request over the output interface. Alternative registers exist that provide more decoupling between the input and output communication interface by the possibility to store a second word [87, 92]

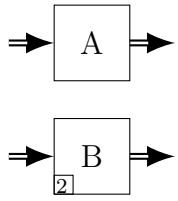

Figure 4.8: Queues

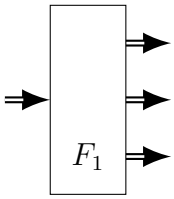

Figure 4.9: Fork

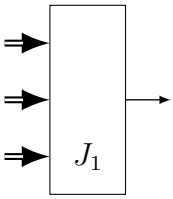

Figure 4.10: Non-buffering/General Join

(thus the instant when data on the input communication interface is captured and the instant when that data over the communication interface is provided is not necessarily the same).

Besides registers, Sparso further introduces the fundamental design modules forks and joins (among others) and later on discusses their physical implementations in [87].

Though the method presented in [87] is very appealing, for fault-tolerant asynchronous designs it is not possible to strictly stick to the asynchronous communication scheme based on handshaking in all cases: A single faulty module that deadlocks may prevent all its successor modules from performing computations, by simply not issuing the acknowledge signal. We therefore extend our high level view with a communication interface comprising of no acknowledge line. Whenever we make use of this communication interface between two modules, we denote it with a thin "→", between the two respective modules.

4.4.1 High Level View

We follow [87] and introduce a set of fundamental modules in the high level model: queues, forks and non-buffering, respectively, General Joins which are depicted in Figures 4.8, 4.9 and 4.10. After an informal description of these modules, an exact definition in terms of our system model is given.

Queue. Queues are FIFO buffers, which are written by a single producer module and read by a single consumer module. There are two kinds of queues: unbounded and bounded queues. While unbounded queues do not have an apriori bound on their maximum fill size, bounded queues have. An unbounded and a bounded queue are depicted in Figure 4.8. For bounded queues the maximum fill size is drawn in the lower left corner. Bounded queues with maximum fill level 1 are identical to *registers*.

Fork. A fork is a data broadcast. Forks are used whenever the same data has to be processed by different successor modules along different lines of computation. A fork for three successor modules is depicted in Figure 4.9. The forks we use here are identical to those introduced in [87]. Forks handshake with all successor modules by waiting to receive acknowledges from *all* of them before acknowledging its predecessor module.

Non-buffering Join. While forks are used to broadcast data, their counterparts, non-buffering joins, are responsible to merge data coming along different data paths. A non-buffering join (comparable to the join in [87]) is a module that allows the join's successor module to handshake with all predecessor modules of the join. The join performs this task, by waiting to receive requests and data from *all* predecessor modules before it hands over the merged data and a request to the successor module. The acknowledge it receives from the successor module is simply forwarded to all predecessor modules. Note that non-buffering joins do not intermediately store data from the predecessor modules and are thus named "non-buffering".

General Join. In case of fault-tolerant on-chip algorithms a join's successor module cannot wait for all predecessor modules' data. Otherwise a single faulty predecessor module could prevent the successor module from making progress. Therefore we generalize a join's behavior as we did for and-transitions: let J be a General Join. Then a type $\theta(J)$ is assigned to J, where the range of θ is defined in analogy to θ of a transition; however, we allow for a more general type $\theta(J)$ than a single threshold, namely,

$$\theta(J) \in \{k_0^{\text{th}}, k_1^{\text{th}}, \langle k_0^{\text{th}}, \ell_1^{\text{th}}\rangle \mid 1 \leqslant k, \ell \leqslant |\mathcal{Q}|\}, \tag{4.6}$$

where $\mathcal{Q}$ is the set of J's predecessor modules. We then define: For a multiset S, $k_i^{\text{th}}(S)$, where $1 \leqslant k \leqslant |S|$, is given by

$$k_i^{\text{th}}(S) := k^{\text{th}}\{x - i \mid x \in S\} \tag{4.7}$$

and $\langle k_0^{\text{th}}, \ell_1^{\text{th}}\rangle(S)$, where $1 \leqslant k, \ell \leqslant |S|$, is given by

$$\langle k_0^{\text{th}}, \ell_1^{\text{th}}\rangle := \max\{k_0^{\text{th}}(S), \ell_1^{\text{th}}(S)\}. \tag{4.8}$$

Figure 4.10 shows a General Join J_1 for three predecessor modules. In this work we require a General Join to have a "$\rightarrow$" communication interface to its successor module. This implies, that the successor module cannot put backpressure onto the General Join, by delaying its acknowledge signal.[3] Clearly the Fork module suffers the same problems as the Non-buffering. Here the same techniques could be applied, obtaining a General Fork module. For simplicity of the thesis' presentation we will however concentrate on the Join Module and obtain a fault-tolerant Fork module simply by dismissing the Fork's acknowledge signals, leaving it with "$\rightarrow$" communication interfaces only.

The idea behind the generalized $\theta(J)$ is to not only allow J to wait for the data of a certain threshold of predecessor modules, but, in case $\theta(J) = \langle \alpha_0^{\text{th}}, \beta_1^{\text{th}}\rangle$ to combine two thresholds with distinct offset together: J will wait for the k^{th}, $k \geqslant 0$, data item from at least α predecessor modules, *or* $k+1^{\text{st}}$ data item from at least β predecessor

[3] It is definitively of interest to eliminate this simplifying restriction in future work.

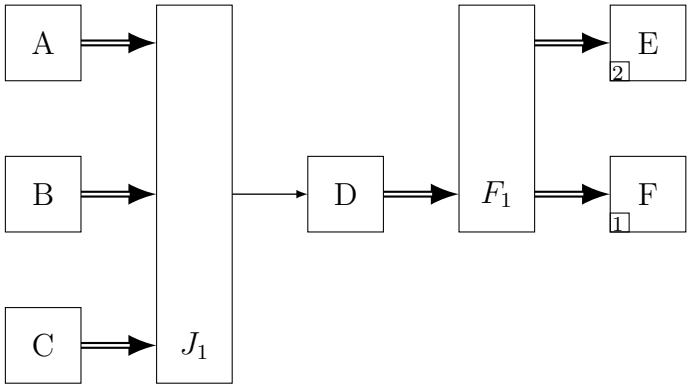

Figure 4.11: Example high level view

modules, before it outputs the $k+1^{\text{st}}$ data item. Clearly, a General Join requires a data buffer for each predecessor module, since the successor module makes progress even if data has not been received from all predecessor modules. Thus buffering is necessary in order not to loose the predecessor modules' data (a reception of the k^{th} data item could be taken for the reception of the $k+1^{\text{st}}$ data item).

While the motivation of the two generalizations of timed event graphs, the first to and-or graphs and the second to threshold graphs, are natural generalizations on the way to express fault-tolerant on-chip algorithms, the combination of two thresholds in a single module does not have a straightforward application. However, it is shown in Chapter 5 that General Joins with two thresholds play an important role in Byzantine fault-tolerant on-chip algorithms like the simulated authenticated broadcast primitive introduced by Srikanth and Toueg in [88] and refined by Widder and Schmid in [97].

Example 4.4. *An example high level model of a simple system is shown in Figure 4.11. Here, three queues A, B and C feed their data into a General Join J_1. The type of the General Join is not further specified here. The result is fed into queue D (without being acknowledged, which is denoted with the "$\rightarrow$" communication interface). D's results are then broadcast (via fork F_1) to two bounded queues E and F.*

It remains to formally specify the queue, fork, non-buffering join and General Join modules. A common technique is to partition a module in a control part (comprising of the req and ack input and output ports) and a data part (comprising of the data lines) and only model the control part. The simplification is justified whenever the data values have no direct consequences on the signals of the control ports. Since this holds for many typical asynchronous circuits, we might follow [87, 92] and model only the control structure. The data path can be added at a later stage by analogous methods as described in [87, 92]. For our purposes (2-phase bundled data) it suffices to think of a simple coupling between control path and data path: transitions (both 0-1 and 1-0) of a module's control signals trigger associated flip-flops in the data path.

4.4.1.1 Queue Module

The unbounded queue module is defined as a module $M = \langle I, O, E \rangle$, with two input ports $I = \{\text{reqI}, \text{ackI}\}$ and two output ports $O = \{\text{reqO}, \text{ackO}\}$. Its set of possible executions E is specified via the timed event graph shown in Figure 4.12. Note that delays D and initial values M_0 are omitted on purpose, as they depend on the queue to be modeled. For example, $M_0(q) = k \geqslant 1$ models a queue prefilled with k data items.

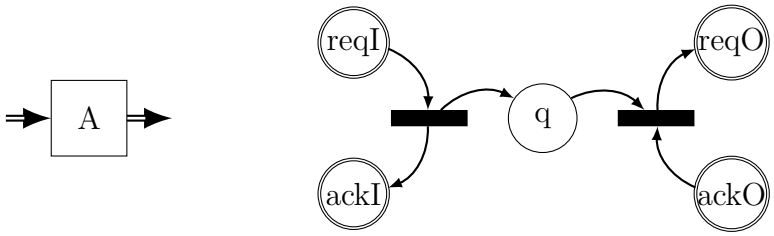

Figure 4.12: Unbounded Queue Module

The bounded queue module of size 1 is defined as a module M with identical input and output ports as the unbounded queue module, however, E is specified by the timed event graph shown in Figure 4.13. A bounded queue module of size $k > 1$ can be modeled as a compound module comprising of a sequence of k bounded queue modules of size 1 each.

4.4.1.2 Non-buffering Join Module

Consider a non-buffering join module with predecessor modules $\mathcal{Q} = \{A, B, \dots\}$. The module is defined to have input ports $I = \{\text{reqA}, \text{reqB}, \dots\} \cup \{\text{ackO}\}$ and output ports $O = \{\text{ackA}, \text{ackB}, \dots\} \cup \{\text{reqO}\}$. The module's behavior is specified by the timed event graph depicted in Figure 4.14; for reasons of simplicity, it shows only two predecessor modules A and B. Furthermore, delays and initial number of tokens are not depicted.

4.4.1.3 General Join Module

A General Join module with predecessor modules $\mathcal{Q} = \{A, B, \dots\}$ is defined as a module with input ports $I = \{\text{reqA}, \text{reqB}, \dots\}$ and output ports $O = \{\text{ackA}, \text{ackB}, \dots\} \cup \{\text{reqO}\}$.

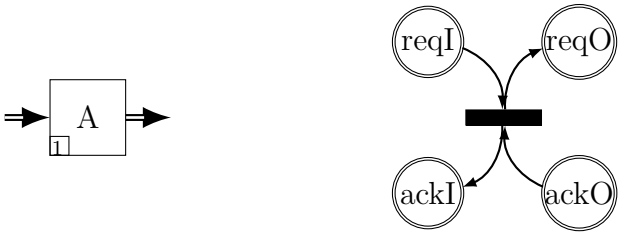

Figure 4.13: Bounded Queue Module of size 1

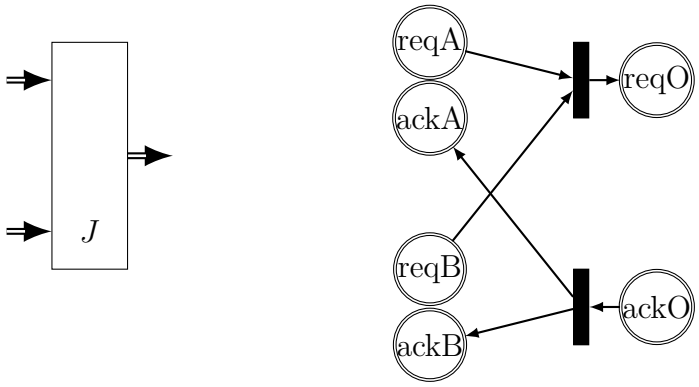

Figure 4.14: Non-buffering Join Module

Its behavior is specified by the threshold graph depicted in Figure 4.15. In the figure, the threshold graph is only depicted for a General Join with two predecessor modules A and B. A generalization to more than two predecessor modules is obtained by a straightforward replication of the thresholds graphs' structure. In the threshold graph, we set all delays to 0, except for those explicitly depicted in the figure. Further the initial number of tokens at all places, except for s and X'^r, with X in $\mathcal{Q}$, are set to 0. For place s, $M_0(s) = 1$ and for all X in $\mathcal{Q}$, $M_0(X'^r) = -1$.

We will start with an informal description of the behavior of a General Join module: the graph can be partitioned into (i) the component comprising of the places reqX, ackX, X^r and X^l, for X in $\mathcal{Q}$, (ii) the places X, for X in $\mathcal{Q}$, th, th' and s, and (iii) places reqO and l. Subgraph (i) roughly looks like unbounded queues without output. It is responsible for storing transitions that arrived at input $reqX$, separately for each X. Subgraph (ii) determines how many times the threshold $\theta(J) = \langle \alpha_0^{\text{th}}, \beta_1^{\text{th}} \rangle$ has been reached in X and stores the result in s. Finally, (iii) generates the output transitions at reqO after a latency delay in $\tau_{PC}^{\pm} + \tau_{TH}^{\pm}$, as well as feeds back the generated transition via l to X^r and X'^r, for all X in $\mathcal{Q}$. This feedback serializes the output transitions generated by J to occur one after the other.

4.4.2 Timing Properties of General Joins

Since General Join modules shall be used as building blocks in fault-tolerant on-chips algorithms a formal characterization of their correctness and timing properties is inevitable.

Before we formally state important properties of a General Join J, we introduce some notation: for time t in $\mathcal{T}$, $k \geqslant 1$ and predecessor module X of General Join module J, we say J *receives transition* k *from* X *at time* t, iff reqX makes transition k at time t. Analogously, we say J *sends transition* k *at time* t iff reqO makes transition k at time t.

In the following, we assume that $\#\text{reqX}(t)$, for each X in $\mathcal{Q}$, is well defined and right-continuous for all t in $\mathcal{T}$ unless stated otherwise. Further assume that $\theta(J) = \langle \alpha_0^{\text{th}}, \beta_1^{\text{th}} \rangle$,

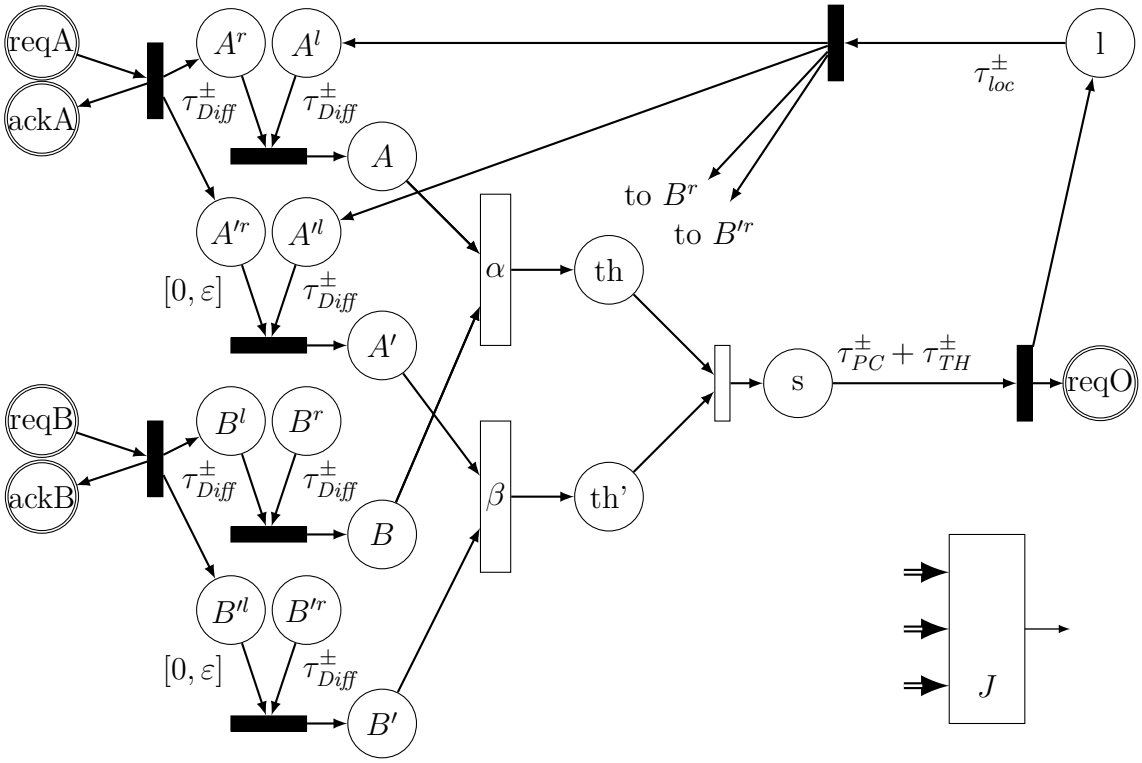

Figure 4.15: General Join Module

for some α and β. Later on in the thesis it will be shown that this assumption is non restrictive, since we can easily extend our results to the simpler case where $\theta(J) = \alpha_i^{\text{th}}$ for some $i \in \{0, 1\}$. We are now prepared to state properties on when J must send transitions:

Lemma 4.1. *Let J be a General Join with predecessor modules $\mathcal{Q} = \{A, B, \dots\}$. Then J sends the first transition at $t_{J,1}$ in $\tau^{\pm}_{PC} + \tau^{\pm}_{TH}$.*

Proof. According to (4.5), for all times t in $\mathcal{T}$,

$$\#\text{reqO}(t) = \#\text{rcvd}_{\text{reqO},s}(t).$$

Let d be the outgoing delay function of s. Then for $t < d(0)$, $\#\text{reqO}(t) = 0$ and for $t \geqslant d(0)$, $\#\text{reqO}(t) \geqslant 1$, i.e., J sends the first transition at time $d(0)$. Since $d(0)$ in $\tau^{\pm}_{PC} + \tau^{\pm}_{TH}$, the lemma follows. ■

Lemma 4.2. *Let J be a General Join with predecessor modules $\mathcal{Q} = \{A, B, \dots\}$. Wlog. assume $\theta(J) = \langle \alpha_0^{th}, \beta_1^{th} \rangle$. For all $k \geqslant 1$: if*

(i) J sends transition k at time $t_{J,k}$ and

(ii) there is a set $Q \subseteq \mathcal{Q}$, with $|Q| \geqslant \alpha$ such that J has received at least transition k from all X in Q by time $t_{Q,k}$,

then J sends transition $k+1$ by time $\max\{t_{J,k} + \tau^+_{loc}, t_{Q,k}\} + \tau^+_{Diff} + \tau^+_{PC} + \tau^+_{TH}$.

Proof. For all t, $\#l(t) = \#\text{reqO}(t)$. Because of assumption (i), for all $t < t_{J,k}$, $\#l(t) \leqslant k-1$ and for all $t \geqslant t_{J,k}$, $\#l(t) \geqslant k$. Let d be the outgoing delay function of l. Then for each X in $\mathcal{Q}$, port X^ℓ as well as port X'^ℓ make transition k at time $t' = d(t_{J,k})$. Due to the delay bounds of d,

$$t' \in t_{J,k} + \tau^\pm_{loc},$$

and therefore, for each X in $\mathcal{Q}$, port rcvd_{X,X^l} makes transition k at t'' with

$$t'' \in t_{J,k} + \tau^\pm_{loc} + \tau^\pm_{Diff}. \tag{4.9}$$

Because of assumption (ii), for all X in Q, port X^r makes transition k by time $t_{Q,k}$. Thus, for all times $t \geqslant t_{Q,k} + \tau^+_{Diff}$ and X in Q,

$$\text{rcvd}_{X,X^r}(t) \geqslant k. \tag{4.10}$$

From (4.9) and (4.10), we deduce, that port X' makes transition k by time t''', with

$$t''' = \max\{t_{J,k} + \tau^+_{loc}, t_{Q,k}\} + \tau^+_{Diff} \tag{4.11}$$

Further

$$\begin{aligned}\text{rcvd}_{s,th}(t''') &= \#th(t''') \\ &= \alpha^{\text{th}}\{\#X(t''') \mid X \in \mathcal{Q}\} \geqslant k,\end{aligned}$$

i.e., $\text{rcvd}_{s,th}$ makes transition k by time t'''.

Clearly, port s makes transition $k + M_0(s) = k+1$ by time t''', too, and thus reqO makes transition $k+1$ by time $t''' + \tau^+_{PC} + \tau^+_{TH}$. Combination with (4.11) yields the desired result and the lemma follows. ■

The following lemma's requirements are similar to Lemma 4.3, except that the lemma only holds for transitions greater or equal to 2, and requires to receive transition $k+1$ from at least β predecessor modules.

Lemma 4.3. *Let J be a General Join with predecessor modules $\mathcal{Q} = \{A, B, \dots\}$. Wlog. assume $\theta(J) = \langle \alpha^{th}_0, \beta^{th}_1 \rangle$. For all $k \geqslant 1$: if*

(i) J sends transition k at time $t_{J,k}$ and

(ii) there is a set $Q \subseteq \mathcal{Q}$, with $|Q| \geqslant \beta$, such that, J has received at least transition $k+1$ from all X in Q by time $t_{Q,k}$,

then J sends transition $k+1$ by time $\max\{t_{J,k} + \tau^+_{loc} + \tau^+_{Diff}, t_{Q,k} + \varepsilon\} + \tau^+_{PC} + \tau^+_{TH}$.

Proof. The proof follows the proof of Lemma 4.2 in large parts.

For all t in $\mathcal{T}$, $\#l(t) = \#\mathrm{reqO}(t)$. Let d be the outgoing delay function of l. Because of assumption (i), for each X in $\mathcal{Q}$, ports X^l and X'^l make transition k at time $t' = d(t_{J,k})$ and due to the delay bounds of d, $t' \in t_{J,k} + \tau^{\pm}_{loc}$. Thus, for each X in $\mathcal{Q}$, port rcvd_{X,X'^l} makes transition k at t' with

$$t' \in t_{J,k} + \tau^{\pm}_{loc} + \tau^{\pm}_{Diff}. \tag{4.12}$$

Because of assumption (ii), for all X in Q, port X'^r makes transition $k+1+M_0(X'^r) = k$ by time $t_{Q,k}$. Therefore, for times $t \geqslant t_{Q,k} + \varepsilon$ and X in Q, it holds that

$$\mathrm{rcvd}_{X,X'^r}(t) \geqslant k. \tag{4.13}$$

From (4.12) and (4.13), we deduce, that port X' makes transition k by time t''', with

$$t''' = \max\{t_{J,k} + \tau^{+}_{loc} + \tau^{+}_{Diff}, t_{Q,k} + \varepsilon\} \tag{4.14}$$

Further

$$\begin{aligned}\mathrm{rcvd}_{s,th'}(t''') &= \#th'(t''') \\ &= \beta^{\mathrm{th}}\{\#X'(t''') \mid X \in \mathcal{Q}\} \geqslant k,\end{aligned}$$

i.e., $\mathrm{rcvd}_{s,th'}$ makes transition k by time t'''.

Again, port s makes transition $k + M_0(s) = k + 1$ by time t''' and thus reqO makes transition $k+1$ by time $t''' + \tau^{+}_{PC} + \tau^{+}_{TH}$. Combination with (4.14) yields the desired result. The lemma follows. ■

Lemmata 4.2 and 4.3 state that a General Join J is able to "make progress" despite the failure of subset of its predecessor modules: both lemmata state that J only needs to receive transitions from a sufficiently large subset Q of predecessor modules to send transitions.

We will next derive a necessary condition on the transitions received by J for sending transition $k \geqslant 2$:

Lemma 4.4. *Let J be a General Join with predecessor modules $\mathcal{Q} = \{A, B, \dots\}$. Wlog. assume $\theta(J) = \langle \alpha^{th}_0, \beta^{th}_1 \rangle$. For all t in $\mathcal{T}$: if $\#reqO(t) = k + 1 \geqslant 2$, then J has sent transition k at time $t_{J,k}$ with*

$$t_{J,k} \leqslant t_{J,k+1} - \tau^{-}_{TH} - \tau^{-}_{PC} - \tau^{-}_{Diff} - \tau^{-}_{loc} \tag{4.15}$$

and either

(i) there exists a set $Q \subseteq \mathcal{Q}$ of size $|Q| \geqslant \alpha$ such that for $t' := t - \tau^{-}_{TH} - \tau^{-}_{PC} - \tau^{-}_{Diff}$:

$$\forall X \in Q : \#reqX(t') \geqslant k \tag{4.16}$$

(ii) or there exists a set $Q \subseteq M$ *of size* $|Q| \geqslant \beta$ *such that for* $t' := t - \tau^-_{TH} - \tau^-_{PC}$*:*

$$\forall X \in Q : \#reqX(t') \geqslant k+1. \tag{4.17}$$

Proof. Since $\#\mathrm{reqO}(t) \geqslant k+1$, and $\#s(0) = 1$,

$$\#s(t - \tau^-_{PC} - \tau^-_{TH}) \geqslant k+1. \tag{4.18}$$

must hold. Further (4.18) only holds if either (i)

$$\alpha^{\mathrm{th}}\{\#X(t - \tau^-_{PC} - \tau^-_{TH}) \mid X \in \mathcal{Q}\} \geqslant k$$

or (ii)

$$\beta^{\mathrm{th}}\{\#X'(t - \tau^-_{PC} - \tau^-_{TH}) \mid X \in \mathcal{Q}\} \geqslant k.$$

We consider both cases one after the other:

(i) Then there exists a set $Q \subseteq X$ of size $|Q| \geqslant \alpha$, such that, for all X in Q, $\#X(t - \tau^-_{PC} - \tau^-_{TH}) \geqslant k$. Since $\#X(t - \tau^-_{PC} - \tau^-_{TH}) \leqslant \#\mathrm{reqX}(t - \tau^-_{PC} - \tau^-_{TH} - \tau^-_{Diff})$, (4.16) follows. Since further $\#X(t - \tau^-_{PC} - \tau^-_{TH}) \leqslant \#\mathrm{reqO}(t - \tau^-_{PC} - \tau^-_{TH} - \tau^-_{Diff} - \tau^-_{loc})$, (4.15) follows. The lemma follows.

(ii) Thus there exists a set $Q \subseteq X$ of size $|Q| \geqslant \beta$, such that, for all X in Q, $\#X'(t - \tau^-_{PC} - \tau^-_{TH}) \geqslant k$. Since $\#X'(t - \tau^-_{PC} - \tau^-_{TH}) + 1 \leqslant \#\mathrm{reqX}(t - \tau^-_{PC} - \tau^-_{TH} - \varepsilon)$, (4.17) follows. Since, further $\#X'(t - \tau^-_{PC} - \tau^-_{TH}) \leqslant \#\mathrm{reqO}(t - \tau^-_{PC} - \tau^-_{TH} - \tau^-_{Diff} - \tau^-_{loc})$, (4.15) follows. The lemma follows.

The lemma follows in both cases. ■

From Lemma 4.4 (4.15), there follows a minimum time between two successive transitions sent by a General Join:

Corollary 4.1. *If General Join* J *sends transition* k*,* $k \geqslant 1$*, at time* $t_{J,k}$ *and transition* $k+1$ *at time* $t_{J,k+1}$*, then* $t_{J,k+1} - t_{J,k} \geqslant \tau^-_{TH} + \tau^-_{PC} + \tau^-_{Diff} + \tau^-_{loc}$*.*

4.5 Implementing General Joins

In the previous sections, a high level view has been introduced, which allows to model the control structure of on-chip algorithms at the level of data flows. The key component herein is the fault-tolerant join, called General Join module, for which the deadlock of one of its preceding input modules does not necessarily lead to a deadlock of the join output. Interestingly threshold transitions introduced in Section 4.3 can be used to formally express join behavior, which does not necessarily follow a "wait-for-all" semantics and still keeps the synchrony between the join's successor module and all the join's predecessor modules.

Given the importance of General Joins in fault-tolerant clockless circuits, the question arises how to physically implement them. For this purpose we make use of the ideas presented in Chapter 3: we will discuss an implementation of a General Join module J by a compound module $GJ_{\text{Imp}}(J)$ comprising less complex sub-modules. For a physical implementation, it thus remains to implement the less complex $GJ_{\text{Imp}}(J)$. In order not to hide the key message, we slightly modify the General Join module J: we discard J's AckX output ports for all X in J's predecessor modules $\mathcal{Q}$. Further, for each X in $\mathcal{Q}$, we attach a so called *input channel*, that is a channel with input port b_X, output port ReqX, delay within $\left[\tau^-_{rem}, \tau^+_{rem}\right]$ and initial value 0 to each of J's inputs. Port b_X is the output port of predecessor module X.

In the previous sections of this chapter we only investigated the behavior of correct modules. Here the analysis is extended to the behavior of a $GJ_{\text{Imp}}(J)$ module in the presence of faulty predecessor modules. We will thus explicitly state whether a predecessor module is assumed to be correct or not. In case a predecessor module is faulty, we do not restrict the behavior of the corresponding input channel, i.e., the channel may produce an unconstrained series of events at its output. We only require that its counting function is well-defined for all $t \in \mathcal{T}$.

We start by formally specifying the architecture of a module $GJ_{\text{Imp}}(J)$ for a given General Join module J, i.e., stating $GJ_{\text{Imp}}(J)$'s sub-modules (including their behavior) and interconnect. It is important to note that the behavioral properties of the sub-modules defined in this subsection are *assumed* properties, i.e., basic properties that must a priori be guaranteed by the implementation of the sub-modules.

Again we will attach *input channels* with delay in $\left[\tau^-_{rem}, \tau^+_{rem}\right]$ and initial value 0 to each of $GJ_{\text{Imp}}(J)$'s input ports. It is then shown in Section 4.6 that $GJ_{\text{Imp}}(J)$ with input channels implements J with input channels, given that certain constraints hold.

4.5.1 Specification of the compound module GJ_{Imp}

Consider a General Join module J with a set of m predecessor modules $\mathcal{Q} = \{q, r, \dots\}$. Wlog. assume that $\theta(J) = \langle \alpha^{\text{th}}_0, \beta^{\text{th}}_1 \rangle$. We will next define a $GJ_{\text{Imp}}(J)$ module. In the following let p be a $GJ_{\text{Imp}}(J)$ module. Further let $t_{p,b} \in \mathcal{T}$ be $p's$ booting completion time. Figure 4.16 shows p's architecture which has been inspired by the architecture of our DARTS fault-tolerant tick generation approach [41]. It consists of one $+/-$ counter module per predecessor module (only two are depicted), four Threshold modules, two of which are implementing the α^{th}_0 rule and the other two implementing the β^{th}_1 rule, and a request broadcast module that finally generates p's output request transitions $\#b_p(t)$. Every $+/-$ counter is refined into several additional sub-modules: A pair of elastic pipes (remote pipe, local pipe) that form FIFO buffers for (remote, local) transitions, a Diff-Gate module that removes matching remote and local transitions from the pipes, and a PCSG module that generates the status signals reflecting the difference of the number of transitions present in the local and the remote pipe.

We will now specify the ports and the behavior of all these modules in detail.

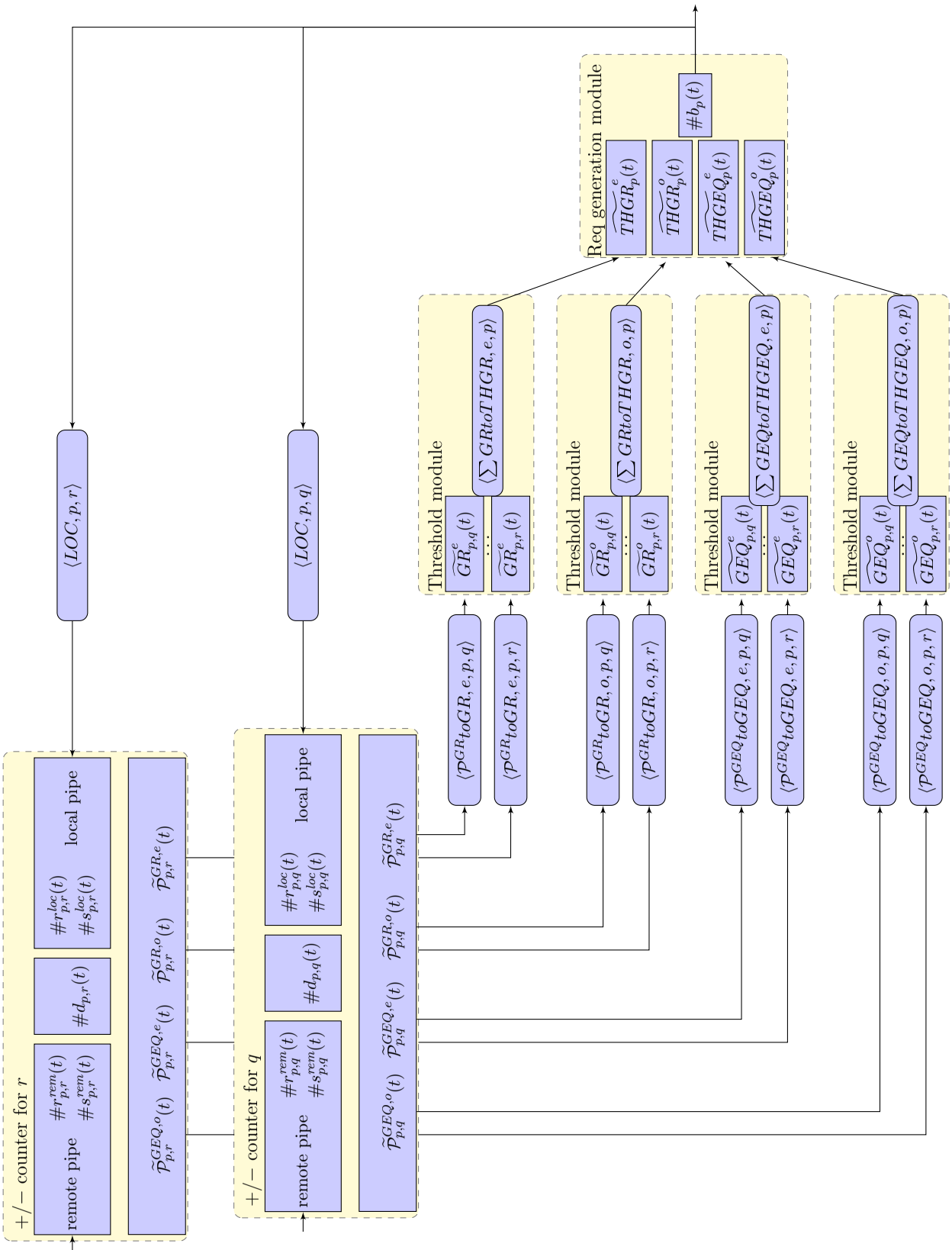

Figure 4.16: Architecture of the GJ_{Imp} module

4.5.1.1 Pairs of elastic pipes

Module p incorporates m pairs of unbounded queues, each of which corresponds to a single $q \in \mathcal{Q}$. We will denote the pair of pipes at p corresponding to q by (p, q) in the sequel. (p, q) consists of a remote pipeline that stores transitions sent by q, and a local pipeline that stores transitions sent by p locally. Since our intention is to physically implement the pair of pipes, we will have to replace the pipepair by bounded queues as soon as we know their maximum attained fill level. For this purpose we denote the maximum attainable fill level of a remote respectively local pipe by S_{rem} respectively S_{loc}. Note that the numbers S_{rem} and S_{loc} depend on the circuit the General Join is used in, as well as its delays. However, as soon as we manage to prove that the bounds S_{rem} and S_{loc} hold, we can substitute the unbounded queue pair by a bounded queue pair with the respective bounds without changing the overall behavior.

The local pipe in (p, q) has a single (req) input port that is fed by p's local transitions, i.e., $\#b_p(t)$, supplied via the channel $\langle LOC, p, q\rangle$, and a single (req) output port represented by the counting function $\#r^{loc}_{p,q}(t)$. Similarly, the remote pipe in (p, q) has a single (req) input port that is fed by q's transitions, and a single output port represented by the counting function $\#r^{rem}_{p,q}(t)$.

We say that *p receives transition k from q at time t*, or, equivalently, that *transition k is received at the remote pipe of (p, q) at time t*, iff $r^{rem}_{p,q}$ makes transition k at time t. Analogously, we say that *transition k is received at the local pipe of (p, q) at time t*, iff $r^{loc}_{p,q}$ makes transition k at time t.

Behavioral description: Both pipes in the pair (p, q) have the behavior of a zero-delay queue: For any $t \geqslant 0$, $\#r^{loc}_{p,q}(t)$ respectively $\#r^{rem}_{p,q}(t)$ counts the number of transitions received at the local respectively remote pipe of (p, q) until time t. Upon reset, both pipes are pre-filled with the virtual transition 0 (such a transition is called virtual since it has never been sent). Formally, for each $t \in [0, t_{p,b}]$, it holds that $\#r^{rem}_{p,q}(t_{p,b}) = \#r^{loc}_{p,q}(t_{p,b}) := 0$.

4.5.1.2 Diff-Gate module

To avoid pipes with infinite capacity, each pair of pipes is equipped with a special Diff-Gate circuit that removes matching transitions, i.e., transitions received in both pipes. The Diff-Gate for (p, q) has two input ports connected to $\#r^{rem}_{p,q}(t)$ and $\#r^{loc}_{p,q}(t)$, and a single output port represented by the counting function $\#d_{p,q}(t)$, which gives the largest transition number that has been removed from both the remote and local pipe of (p, q) by time t. To formalize the removal of the virtual transition 0, the initial value is $\#d_{p,q}(t_{p,b}) = -1$. Formally, we say that *transition $k \geqslant 0$ is removed from (p, q) at time t* iff port $d_{p,q}$ makes transition k at time t.

Behavioral description: Recall that virtual transition 0 is received in all local and remote pipes at booting completion time $t_{loc,0} = t_{p,b}$. Transitions are removed from pairs of pipes as follows:

- $k = 0$: If
  - transition $k+1 = 1$ is received at the remote pipe of (p, q) at time $t_{rem,k+1}$, and
  - transition $k+1 = 1$ is received at the local pipe of (p, q) at time $t_{loc,k+1}$,

  then transition $k = 0$ is removed from (p, q) at $t_{rmv,k}$, with
  $$t_{rmv,k} \in \max\{t_{rem,k+1}, t_{loc,k+1}\} + \left[\tau^-_{Diff}, \tau^+_{Diff}\right].$$
- $k \geqslant 1$: If
  - transition $k+1$ is received at the remote pipe of (p, q) at time $t_{rem,k+1}$, and
  - transition $k+1$ is received at the local pipe of (p, q) at time $t_{loc,k+1}$, and
  - transition $k-1$ has been removed from (p, q) at time $t_{rmv,k-1}$,

  then transition k is removed from (p, q) at $t_{rmv,k}$, with
  $$t_{rmv,k} \in \max\{t_{rem,k+1}, t_{loc,k+1}, t_{rmv,k-1}\} + \left[\tau^-_{Diff}, \tau^+_{Diff}\right].$$

On top of $\#r^{rem}_{p,q}(t)$, $\#r^{loc}_{p,q}(t)$ and $\#d_{p,q}(t)$, we define the actual fill size of the local and remote pipe of (p, q) at time t as

$$\#s^{loc}_{p,q}(t) := \#r^{loc}_{p,q}(t) - \#d_{p,q}(t)$$
$$\#s^{rem}_{p,q}(t) := \#r^{rem}_{p,q}(t) - \#d_{p,q}(t).$$

4.5.1.3 Pipe Compare Signal Generator (PCSG) module

The ports provided by the pair of pipes (p, q) and its Diff-Gate are connected to the PCSG, which generates four status ports $\widetilde{\mathcal{P}}^{GEQ,o}_{p,q}(t)$, $\widetilde{\mathcal{P}}^{GEQ,e}_{p,q}(t)$, $\widetilde{\mathcal{P}}^{GR,o}_{p,q}(t)$ and $\widetilde{\mathcal{P}}^{GR,e}_{p,q}(t)$ that characterize the difference of the number of transitions stored in the remote and local pipes by time t. Different ports are provided for odd and even transitions. For example, $\widetilde{\mathcal{P}}^{GEQ,o}_{p,q}(t)$ signals when the number of transitions in the remote pipe of (p, q) is greater or equal than the number of transitions in the local pipe of (p, q), provided that the last transition that was received in the local pipe of (p, q) was odd; $\widetilde{\mathcal{P}}^{GR,o}_{p,q}(t)$ does the same for "greater" replacing "greater or equal".

All these signals are fed, via dedicated channels that add some delay, to the Threshold modules of the General Join Module p.

Behavioral description: The status signals generated by the PCSG associated with (p, q) must satisfy the following properties: for all q in $\mathcal{Q}$ and all t in $\mathcal{T}$,

$$\widetilde{\mathcal{P}}^{GEQ,o}_{p,q}(t) := [\#r^{rem}_{p,q}(t) \geqslant \#r^{loc}_{p,q}(t)] \wedge [\#r^{loc}_{p,q}(t) \in \{1, 3, \dots\}] \wedge [\#s^{loc}_{p,q}(t) = 1]$$
$$\widetilde{\mathcal{P}}^{GEQ,e}_{p,q}(t) := [\#r^{rem}_{p,q}(t) \geqslant \#r^{loc}_{p,q}(t)] \wedge [\#r^{loc}_{p,q}(t) \in \{0, 2, \dots\}] \wedge [\#s^{loc}_{p,q}(t) = 1]$$
$$\widetilde{\mathcal{P}}^{GR,o}_{p,q}(t) := [\#r^{rem}_{p,q}(t) > \#r^{loc}_{p,q}(t)] \wedge [\#r^{loc}_{p,q}(t) \in \{1, 3, \dots\}] \wedge [\#s^{loc}_{p,q}(t) = 1]$$
$$\widetilde{\mathcal{P}}^{GR,e}_{p,q}(t) := [\#r^{rem}_{p,q}(t) > \#r^{loc}_{p,q}(t)] \wedge [\#r^{loc}_{p,q}(t) \in \{0, 2, \dots\}] \wedge [\#s^{loc}_{p,q}(t) = 1]$$

Note that these signals need to be valid only if the local pipes contain exactly one transition ($\#s^{loc}_{p,q}(t) = 1$), which makes it easier for a physical implementation to fulfill these properties.

The above signals are fed into four dedicated channels that connect the PCSG with the Threshold modules: for all q in $\mathcal{Q}$, there exist the following channels, all of which are initialized to 0:

- Channel $\langle \mathcal{P}^{GEQ}toGEQ, o, p, q\rangle$ with input $\widetilde{\mathcal{P}}^{GEQ,o}_{p,q}(t)$, output $\widetilde{GEQ}^{o}_{p,q}(t)$ and delay in $\left[\tau^{-}_{PC}, \tau^{+}_{PC}\right]$.
- Channel $\langle \mathcal{P}^{GEQ}toGEQ, e, p, q\rangle$ with input $\widetilde{\mathcal{P}}^{GEQ,e}_{p,q}(t)$, output $\widetilde{GEQ}^{e}_{p,q}(t)$ and delay in $\left[\tau^{-}_{PC}, \tau^{+}_{PC}\right]$.
- Channel $\langle \mathcal{P}^{GR}toGR, o, p, q\rangle$ with input $\widetilde{\mathcal{P}}^{GR,o}_{p,q}(t)$, output $\widetilde{GR}^{o}_{p,q}(t)$ and delay in $\left[\tau^{-}_{PC}, \tau^{+}_{PC}\right]$.
- Channel $\langle \mathcal{P}^{GR}toGR, e, p, q\rangle$ with input $\widetilde{\mathcal{P}}^{GR,e}_{p,q}(t)$, output $\widetilde{GR}^{e}_{p,q}(t)$ and delay in $\left[\tau^{-}_{PC}, \tau^{+}_{PC}\right]$.

4.5.1.4 Threshold modules

$\widetilde{GEQ}^{o/e}_{p,q}(t)$ and $\widetilde{GR}^{o/e}_{p,q}(t)$ are further processed at four Threshold modules: If the number of active $\widetilde{GEQ}^{o/e}_{p,q}(t)$ receptively $\widetilde{GR}^{o/e}_{p,q}(t)$ signals exceeds the α respectively β threshold, the corresponding threshold signal $\widetilde{THGEQ}^{o/e}_{p}(t)$ respectively $\widetilde{THGR}^{o/e}_{p}(t)$ becomes active within $\left[\tau^{-}_{TH}, \tau^{+}_{TH}\right]$. This property will be formalized below as a logical predicate [which is a function of time here] involving the sum of the status functions of a Threshold module's input ports, which is fed into a channel.

Behavioral description: The Threshold modules are modeled as Boolean function modules with channels at their outputs: for each q in $\mathcal{Q}$ there are the following Boolean function modules and channels, all of which are initialized to 0:

- Channel $\langle \sum GEQtoTHGEQ, o, p\rangle$ with input $\sum_{q\in Q} \widetilde{GEQ}^{o}_{p,q}(t) \geqslant \alpha$, output $\widetilde{THGEQ}^{o}_{p}(t)$ and delay in $\left[\tau^{-}_{TH}, \tau^{+}_{TH}\right]$.
- Channel $\langle \sum GEQtoTHGEQ, e, p\rangle$ with input $\sum_{q\in Q} \widetilde{GEQ}^{e}_{p,q}(t) \geqslant \alpha$, output $\widetilde{THGEQ}^{e}_{p}(t)$ and delay in $\left[\tau^{-}_{TH}, \tau^{+}_{TH}\right]$.
- Channel $\langle \sum GRtoTHGR, o, p\rangle$ with input $\sum_{q\in Q} \widetilde{GR}^{o}_{p,q}(t) \geqslant \beta$, output $\widetilde{THGR}^{o}_{p}(t)$ and delay in $\left[\tau^{-}_{TH}, \tau^{+}_{TH}\right]$.
- Channel $\langle \sum GRtoTHGR, e, p\rangle$ with input $\sum_{q\in Q} \widetilde{GR}^{e}_{p,q}(t) \geqslant \beta$, output $\widetilde{THGR}^{e}_{p}(t)$ and delay in $\left[\tau^{-}_{TH}, \tau^{+}_{TH}\right]$.

4.5.1.5 Req generation module

Module p sends the next transition at time t, when (i) both threshold outputs for the previously generated transition, say, $\widetilde{THGEQ}_p^o(t)$ and $\widetilde{THGR}_p^o(t)$, are inactive again, and (ii) at least one threshold output $\widetilde{THGEQ}_p^e(t)$ or $\widetilde{THGR}_p^e(t)$ for the current transition becomes active. We will refer to (i) as the *disabling path* and to (ii) as the *enabling path* in the sequel. The Req generation module hence has four input ports connected to the threshold output ports, and a single output port represented by the counting function $\#b_p(t)$, which is the number of transitions broadcast by p by time t. Finally, $\#b_p(t)$ is distributed to the local pipe in (p,q) at p for all $q \in \mathcal{Q}$, via dedicated channels $\langle LOC, p, q\rangle$, and to the successor module.

Behavioral description: Let port b_p be defined by its event trace $\widehat{b}_p$ as the set for which

$$(0,0) \in \widehat{b}_p$$

and for all t in $\mathcal{T}$,

$$(\widetilde{THGEQ}_p^o(t) \vee \widetilde{THGR}_p^o(t)) \wedge \neg(\widetilde{THGEQ}_p^e(t) \vee \widetilde{THGR}_p^e(t)) \Leftrightarrow (t,0) \in \widehat{b}_p$$
$$(\widetilde{THGEQ}_p^e(t) \vee \widetilde{THGR}_p^e(t)) \wedge \neg(\widetilde{THGEQ}_p^o(t) \vee \widetilde{THGR}_p^o(t)) \Leftrightarrow (t,1) \in \widehat{b}_p.$$

Then, $\#b_p(t)$ is the counting function of b_p, with initial value $\#b_p(0) = 0$.

4.6 Correctness Proofs

In the previous section we have formally defined module $GJ_{\mathrm{Imp}}(J)$, where J is a General Join module with threshold $\Theta(p) = \langle \alpha_0^{\mathrm{th}}, \beta_1^{\mathrm{th}}\rangle$. It remains to formally prove that $GJ_{Imp}(J)$ with input channels implements module J with input channels. For this purpose we derive properties for $GJ_{\mathrm{Imp}}(J)$ analogous to the properties obtained for the General Join module in Section 4.4.1.3.

For ease of reading we denote the delay function d of a channel Ch in module $GJ_{\mathrm{Imp}}(J)$ with $d(Ch;\cdot)$. For example, the value of some channel $\langle \mathcal{P}^{GR}toGR, o, p, q\rangle$'s delay function at time $t \in \mathcal{T}$ is $d(\langle \mathcal{P}^{GR}toGR, o, p, q\rangle; t)$ and the value of the inverse delay function at time t is $d^{-1}(\langle \mathcal{P}^{GR}toGR, o, p, q\rangle; t)$.

We start our detailed treatment with the technical Lemma 4.5, which asserts a certain persistence of the number of transitions present in the local and remote pipe for a certain time.

Lemma 4.5. *Let p be a correct $GJ_{Imp}(J)$ module. If, for a correct predecessor module $q \in \mathcal{Q}$, at time t it holds that*

$$(\#r_{p,q}^{loc}(t) = k) \wedge (\#s_{p,q}^{loc}(t) = 1)$$

for some $k \geqslant 1$, then it must hold that

$$\#r^{loc}_{p,q}(t - \tau^-_{Diff}) \geqslant k \text{ and}$$
$$\#r^{rem}_{p,q}(t - \tau^-_{Diff}) \geqslant k.$$

Proof.

$$(\#r^{loc}_{p,q}(t) = k) \wedge (\#s^{loc}_{p,q}(t) = 1) \equiv (\#r^{loc}_{p,q}(t) = k) \wedge (\#r^{loc}_{p,q}(t) - \#d_{p,q}(t) = 1)$$
$$\Rightarrow \#d_{p,q}(t) = k - 1 \tag{4.19}$$

Let $t_{rmv,k-1}$ be the time at that transition $k-1$ is removed from the pipepair (p, q). From (4.19) it follows that

$$t_{rmv,k-1} \leqslant t. \tag{4.20}$$

Now assume by contradiction that

$$\#r^{loc}_{p,q}(t - \tau^-_{Diff}) < k \text{ or}$$
$$\#r^{rem}_{p,q}(t - \tau^-_{Diff}) < k. \tag{4.21}$$

Denoting by $t_{loc,k}$ (respectively $t_{rem,k}$) the time at that transition k is received in the local (respectively remote) pipe of (p, q), it follows that

$$t_{loc,k} > t - \tau^-_{Diff} \text{ resp.} \tag{4.22}$$
$$t_{rem,k} > t - \tau^-_{Diff}. \tag{4.23}$$

Combination of (4.20) with (4.22) (respectively (4.23)) yields

$$t_{rmv,k-1} < t_{loc,k} + \tau^-_{Diff} \text{ resp.} \tag{4.24}$$
$$t_{rmv,k-1} < t_{rem,k} + \tau^-_{Diff}. \tag{4.25}$$

From the behavioral specification of the Diff-gate, however, we know that

$$t_{rmv,k-1} \geqslant t_{loc,k} + \tau^-_{Diff} \text{ and}$$
$$t_{rmv,k-1} \geqslant t_{rem,k} + \tau^-_{Diff},$$

contradicting (4.24) and (4.25). ■

We next establish a main result, the Interlocking Lemma 4.6, which states that an "old" transition $k-2, k-4$, etc. is never falsely interpreted as a "new" transition k in the GR and GEQ rules of the algorithm. This is not immediately evident: An even transition $k+1$ is generated by either $\widetilde{THGEQ}^o$ or $\widetilde{THGR}^o$ being active (depending on whether the GEQ or the GR rule triggered the broadcast); assume that it was triggered by the $\widetilde{THGEQ}^o$ signal. $\widetilde{THGEQ}^o$ is only enabled, if enough (at least α) $\widetilde{GEQ}^o$ signals are active. For all of

these signals, it must hold that the responsible $+/-$ counter has received an even number of transitions locally, that is, any $\widetilde{GEQ}^o$ may be based on local transition k, $k-2$ or even earlier. We, however, require transition $k+1$ to depend solely on $\widetilde{GEQ}^o$ signals based on transition k only.

The Interlocking Lemma will require that all the local loop delays in p are within a factor 2 of each other, which is expressed formally in Constraint 1.

Constraint 1. [C1] (Interlocking Constraint). *With the abbreviations*

$$\begin{aligned} T_{max} &:= \tau^+_{TH} + \tau^+_{PC} + \tau^+_{loc} \\ T_{min} &:= \tau^-_{TH} + \tau^-_{PC} + \tau^-_{loc} + \tau^-_{Diff} \\ T_{min,dis} &:= \tau^-_{TH} + \tau^-_{PC} + \tau^-_{loc}, \end{aligned} \tag{4.26}$$

the relation $T_{max} \leqslant T_{min} + T_{min,dis}$ must hold.

Lemma 4.6 (**C1** Interlocking Lemma)**.** *Let p be a correct $GJ_{Imp}(J)$ module. For all times t in $\mathcal{T}$: if $\#b_p(t) = k+1 \geqslant 2$, then*

(i) either there exists a set Q of size $|Q| \geqslant \alpha$ such that for $t' := t - \tau^-_{TH} - \tau^-_{PC}$:

$$\begin{aligned} k \in \{0,2,\dots\} &\Rightarrow \forall q \in Q : \exists t_q \leqslant t' : \widetilde{\mathcal{P}}^{GEQ,e}_{p,q}(t_q) \wedge \#r^{loc}_{p,q}(t_q) \geqslant k \\ k \in \{1,3,\dots\} &\Rightarrow \forall q \in Q : \exists t_q \leqslant t' : \widetilde{\mathcal{P}}^{GEQ,o}_{p,q}(t_q) \wedge \#r^{loc}_{p,q}(t_q) \geqslant k \end{aligned}$$

(ii) or there exists a set Q of size $|Q| \geqslant \beta$ such that for $t' := t - \tau^-_{TH} - \tau^-_{PC}$:

$$\begin{aligned} k \in \{0,2,\dots\} &\Rightarrow \forall q \in Q : \exists t_q \leqslant t' : \widetilde{\mathcal{P}}^{GR,e}_{p,q}(t_q) \wedge \#r^{loc}_{p,q}(t_q) \geqslant k \\ k \in \{1,3,\dots\} &\Rightarrow \forall q \in Q : \exists t_q \leqslant t' : \widetilde{\mathcal{P}}^{GR,o}_{p,q}(t_q) \wedge \#r^{loc}_{p,q}(t_q) \geqslant k \end{aligned}$$

Proof. The proof is by induction on $k+1 \geqslant 2$, the number of transitions sent by p until time t. To carry out the induction step, we will prove a slightly stronger lemma, where in case of (i) we require

$$\begin{aligned} t_q &= d^{-1}(\langle \mathcal{P}^{GEQ} toGEQ, e, p, q\rangle; d^{-1}(\langle \sum GEQtoTHGEQ, e, p\rangle; t_{p,k})) \text{ for } k \in \{0,2,\dots\}, \\ t_q &= d^{-1}(\langle \mathcal{P}^{GEQ} toGEQ, o, p, q\rangle; d^{-1}(\langle \sum GEQtoTHGEQ, o, p\rangle; t_{p,k})) \text{ for } k \in \{1,3,\dots\}, \end{aligned}$$

and in case of (ii) we require

$$\begin{aligned} t_q &= d^{-1}(\langle \mathcal{P}^{GR} toGR, e, p, q\rangle; d^{-1}(\langle \sum GRtoTHGR, e, p\rangle; t_{p,k})) \text{ for } k \in \{0,2,\dots\}, \\ t_q &= d^{-1}(\langle \mathcal{P}^{GR} toGR, o, p, q\rangle; d^{-1}(\langle \sum GRtoTHGR, o, p\rangle; t_{p,k})) \text{ for } k \in \{1,3,\dots\}. \end{aligned}$$

The lemma is first shown for transition $k+1 = 2$. Then we assume that some transition $k+1 > 2$ is the first transition for which the lemma does not hold. By investigating

the cause which triggered the sending of this transition, we obtain a contradiction to Constraint 1.

Begin $(k+1=2)$**:** Assume p sends transition 2 at time $t_{p,2}$. Then, by the module specification (specification of the Req generation module), either (a) $\widetilde{THGEQ}_p^o(t_{p,2})$ or (b) $\widetilde{THGR}_p^o(t_{p,2})$ must have held. We consider both cases:

case (a): If $\widetilde{THGEQ}_p^o(t_{p,2})$, then by the module specification (specification of the Threshold modules), there must be a set $Q \subseteq \mathcal{Q}$, of size $|Q| \geqslant \alpha$, such that, for time $t' := d^{-1}(\langle \sum GEQtoTHGEQ, o, p\rangle; t_{p,2})$

$$\forall q \in Q : \widetilde{GEQ}_{p,q}^o(t').$$

Again by the module specification (specification of the PCSG to Threshold module channels), and with the time $t_q := d^{-1}(\langle \mathcal{P}^{GEQ}toGEQ, o, p, q\rangle; t')$ defined for every $q \in Q$, we obtain

$$\forall q \in Q : \widetilde{\mathcal{P}}_{p,q}^{GEQ,o}(t_q)$$

and by this

$$\widetilde{\mathcal{P}}_{p,q}^{GEQ,o}(t_q) \equiv [\#r_{p,q}^{rem}(t_q) \geqslant \#r_{p,q}^{loc}(t_q)] \wedge [\#r_{p,q}^{loc}(t_q) \in \{1,3,\dots\}] \wedge [\#s_{p,q}^{loc}(t_q) = 1]. \quad (4.27)$$

Since $\#r_{p,q}^{loc}(t_q) \geqslant 0$ from reset on and $\#r_{p,q}^{loc}(t_q) \in \{1,3,\dots\}$,

$$\#r_{p,q}^{loc}(t_q) \geqslant 1 = k. \quad (4.28)$$

Finally, from the channel properties, we know that

$$\begin{aligned} t_{p,2} - t_q &= t_{p,2} - d^{-1}(\langle \mathcal{P}^{GEQ}toGEQ, o, p, q\rangle; d^{-1}(\langle \sum GEQtoTHGEQ, o, p\rangle; t_{p,2})) \\ &\geqslant \tau_{TH}^- + \tau_{PC}^-. \end{aligned}$$

case (b): If $\widetilde{THGR}_p^o(t_{p,2})$, then, by the module specification (specification of the Threshold modules), there must be a set $Q \subseteq P \setminus \{p\}$, of size $|Q| \geqslant \beta$, such that, for $t' := d^{-1}(\langle \sum GRtoTHGR, o, p\rangle; t_{p,2})$

$$\forall q \in Q : \widetilde{GR}_{p,q}^o(t').$$

By the module specification (specification of the PCSG to Threshold module channels), and with the point in time $t_q := d^{-1}(\langle \mathcal{P}^{GR}toGR, o, p, q\rangle; t')$ defined for every $q \in Q$, this implies

$$\forall q \in Q : \widetilde{\mathcal{P}}_{p,q}^{GR,o}(t_q) \quad (4.29)$$

and by this

$$\widetilde{\mathcal{P}}^{GR,o}_{p,q}(t_q) \equiv [\#r^{rem}_{p,q}(t_q) > \#r^{loc}_{p,q}(t_q)] \wedge [\#r^{loc}_{p,q}(t_q) \in \{1,3,\dots\}] \wedge [\#s^{loc}_{p,q}(t_q) = 1]. \quad (4.30)$$

Since $\#r^{loc}_{p,q}(t_q) \geqslant 0$ from reset on and $\#r^{loc}_{p,q}(t_q) \in \{1,3,\dots\}$,

$$\#r^{loc}_{p,q}(t_q) \geqslant 1 = k. \quad (4.31)$$

From the channel properties, we know that

$$\begin{aligned} t_{p,2} - t_q &= t_{p,2} - d^{-1}(\langle \mathcal{P}^{GR}toGR, o, p, q\rangle; d^{-1}(\langle \sum GRtoTHGR, o, p\rangle; t_{p,2})) \\ &\geqslant \tau^-_{TH} + \tau^-_{PC}. \end{aligned}$$

Step ($k+1 \geqslant 3$): Assume by contradiction that $k+1$ is the first transition for which the lemma does not hold. Let $t_{p,k+1}$ be the time p issues transition $k+1$. Assume wlog. that $k \in \{1,3,\dots\}$[4]. We will establish two delay bounds, one on the enabling path and the other on the disabling path.

Enabling path: To send transition $k+1$ at time $t_{p,k+1}$, by the module specification (specification of the Req generation module), at least one of the two threshold signals must have been enabled, i.e., (a) $\widetilde{THGEQ}^{o}_{p}(t_{p,k+1})$ or (b) $\widetilde{THGR}^{o}_{p}(t_{p,k+1})$ must have held. We consider both cases:

case (a): If $\widetilde{THGEQ}^{o}_{p}(t_{p,k+1})$, then, by the algorithm (specification of the Threshold modules), there must be a set $Q \subseteq \mathcal{Q}$, of size $|Q| \geqslant \alpha$, such that, for time $t' := d^{-1}(\langle \sum GEQtoTHGEQ, o, p\rangle; t_{p,k+1})$

$$\forall q \in Q : \widetilde{GEQ}^{o}_{p,q}(t'). \quad (4.32)$$

Again by the algorithm (specification of the PCSG to Threshold module channels), and with the time $t_q := d^{-1}(\langle \mathcal{P}^{GEQ}toGEQ, o, p, q\rangle; t')$ defined for every $q \in Q$, this implies

$$\forall q \in Q : \widetilde{\mathcal{P}}^{GEQ,o}_{p,q}(t_q) \quad (4.33)$$

and by this

$$\widetilde{\mathcal{P}}^{GEQ,o}_{p,q}(t_q) \equiv [\#r^{rem}_{p,q}(t_q) \geqslant \#r^{loc}_{p,q}(t_q)] \wedge [\#r^{loc}_{p,q}(t_q) \in \{1,3,\dots\}] \wedge [\#s^{loc}_{p,q}(t_q) = 1]. \quad (4.34)$$

By the channel properties

$$\tau^-_{TH} + \tau^-_{PC} \leqslant t_{p,k+1} - t_q \leqslant \tau^+_{TH} + \tau^+_{PC}. \quad (4.35)$$

[4]The proof for $k \in \{0,2,\dots\}$ is analogous.

Assuming that $\forall q \in Q : \#r^{loc}_{p,q}(t_q) \geqslant k$ yields the desired result of the Lemma. Thus we only have to investigate the negation:

$$\exists q \in Q : \#r^{loc}_{p,q}(t_q) < k.$$

Since, by (4.34), $\#r^{loc}_{p,q}(t_q) \in \{1, 3, \dots\}$, we obtain

$$\exists q \in Q : \#r^{loc}_{p,q}(t_q) \leqslant k - 2.$$

Thus, transition $k-1$ must be received in the local pipe of pipepair (p, q) at time $t_{rcv,k-1}$, with

$$t_{rcv,k-1} > t_q.$$

The combination with (4.35) yields

$$t_{p,k+1} - t_{rcv,k-1} < \tau^+_{TH} + \tau^+_{PC}. \tag{4.36}$$

Let $t_{p,k-1}$ be the sending time of transition $k-1$. Clearly, by the local channel properties, we arrive at

$$\begin{aligned} &t_{p,k-1} \geqslant t_{rcv,k-1} - \tau^+_{loc} \quad \Rightarrow \\ &t_{p,k+1} - t_{p,k-1} < \tau^+_{TH} + \tau^+_{PC} + \tau^+_{loc}. \end{aligned} \tag{4.37}$$

Before proceeding further, we will handle the disabling path.

Disabling path: Let $t_{p,k}$ be the sending time of transition k. By the induction hypothesis, we know that the lemma holds for transition k. According to the lemma, we have to distinguish two cases (i) and (ii):

case (a.i): Assume that implication (i) is valid, i.e., there exists a set Q' of size $|Q'| \geqslant \alpha$, such that, for

$$t_{q'} := d^{-1}(\langle \mathcal{P}^{GEQ} toGEQ, e, p, q' \rangle; d^{-1}(\langle \sum GEQtoTHGEQ, e, p \rangle; t_{p,k}))$$

it holds that

$$\forall q' \in Q' : \widetilde{\mathcal{P}}^{GEQ,e}_{p,q'}(t_{q'}) \wedge \#r^{loc}_{p,q'}(t_{q'}) \geqslant k - 1. \tag{4.38}$$

Thus, transition $k-1$ must have been received in the local pipe of (p, q') at time $t_{q',rcv,k-1}$, with

$$t_{q',rcv,k-1} \leqslant t_{q'} - \tau^-_{Diff}$$

by Lemma 4.5. Furthermore, by the properties of the local channels,

$$t_{q',rcv,k-1} - t_{p,k-1} \geqslant \tau^-_{loc}.$$

Thus, we find

$$t_{p,k} - t_{p,k-1} \geqslant \tau^-_{TH} + \tau^-_{PC} + \tau^-_{Diff} + \tau^-_{loc}. \tag{4.39}$$

From the algorithm (specification of the Req generation module), it follows that at time $t_{p,k+1}$

$$\neg \widetilde{THGEQ}^{e}_{p}(t_{p,k+1}) \tag{4.40}$$

[and also $\neg \widetilde{THGR}^{e}_{p}(t_{p,k+1})$, which is handled analogously, cf. (a.ii) below] must hold, i.e., the threshold signals that generated (the odd) transition k must be inactive again. Therefore, for the times $t_{q''}$ defined as

$$t_{q''} := d^{-1}(\langle \mathcal{P}^{GEQ}toGEQ, e, p, q' \rangle; d^{-1}(\langle \sum GEQtoTHGEQ, e, p \rangle; t_{p,k+1})) \tag{4.41}$$

it must hold that

$$\nexists Q'', |Q''| \geqslant \alpha : \forall q'' \in Q'' : \widetilde{\mathcal{P}}^{GEQ,e}_{p,q''}(t_{q''}). \tag{4.42}$$

Let us choose $Q'' := Q'$. Because of the FIFO property of the channels and because of $t_{p,k} < t_{p,k+1}$, we obtain

$$t_{q'} < t_{q''}.$$

Clearly, there has to be at least one $q' \in Q'$ for which

$$\widetilde{\mathcal{P}}^{GEQ,e}_{p,q'}(t_{q'}) \text{ but } \neg\widetilde{\mathcal{P}}^{GEQ,e}_{p,q'}(t_{q''}), \tag{4.43}$$

since otherwise $Q'' := Q'$ would have been a choice for Q'', contradicting (4.42). However, (4.43) can only hold if local transition k has been in pipepair (p, q') at time $t_{q',rcv,k}$, with

$$t_{q',rcv,k} \leqslant t_{q''}.$$

In combination with (4.41) and the channel properties, this implies

$$t_{p,k+1} - t_{q',rcv,k} \geqslant \tau^-_{TH} + \tau^-_{PC} \tag{4.44}$$

and, by the local channel properties,

$$t_{q',rcv,k} - t_{p,k} \geqslant \tau^-_{loc}. \tag{4.45}$$

Finally, (4.39) together with (4.44) and (4.45) yields

$$t_{p,k+1} - t_{p,k-1} \geqslant (\tau^-_{TH} + \tau^-_{PC} + \tau^-_{Diff} + \tau^-_{loc}) + (\tau^-_{TH} + \tau^-_{PC} + \tau^-_{loc}). \tag{4.46}$$

case (a.ii): Assuming that implication (ii) is valid, i.e., that there exists a set Q' of size $|Q'| \geqslant \beta$, such that, for

$$t_{q'} := d^{-1}(\langle \mathcal{P}^{GR} toGR, e, p, q' \rangle; d^{-1}(\langle \sum GRtoTHGR, e, p \rangle; t_{p,k}))$$

it holds that

$$\forall q' \in Q' : \widetilde{\mathcal{P}}^{GR,e}_{p,q'}(t_{q'}) \wedge \#r^{loc}_{p,q'}(t_{q'}) \geqslant k-1.$$

By analogous arguments as in case (a.i), we obtain

$$t_{p,k+1} - t_{p,k-1} \geqslant (\tau^-_{TH} + \tau^-_{PC} + \tau^-_{Diff} + \tau^-_{loc}) + (\tau^-_{TH} + \tau^-_{PC} + \tau^-_{loc}). \tag{4.47}$$

Combination of (a.i) and (a.ii): Combining (4.37), (4.46) and (4.47) leads to

$$\begin{aligned}(\tau^-_{TH} + \tau^-_{PC} + \tau^-_{Diff} + \tau^-_{loc}) + (\tau^-_{TH} + \tau^-_{PC} + \tau^-_{loc}) \\ \leqslant t_{p,k+1} - t_{p,k-1} < \tau^+_{TH} + \tau^+_{PC} + \tau^+_{loc},\end{aligned}$$

which is a contradiction to Constraint 1.

case (b): If $\widetilde{THGR}^{o}_{p}(t_{p,k+1})$, then, by the algorithm (specification of the Threshold module), there must be a set $Q \subseteq P \setminus \{p\}$, of size $|Q| \geqslant \beta$, such that, for $t' := d^{-1}(\langle \sum GRtoTHGR, o, p \rangle; t_{p,k+1})$

$$\forall q \in Q : \widetilde{GR}^{o}_{p,q}(t').$$

By analogous arguments as in case (a), we obtain an analogon to (4.37) as

$$t_{p,k+1} - t_{p,k-1} < \tau^+_{TH} + \tau^+_{PC} + \tau^+_{loc}, \tag{4.48}$$

as well as analogons to (4.46) and (4.47), namely,

$$t_{p,k+1} - t_{p,k-1} \geqslant (\tau^-_{TH} + \tau^-_{PC} + \tau^-_{Diff} + \tau^-_{loc}) + (\tau^-_{TH} + \tau^-_{PC} + \tau^-_{loc}). \tag{4.49}$$

and

$$t_{p,k+1} - t_{p,k-1} \geqslant (\tau^-_{TH} + \tau^-_{PC} + \tau^-_{Diff} + \tau^-_{loc}) + (\tau^-_{TH} + \tau^-_{PC} + \tau^-_{loc}). \tag{4.50}$$

Combination of (b.i) and (b.ii): Combining (4.48), (4.49) and (4.50) leads to

$$\begin{aligned}(\tau^-_{TH} + \tau^-_{PC} + \tau^-_{Diff} + \tau^-_{loc}) + (\tau^-_{TH} + \tau^-_{PC} + \tau^-_{loc}) \\ \leqslant t_{p,k+1} - t_{p,k-1} < \tau^+_{TH} + \tau^+_{PC} + \tau^+_{loc},\end{aligned}$$

which is a contradiction to Constraint 1. ■

The next lemma provides a lower bound on the time p sends transition $k \geqslant 1$. Before we state the lemma, we introduce some notation:

Consider a $GJ_{\mathrm{Imp}}(J)$ module p. For any q in $\mathcal{Q}$ and $k \geqslant 0$, we denote with $t_{q,k}^{\mathrm{GEQ}}$ (respectively $t_{q,k}^{\mathrm{GR}}$) the first time t, such that, in case $k \in \{0, 2, \dots\}$, $\widetilde{\mathcal{P}}_{p,q}^{GEQ,e}(t)$ (respectively $\widetilde{\mathcal{P}}_{p,q}^{GR,e}(t)$) and $\#r_{p,q}^{loc}(t) = k$ hold, and in case $k \in \{1, 3, \dots\}$, $\widetilde{\mathcal{P}}_{p,q}^{GEQ,o}(t)$ (respectively $\widetilde{\mathcal{P}}_{p,q}^{GR,o}(t)$) and $\#r_{p,q}^{loc}(t) = k$ hold. For each $k \geqslant 0$, let t_k^{GEQ} be the earliest time t, such that, there exists a set $Q \subseteq \mathcal{Q}$ of size $|Q| \geqslant \alpha$, where for all q in Q, $\#r_{p,q}^{rem}(t) \geqslant k$ and let t_{k+1}^{GR} be the earliest time t, such that, there exists a set $Q \subseteq \mathcal{Q}$ of size $|Q| \geqslant \beta$, where for all q in Q, $\#r_{p,q}^{rem}(t) \geqslant k+1$.

We are now ready to state:

Lemma 4.7 (C1). *Let p be a correct $GJ_{Imp}(J)$ module with input channels. Then p sends transition $k \geqslant 1$ at time $t_{p,k}$ with*

$$t_{p,1} \geqslant \tau_{PC}^{-} + \tau_{TH}^{-} \text{ and} \tag{4.51}$$

$$t_{p,k+1} \geqslant \min \left\{ \begin{array}{l} \max\{t_k^{GEQ}, t_{p,k} + \tau_{loc}^{-}\} + \tau_{Diff}^{-} \\ \max\{t_{k+1}^{GR}, t_{p,k} + \tau_{loc}^{-} + \tau_{Diff}^{-}\} \end{array} \right\} + \tau_{PC}^{-} + \tau_{TH}^{-} \tag{4.52}$$

Proof. By the algorithm (specification of the PCSG to Threshold module channels and Threshold modules), it follows that p sends transition 1 at $t_{p,1}$ with

$$t_{p,1} \geqslant \tau_{PC}^{-} + \tau_{TH}^{-}.$$

Thus (4.51) holds.

We will next prove, that (4.52) holds, too. Assume that p sends transition $k+1 \geqslant 2$ at time $t_{p,k+1}$, wlog. for $k \in \{0, 2, \dots\}$. We apply Lemma 4.6 for transition $k+1$. Thus, either implication (i) or (ii) has to be true:

case (i): There exists at set $Q \subseteq \mathcal{Q}$, $|Q| \geqslant \alpha$, where for all q in Q, for some $t_q \leqslant t_{p,k+1} - \tau_{TH}^{-} - \tau_{PC}^{-}$,

$$\begin{array}{l} \widetilde{\mathcal{P}}_{p,q}^{GEQ,e}(t_q) \wedge (\#r_{p,q}^{loc}(t_q) \geqslant k) \quad \Rightarrow \\ (\#s_{p,q}^{loc}(t_q) = 1) \wedge (\#r_{p,q}^{loc}(t_q) \geqslant k). \end{array}$$

By applying Lemma 4.5, we obtain

$$\#r_{p,q}^{loc}(t_q - \tau_{Diff}^{-}) \geqslant k \text{ and} \tag{4.53}$$

$$\#r_{p,q}^{rem}(t_q - \tau_{Diff}^{-}) \geqslant k. \tag{4.54}$$

Because of (4.54) it must hold that,

$$t_k^{\mathrm{GEQ}} \leqslant t_{p,k+1} - \tau_{Diff}^{-} - \tau_{TH}^{-} - \tau_{PC}^{-}. \tag{4.55}$$

Furthermore, by the local channel properties,

$$\#r^{loc}_{p,q}(t_q - \tau^-_{Diff}) \leqslant \#b_p(t_q - \tau^-_{Diff} - \tau^-_{loc}). \tag{4.56}$$

Combining (4.53) and (4.56) yields

$$\#b_p(t_q - \tau^-_{Diff} - \tau^-_{loc}) \geqslant k,$$

i.e., transition k must have been sent at time $t_{p,k}$ with

$$\begin{aligned} t_{p,k} &\leqslant t_q - \tau^-_{Diff} - \tau^-_{loc} \\ &\leqslant t_{p,k+1} - \tau^-_{Diff} - \tau^-_{TH} - \tau^-_{PC} - \tau^-_{loc}. \end{aligned} \tag{4.57}$$

Combination of (4.55) and (4.57) yields,

$$t_{p,k+1} \geqslant \max\{t^{\mathrm{GEQ}}_k, t_{p,k} + \tau^-_{loc}\} + \tau^-_{Diff} + \tau^-_{TH} + \tau^-_{PC} \tag{4.58}$$

case (ii): There exists at set $Q \subseteq \mathcal{Q}$, $|Q| \geqslant \beta$, where for all q in Q, for some $t_q \leqslant t_{p,k+1} - \tau^-_{TH} - \tau^-_{PC}$,

$$\begin{aligned} &\widetilde{\mathcal{P}}^{GR,e}_{p,q}(t_q) \wedge (\#r^{loc}_{p,q}(t_q) \geqslant k) \quad \Rightarrow \\ &(\#s^{loc}_{p,q}(t_q) = 1) \wedge (\#r^{loc}_{p,q}(t_q) \geqslant k) \wedge (\#r^{rem}_{p,q}(t_q) \geqslant k+1). \end{aligned} \tag{4.59}$$

Thus it must hold,

$$t^{\mathrm{GR}}_{k+1} \leqslant t_{p,k+1} - \tau^-_{TH} - \tau^-_{PC}. \tag{4.60}$$

Furthermore, from applying Lemma 4.5 to (4.59) we obtain,

$$\#r^{loc}_{p,q}(t_q - \tau^-_{Diff}) \geqslant k \text{ and} \tag{4.61}$$

By the local channel properties,

$$\#r^{loc}_{p,q}(t_q - \tau^-_{Diff}) \leqslant \#b_p(t_q - \tau^-_{Diff} - \tau^-_{loc}). \tag{4.62}$$

Combining (4.61) and (4.62) yields

$$\#b_p(t_q - \tau^-_{Diff} - \tau^-_{loc}) \geqslant k,$$

i.e., transition k must have been sent at time $t_{p,k}$ with

$$\begin{aligned} t_{p,k} &\leqslant t_q - \tau^-_{Diff} - \tau^-_{loc} \\ &\leqslant t_{p,k+1} - \tau^-_{Diff} - \tau^-_{TH} - \tau^-_{PC} - \tau^-_{loc}. \end{aligned} \tag{4.63}$$

Combination of (4.60) and (4.63) yields,

$$t_{p,k+1} \geqslant \max\{t^{\mathrm{GR}}_{k+1}, t_{p,k} + \tau^-_{loc} + \tau^-_{Diff}\} + \tau^-_{TH} + \tau^-_{PC} \tag{4.64}$$

The lemma follows from combining (4.58) and (4.64). ■

From Lemma 4.7, a minimum duration between two successive transitions sent by a correct $GJ_{\mathrm{Imp}}(J)$ module p directly follows:

Corollary 4.2 (C1). *Let p be a correct $GJ_{Imp}(J)$ module. If p sends transition $k \geqslant 1$ at time $t_{p,k}$, then it cannot send transition $k+1$ before $t_{p,k} + T_{min}$.*

Before we proceed to prove further properties on a $GJ_{\mathrm{Imp}}(J)$ module p with input channels, we need to introduce some notation: for predecessor module q in $\mathcal{Q}$, we say that *q sends transition $k \geqslant 1$ at time t*, iff b_q makes transition k at time t.

Lemma 4.8 together with Constraint 2 allows us to exclude the possibility of unbounded queuing effects inside a pipepair (p, q) for q in $\mathcal{Q}$. Constraint 2 and Assumption 1 ensure that the Diff-gate can remove matching transitions at least as fast as any predecessor module can send them.

Constraint 2. [C2] $\tau^+_{Diff} \leqslant T_{min}$

Assumption 1. *We say that Assumption 1 holds for a $GJ_{Imp}(J)$ module p, iff, for all q in $\mathcal{Q}$: if q is correct and it sends transition $k \geqslant 1$ at time $t_{q,k}$, then it does not send transition $k+1$ before $t_{q,k} + T_{min}$.*

Lemma 4.8 (C1,C2). *Let p be a correct $GJ_{Imp}(J)$ module with input channels and assume Assumption 1 holds for p. For any correct predecessor module q in $\mathcal{Q}$ and $k \geqslant 1$: If both*

(i) p sent transition k at $t_{p,k}$ and

(ii) q sent transition k at $t_{q,k}$,

then transition $k-1$ is removed from the local and remote pipe of (p, q) by time $\max\{t_{k,p} + \tau^+_{loc}, t_{k,q} + \tau^+_{rem}\} + \tau^+_{Diff}$.

Proof. The proof is by induction on the number of transitions $k \geqslant 1$ that are sent by p and q.

Begin ($k = 1$): Assume that p sends transition 1 at $t_{p,k}$. Transition 1 will be received in any of p's local pipes by $t_{p,k} + \tau^+_{loc}$. Furthermore, assume that q sends transition 1 at $t_{q,k}$. It will certainly be received in the remote pipe of the pipepair (p, q) by $t_{q,k} + \tau^+_{rem}$. Since there is no transition that could block the removing of transition 0, transition 0 is removed by $\max\{t_{p,k} + \tau^+_{loc}, t_{q,k} + \tau^+_{rem}\} + \tau^+_{Diff}$ from both the local and remote pipe, according to the Diff-gate properties.

Step ($k > 1$): As our induction hypothesis, assume that transition $k-2$ is removed from both pipes by $\max\{t_{p,k-1} + \tau^+_{loc}, t_{q,k-1} + \tau^+_{rem}\} + \tau^+_{Diff}$. By analogous arguments as above, transition k is received in both pipes of (p, q) by t_{both}, defined as

$$t_{both} := \max\{t_{p,k} + \tau^+_{loc}, t_{q,k} + \tau^+_{rem}\}.$$

Because of Corollary 4.2 and since Assumption 1 holds for p, consecutive transitions cannot be generated with less than T_{min} distance in-between, i.e.,

$$t_{p,k} - t_{p,k-1} \geqslant T_{min} \text{ and}$$
$$t_{q,k} - t_{q,k-1} \geqslant T_{min}.$$

Thus

$$\begin{aligned} t_{both} &\geqslant \max\{t_{p,k-1} + \tau^+_{loc}, t_{q,k-1} + \tau^+_{rem}\} + T_{min} \\ &\geqslant \max\{t_{p,k-1} + \tau^+_{loc}, t_{q,k-1} + \tau^+_{rem}\} + \tau^+_{Diff} \end{aligned}$$

by Constraint 2. According to the induction hypothesis, transition $k-2$ has already been removed by t_{both}. By the properties of the Diff-gate, transition $k-1$ is hence removed by $t_{both} + \tau^+_{Diff}$. ■

On executions with marginal queueing effects. Lemma 4.8 provided us with means to get rid of queueing effects when considering executions of a correct $GJ_{\text{Imp}}(J)$ module with input channels: whenever a local or remote transition $k \geqslant 1$ arrives at the latest possible time at pipepair (p, q) for some correct q in $\mathcal{Q}$, there are no queueing effects from before that could further delay the removal of transition $k-1$.

We will next show that it even suffices to study executions of $GJ_{\text{Imp}}(J)$ modules with input channels, where queuing effects can block the removal of transition $k-1$ no longer than ε, for an arbitrarily small $\varepsilon \geqslant 0$. For this purpose we introduce the concept of non-ε-queueing respectively non-ε-blocking executions:

Consider an execution e of a $GJ_{\text{Imp}}(J)$ module with input channels. Let us denote the time of arrival of transition k at the local (respectively remote) pipe of (p, q) in e by $t^l_{q,k}$ (respectively $t^r_{q,k}$).

For $k \geqslant 2$, by the Diff-gate specification, the removal of transition $k-1$ starts at time $t_{dif,k-2} = \max\{t^r_{q,k}, t^l_{q,k}, t_{rmv,k-2}\}$, where $t_{rmv,k-2}$ is the time transition $k-2 \geqslant 0$ is removed from both the local and remote pipe of (p, q). Thus a late $t_{rmv,k-2}$ can lead to queueing effects in deleting transition $k-1$ from the pipepair (p, q). We say that the *Diff-gate is busy* at time $t \geqslant 0$, iff t is within an interval $[t_{dif,k}, t_{rmv,k}]$, for $k \geqslant 0$. For an arbitrary interval $\mathcal{I}$, the Diff-gate is busy during $\mathcal{I}$, iff it is busy for all t in $\mathcal{I}$. For some small $\varepsilon \geqslant 0$, we say *the removal of transition $k-1$ at (p, q) did not face ε-queueing* whenever

$$t^r_{q,k} > t_{rmv,k-2} - \varepsilon \text{ or } t^l_{q,k} > t_{rmv,k-2} - \varepsilon,$$

which is equivalent to

$$t_{rmv,k-2} < \max\{t^r_{q,k}, t^l_{q,k}\} + \varepsilon.$$

Otherwise *it did face ε-queueing*. In case of 0-queuing, we simply speak of *queueing*. If the removal of transition $k-1$ at (p, q) did not face ε-queueing,

$$t_{dif,k-1} \in \max\{t^r_{q,k}, t^l_{q,k}\} + [0, \varepsilon] \tag{4.65}$$

holds. We say that an execution e of $GJ_{\mathrm{Imp}}(J)$ is *non-ε-queuing* iff for all $k \geqslant 2$, q in $\mathcal{Q}$, the removal of transition $k-1$ at (p,q) did not face ε-queueing.

We obtain the following result:

Lemma 4.9 (C1,C2). *Let e be an execution of a $GJ_{Imp}(J)$ module p with input channels. If Assumption 1 holds for p, then there exists an execution e' of p with input channels, denoted by $e' = nonqueueing_{\varepsilon}(e)$, for an arbitrarily small $\varepsilon > 0$, for which:*

- *e' is non-ε-queueing,*
- *e' and e have the same behavior of input and output ports,*

Proof. Let e' be identical to e, except that for each $k \geqslant 1$, we shift the time of arrival of transition k at the local pipe of (p,q) in e, say $t^l_{q,k}$ to $t'^l_{q,k}$ in e', and the time of arrival of transition k at the remote pipe of (p,q) in e, say $t^r_{q,k}$ to $t'^r_{q,k}$ in e' as follows. Clearly, one has to make sure that e' still is a valid execution. For this purpose, we will show by induction on $k \geqslant 1$ that the new arrival times $t'^l_{q,k}$ and $t'^r_{q,k}$ (i) do not violate the FIFO property or the delay bounds of the local and remote channels and (ii) during $\left[t^l_{q,k}, t'^l_{q,k}\right]$ as well as during $\left[t^r_{q,k}, t'^r_{q,k}\right]$ the Diff-gate is busy. The latter is important, as it makes sure that the behavior of the PCSG's output ports is the same in e and e': if the Diff-gate of pipepair (p,q) is busy at time $t \in \mathcal{T}$, all output ports of (p,q)'s PCSG module are inactive.

For $k = 1$: $t'^l_{q,k} = t^l_{q,k}$ and $t'^r_{q,k} = t^r_{q,k}$. The FIFO property and the delay bounds of both the local and remote channels are trivially fulfilled. Thus both (i) and (ii) are valid.

For $k \geqslant 2$: assume that both (i) and (ii) hold for $k-1$. We will show that they hold for k, too. We distinguish between two cases:

- Assume the removal of transition $k-1$ at (p,q) did *not* face queueing. Further we distinguish between two cases on the order of occurrence of $t^r_{q,k}$ and $t^l_{q,k}$:
  - In case $t^r_{q,k} \geqslant t^l_{q,k}$ and since $k-1$ did not face queuing at (p,q), $t_{rmv,k-2} < t^r_{q,k}$ and $t_{dif,k-1} = t^r_{q,k}$. Thus the Diff-gate is not busy within $\left[t_{rmv,k-2}, t^r_{q,k}\right]$. By the induction hypothesis, (ii) holds for $k-1$ and thus the arrival of remote transition $k-1$ is not shifted beyond $t^r_{q,k}$. We may thus safely set
  $$t'^r_{q,k} = t^r_{q,k},$$
  without violating the FIFO property of the remote channel. We will next choose a value for $t'^l_{q,k-1}$. If $t^l_{q,k} > t'^l_{q,k-1}$, we may safely set
  $$t'^l_{q,k} = t^l_{q,k}. \tag{4.66}$$
  Otherwise, (4.66) would violate the FIFO property of the local channel and we set $t'^l_{q,k}$ to a value within $\left(t'^l_{q,k-1}, t_{rmv,\ell}\right)$, where $t_{rmv,\ell}$, $\ell \geqslant 0$, is the next removal of a transition at (p,q) that occurs after $t'^l_{q,k-1}$. Note that the Diff-gate is busy throughout $\left[t^l_{q,k}, t'^l_{q,k}\right]$. Thus both (i) and (ii) are fulfilled.

– Otherwise, if $t^r_{q,k} < t^l_{q,k}$, we set

$$t'^l_{q,k} = t^l_{q,k}.$$

If $t^r_{q,k} > t'^r_{q,k-1}$, we set

$$t'^r_{q,k} = t^r_{q,k}.$$

Otherwise, in order not to violate the FIFO property of the remote channel, we set $t'^r_{q,k}$ to a value within $\big(t'^r_{q,k-1}, t_{rmv,\ell}\big)$, where $t_{rmv,\ell}$, $\ell \geqslant 0$, is the next removal of a transition at (p,q) that occurs after $t'^r_{q,k-1}$. Again the the Diff-gate is busy throughout $\big[t^r_{q,k}, t'^r_{q,k}\big]$. By analogous arguments as before, (i) and (ii) are hence fulfilled.

(i) and (ii) are fulfilled in both cases.

- Assume the removal of transition $k-1$ at (p,q) did face queueing. We distinguish between two cases for q.

  – q is correct. Then we may apply Lemma 4.8 and obtain that transition $k-2$ is removed at

$$\begin{aligned} t_{rmv,k-2} &\leqslant \max\{t_{p,k-1} + \tau^+_{loc}, t_{q,k-1} + \tau^+_{rem}\} + \tau^+_{Diff} \\ &\leqslant \max\{t_{p,k} - T_{min} + \tau^+_{loc}, t_{q,k} - T_{min} + \tau^+_{rem}\} + \tau^+_{Diff} \\ &\leqslant \max\{t_{p,k} + \tau^+_{loc}, t_{q,k} + \tau^+_{rem}\}. \end{aligned}$$

Thus, it is always possible, for $t'^l_{q,k}$ respectively $t'^r_{q,k}$, to be set such that $t'^l_{q,k}$ within $t_{rmv,k-2} + [-\varepsilon, 0)$ respectively $t'^r_{q,k}$ within $t_{rmv,k-2} + [-\varepsilon, 0)$ without violating the delay bounds of the local and remote channels. Wlog. assume that we may shift $t'^r_{q,k}$ to a later time, by setting $t'^r_{q,k}$ to a value $\geqslant t^r_{q,k}$ within $t_{rmv,k-2} + [-\varepsilon, 0)$. Note that the Diff-gate is busy during $\big[t^r_{q,k}, t'^r_{q,k}\big]$. For $t^l_{q,k}$ we again have to make sure that it does not violate the FIFO property of the local channel. If $t^l_{q,k} > t'^l_{q,k-1}$, set

$$t'^l_{q,k} = t^l_{q,k}$$

and otherwise set $t'^l_{q,k}$ to a value within $\big(t'^l_{q,k-1}, t_{rmv,\ell}\big)$, where $t_{rmv,\ell}$, $\ell \geqslant 0$, is the next removal of a transition at (p,q) that occurs after $t'^l_{q,k-1}$. Again the Diff-gate is busy during $\big[t^l_{q,k}, t'^l_{q,k}\big]$. Thus (i) and (ii) are fulfilled.

  – In case q is faulty, we do not assume any bounds on the remote channel delay. We can thus choose $t'^r_{q,k}$ to lie within $t_{rmv,k-2} + [-\varepsilon, 0)$ without having to pay attention to not violating the delay bounds of the remote channel and then proceed as above. Again (i) and (ii) are fulfilled.

(i) and (ii) are hence fulfilled in both cases as well.

Thus, e' does not violate the FIFO property of the local and remote channels, does not violate the upper bounds of the local and remote channel delays, and is non-ε-queueing by definition.

Further, since, the arrival of local and remote transitions are only shifted throughout intervals during which the Diff-gate is busy, the status outputs of the PCSG modules behave the same in e' and e. Thus, e' is a valid execution. ■

One can transform an execution of $GJ_{\mathrm{Imp}}(J)$ with input channels even further: we say that execution e of the General Join implementation p is *non-ε-blocking*, for an arbitrarily small $\varepsilon \geqslant 0$, iff (i) e is non-ε-queueing and (ii) it holds that for all $k \geqslant 1$ and q in $\mathcal{Q}$, transition $k-1$ is removed from pipepair (p,q) at

$$t_{rmv,k-1} \leqslant \max\{t^l_{q,k} + \tau^+_{Diff}, t^r_{q,k+1}\} + \varepsilon. \tag{4.67}$$

We are now ready to deduce:

Lemma 4.10 (C1,C2). *Let e be an execution of $GJ_{Imp}(J)$ module p with input channels. If Assumption 1 holds for p, then, for an arbitrarily small $\varepsilon > 0$, there exists an execution e' of p with input channels denoted by $e' = nonblocking_{\varepsilon}(e)$, for which:*

- *e' is non-ε-blocking,*
- *e' and e have the same behavior of input and output ports.*

Proof. Let e' be identical to $nonqueueing_{\varepsilon}(e)$ except for the following changes:

For each $k \geqslant 1$ and q in $\mathcal{Q}$, distinguish between the two cases:

- Assume $t^r_{q,k+1} \geqslant t^l_{q,k+1}$. Clearly then, since e' is non-ε-queueing, transition $k-1$ is removed from the pipepair (p,q) at $t_{rmv,k-1}$ with

$$\begin{aligned} t_{rmv,k-1} &< \max\{t^r_{q,k+1}, t^l_{q,k+1}\} + \varepsilon \\ &= t^r_{q,k+1} + \varepsilon \;\Rightarrow \\ t_{rmv,k-1} &\leqslant \max\{t^r_{q,k+1} + \varepsilon, t^l_{q,k} + \tau^+_{Diff} + \varepsilon\}. \end{aligned}$$

  Thus (4.67) is fulfilled.

- Otherwise, assume that $t^r_{q,k+1} < t^l_{q,k+1}$. We distinguish between two cases for $t^r_{q,k+1}$, namely, (i) $t^r_{q,k+1} + \varepsilon \geqslant t_{rmv,k-1}$ and (ii) $t^r_{q,k+1} + \varepsilon < t_{rmv,k-1}$:

  ad (i) By the assumption transition $k-1$ is removed at

$$\begin{aligned} t_{rmv,k-1} &\leqslant t^r_{q,k+1} + \varepsilon \\ &\leqslant \max\{t^r_{q,k+1} + \varepsilon, t^l_{q,k} + \tau^+_{Diff} + \varepsilon\}. \end{aligned}$$

Thus (4.67) is fulfilled.

ad (ii) This is the only case, where e' is changed. Choose $t'^r_{q,k+1}$ such that $t'^r_{q,k+1} \geqslant t^r_{q,k+1}$, $t'^r_{q,k+1} > t'^r_{q,k}$ and $t'^r_{q,k+1}$ within

$$t_{rmv,k-1} + [-\varepsilon, 0)\,.$$

Clearly, this shifting ensures the FIFO property of the remote channel. Further it does not violate the remote channel delay bounds and does not change the value of $\widetilde{\mathcal{P}}^{GEQ,e}_{p,q}(t)$, $\widetilde{\mathcal{P}}^{GEQ,o}_{p,q}(t)$, $\widetilde{\mathcal{P}}^{GR,e}_{p,q}(t)$ and $\widetilde{\mathcal{P}}^{GR,o}_{p,q}(t)$ in e' for all $t \in \mathcal{T}$, since the Diff-gate is busy during $\left[t^r_{q,k+1}, t'^r_{q,k+1}\right]$.

■

We will next derive results for non-ε-queueing and non-ε-blocking executions of a $GJ_{\mathrm{Imp}}(J)$ module with input channels.

Lemma 4.11 (C1,C2). *Consider a non-ε-queueing execution of a $GJ_{Imp}(J)$ module p with input channels, where $\varepsilon \geqslant 0$. If Assumption 1 holds for p, then p sends transition $k+1 \geqslant 2$ at time $t_{p,k+1}$ with*

$$t_{p,k+1} \leqslant \max\{t^{GEQ}_k, t_{p,k} + \tau^+_{loc}\} + \tau^+_{Diff} + \varepsilon + \tau^+_{PC} + \tau^+_{TH} \tag{4.68}$$

Proof. Wlog. assume that $k \in \{0, 2, \dots\}$. Let

$$t' := \max\{t^{\mathrm{GEQ}}_k, t_{p,k} + \tau^+_{loc}\} + \tau^+_{Diff} + \varepsilon + \tau^+_{PC} + \tau^+_{TH}.$$

By definition of t^{GEQ}_k, there exists a set $Q \subseteq \mathcal{Q}$ with $|Q| \geqslant \alpha$ such that for all $q \in Q$, $\#r^{rem}_{p,q}(t^{\mathrm{GEQ}}_k) \geqslant k$. We distinguish two cases:

(a) Suppose that $\exists q \in Q : \#r^{loc}_{p,q}(t') > k$. Since $\#r^{loc}_{p,q}(t') \leqslant \#b_p(t')$, as no transition can be locally received if it has not been sent before, $\#b_p(t') > k$ must hold. Thus, p has sent transition $k+1$ by t'. The upper bound follows.

(b) Otherwise, assume that

$$\forall q \in Q : \#r^{loc}_{p,q}(t') \leqslant k. \tag{4.69}$$

Enabling condition: p must receive transition k in all its local pipes by $t_{rcv,p} := t_{p,k} + \tau^+_{loc}$. Since the execution is non-ε-queueing, it follows that transition $k-1$ is removed from all of pipepairs (p,q) (for all $q \in Q$) by

$$t_{del,k-1} := \max\{t_{rcv,p}, t^{\mathrm{GEQ}}_k\} + \tau^+_{Diff} + \varepsilon.$$

Consequently, at

$$\begin{aligned} t_{rcv} &:= \max\{t_{rcv,p}, t_k^{\mathrm{GEQ}}, t_{del,k-1}\} \\ &\leqslant \max\{t_{p,k} + \tau_{loc}^+, t_k^{\mathrm{GEQ}}\} + \tau_{Diff}^+ + \varepsilon \end{aligned}$$

with $t_{rcv} \leqslant t'$, it holds that:

$$\#r_{p,q}^{loc}(t_{rcv}) \geqslant k \qquad \#r_{p,q}^{rem}(t_{rcv}) \geqslant k \qquad \#d_{p,q}(t) \geqslant k-1 \tag{4.70}$$

By combination of (4.70) with (4.69), it holds that for all $\xi \in [t_{rcv}, t']$:

$$\#r_{p,q}^{loc}(\xi) = k. \tag{4.71}$$

Furthermore,

$$\begin{aligned} \widetilde{\mathcal{P}}_{p,q}^{GEQ,e}(\xi) &\equiv (\#r_{p,q}^{rem}(\xi) \geqslant \#r_{p,q}^{loc}(\xi)) \wedge (\#r_{p,q}^{loc}(\xi) \in \{0,2,\dots\}) \wedge (\#s_{p,q}^{loc}(\xi) = 1) \\ &\equiv (\#r_{p,q}^{rem}(\xi) \geqslant k) \wedge (\#r_{p,q}^{loc}(\xi) - \#d_{p,q}(\xi) = 1) \\ &\equiv (\#d_{p,q}(\xi) = k-1). \end{aligned} \tag{4.72}$$

Assuming $\exists q \in Q : \#d_{p,q}(\xi) > k-1$ implies $\#r_{p,q}^{loc}(\xi) > k$, contradicting (4.71). Thus $\forall q \in Q : \#d_{p,q}(\xi) = k-1$ must hold. Therefore $\forall q \in Q : \widetilde{\mathcal{P}}_{p,q}^{GR,e}(\xi)$ is true for $\xi \in [t_{rcv}, t']$.

By the algorithm (specification of the PCSG to Threshold module channels), $\forall q \in Q : \widetilde{GEQ}_{p,q}^{e}(\xi)$ is true for $\xi \in \left[t_{rcv} + \tau_{PC}^+, t'\right]$. Again by the algorithm (Threshold module),

$$\widetilde{THGEQ}_{p}^{e}(\xi) \tag{4.73}$$

is true for

$$\begin{aligned} \xi \in &\left[t_{rcv} + \tau_{PC}^+ + \tau_{TH}^+, t'\right] \subseteq \\ &\left[\max\{t_{p,k} + \tau_{loc}^+, t_k^{\mathrm{GEQ}}\} + \tau_{Diff}^+ + \varepsilon + \tau_{PC}^+ + \tau_{TH}^+, t'\right]. \end{aligned}$$

Disabling condition: It remains to be shown that the disabling path cannot inhibit the generation of transition $k+1$ at p. For the sake of contradiction, assume that the disabling path can enforce $t_{p,k+1} > t'$.

Transition k must eventually be received in all of p's local pipes corresponding to a predecessor module r in $\mathcal{Q}$. Thus, for all r in $\mathcal{Q}$, $\#r_{p,r}^{loc}(\xi) \geqslant k$, for $\xi \in \left[t_{p,k} + \tau_{loc}^+, t'\right]$. Since transition $k+1$ is not generated by t', we actually have $\#r_{p,r}^{loc}(\xi) = k$. As $k \in \{0,2,\dots\}$, this implies

$$\neg\left(\widetilde{\mathcal{P}}_{p,r}^{GR,o}(\xi) \vee \widetilde{\mathcal{P}}_{p,r}^{GEQ,o}(\xi)\right).$$

Consequently, by the algorithm (specification of the PCSG to Threshold module channels and the Threshold modules),

$$\neg \left(\overset{o}{\widetilde{THGR}}_p(\xi) \vee \overset{o}{\widetilde{THGE}}Q_p(\xi) \right) \tag{4.74}$$

for $\xi \in \left[t_{p,k} + \tau^+_{loc} + \tau^+_{PC} + \tau^+_{TH}, t' \right]$.

Combining (4.73) and (4.74), it is apparent that p must send transition $k+1$ by t', providing the required contradiction.

The lemma follows in both cases. ■

From Lemma 4.9 and Lemma 4.11 we can deduce:

Corollary 4.3 (C1,C2). *Let p be a $GJ_{Imp}(J)$ module with input channels and assume that Assumption 1 holds for p. If there exists a set $Q \subseteq \mathcal{Q}$ of correct processes with $|Q| \geqslant \alpha$, such that, for all $q \in Q$, all processes q send transition $k \geqslant 1$ by $t_{Q,k}$, then p sends transition $k+1$ at $t_{p,k+1} \leqslant \max\{t_{p,k} + \tau^+_{loc}, t_{Q,k} + \tau^+_{rem}\} + \tau^+_{Diff} + \tau^+_{PC} + \tau^+_{TH}$.*

Lemma 4.12 (C1,C2). *Consider a non-ε-blocking execution, $\varepsilon \geqslant 0$, of a $GJ_{Imp}(J)$ module p for which Assumption 1 holds. Then p sends transition $k+1 \geqslant 2$ at time $t_{p,k+1}$ with*

$$t_{p,k+1} \leqslant \max\{t^{GR}_{k+1}, t_{p,k} + \tau^+_{loc} + \tau^+_{Diff} + \varepsilon\} + \tau^+_{PC} + \tau^+_{TH} \tag{4.75}$$

The proof follows the proof of Lemma 4.11 throughout large parts. Nonetheless, to fully convince the reader from the result's correctness it is stated here.

Proof. Wlog. assume that $k \in \{0, 2, \dots\}$. Let

$$t' := \max\{t^{\mathrm{GR}}_{k+1}, t_{p,k} + \tau^+_{loc} + \tau^+_{Diff} + \varepsilon\} + \tau^+_{PC} + \tau^+_{TH}.$$

By definition of t^{GR}_{k+1}, there exists a set $Q \subseteq \mathcal{Q}$ with $|Q| \geqslant \beta$ such that for all $q \in Q$, $\#r^{rem}_{p,q}(t^{\mathrm{GR}}_{k+1}) \geqslant k+1$. We distinguish two cases:

(a) Suppose that $\exists q \in Q : \#r^{loc}_{p,q}(t') > k$. Since $\#r^{loc}_{p,q}(t') \leqslant \#b_p(t')$, as no transition can be locally received if it has not been sent before, $\#b_p(t') > k$ must hold. Thus, p has sent transition $k+1$ by t'. The upper bound follows.

(b) Otherwise, assume that

$$\forall q \in Q : \#r^{loc}_{p,q}(t') \leqslant k. \tag{4.76}$$

Enabling condition: p must receive transition k in all its local pipes by $t_{rcv,p} := t_{p,k} + \tau^+_{loc}$. Since the execution is non-ε-blocking, it follows that transition $k-1$ is removed from all pipepairs (p,q), for all $q \in Q$, by

$$t_{del,k-1} := \max\{t_{rcv,p} + \tau^+_{Diff}, t^{\mathrm{GR}}_{k+1}\} + \varepsilon.$$

Consequently, at

$$\begin{aligned} t_{rcv} &:= \max\{t_{rcv,p}, t_{k+1}^{\mathrm{GR}}, t_{del,k-1}\} \\ &\leqslant \max\{t_{p,k} + \tau_{loc}^{+} + \tau_{Diff}^{+}, t_{k+1}^{\mathrm{GR}}\} + \varepsilon \end{aligned}$$

with $t_{rcv} \leqslant t'$, it holds that:

$$\#r_{p,q}^{loc}(t_{rcv}) > k \qquad \#r_{p,q}^{rem}(t_{rcv}) \geqslant k \qquad \#d_{p,q}(t) \geqslant k-1 \tag{4.77}$$

By combination of (4.77) with (4.76), it holds that for all $\xi \in [t_{rcv}, t']$:

$$\#r_{p,q}^{loc}(\xi) = k. \tag{4.78}$$

Furthermore,

$$\begin{aligned} \widetilde{\mathcal{P}}_{p,q}^{GEQ,e}(\xi) &\equiv (\#r_{p,q}^{rem}(\xi) > \#r_{p,q}^{loc}(\xi)) \wedge (\#r_{p,q}^{loc}(\xi) \in \{0, 2, \dots\}) \wedge (\#s_{p,q}^{loc}(\xi) = 1) \\ &\equiv (\#r_{p,q}^{rem}(\xi) \geqslant k) \wedge (\#r_{p,q}^{loc}(\xi) - \#d_{p,q}(\xi) = 1) \\ &\equiv (\#d_{p,q}(\xi) = k-1). \end{aligned} \tag{4.79}$$

Assuming $\exists q \in Q : \#d_{p,q}(\xi) > k-1$ implies $\#r_{p,q}^{loc}(\xi) > k$, contradicting (4.78). Thus $\forall q \in Q : \#d_{p,q}(\xi) = k-1$ must hold. Therefore $\forall q \in Q : \widetilde{\mathcal{P}}_{p,q}^{GR,e}(\xi)$ is true for $\xi \in [t_{rcv}, t']$.

By the algorithm (specification of the PCSG to Threshold module channels), $\forall q \in Q : \widetilde{GR}_{p,q}^{e}(\xi)$ is true for $\xi \in \left[t_{rcv} + \tau_{PC}^{+}, t'\right]$. Again by the algorithm (Threshold module),

$$\widetilde{THGR}_{p}^{e}(\xi) \tag{4.80}$$

is true for

$$\begin{aligned} \xi \in &\left[t_{rcv} + \tau_{PC}^{+} + \tau_{TH}^{+}, t'\right] \subseteq \\ &\left[\max\{t_{p,k} + \tau_{loc}^{+} + \tau_{Diff}^{+}, t_{k+1}^{\mathrm{GR}}\} + \varepsilon + \tau_{PC}^{+} + \tau_{TH}^{+}, t'\right]. \end{aligned}$$

Disabling condition: It remains to be shown that the disabling path cannot inhibit the generation of transition $k+1$ at p. For the sake of contradiction, assume that the disabling path can enforce $t_{p,k+1} > t'$.

Transition k must eventually be received in all of p's local pipes corresponding to a predecessor module r in $\mathcal{Q}$. Thus, for all r in $\mathcal{Q}$, $\#r_{p,r}^{loc}(\xi) \geqslant k$, for $\xi \in \left[t_{p,k} + \tau_{loc}^{+}, t'\right]$. Since transition $k+1$ is not generated by t', we actually have $\#r_{p,r}^{loc}(\xi) = k$. As $k \in \{0, 2, \dots\}$, this implies

$$\neg\left(\widetilde{\mathcal{P}}_{p,r}^{GR,o}(\xi) \vee \widetilde{\mathcal{P}}_{p,r}^{GEQ,o}(\xi)\right).$$

Consequently, by the algorithm (specification of the PCSG to Threshold module channels and the Threshold modules),

$$\neg \left(\overset{o}{\widetilde{THGR}}_p(\xi) \vee \overset{o}{\widetilde{THGEQ}}_p(\xi) \right) \tag{4.81}$$

for $\xi \in \left[t_{p,k} + \tau^+_{loc} + \tau^+_{PC} + \tau^+_{TH}, t' \right]$.

Combining (4.80) and (4.81), it is apparent that p must send transition $k+1$ by t', providing the required contradiction.

The lemma follows in both cases. ■

From Lemma 4.10 and Lemma 4.12 we can deduce:

Corollary 4.4 (C1,C2). *Consider a $GJ_{Imp}(J)$ module p for which Assumption 1 holds: If there exists a set $Q \subseteq \mathcal{Q}$ of correct processes with $|Q| \geqslant \beta$, such that, for all $q \in Q$, all processes q send transition $k+1 \geqslant 2$ by $t_{Q,k+1}$, then p sends transition $k+1$ at $t_{p,k+1} \leqslant \max\{t_{p,k} + \tau^+_{loc} + \tau^+_{Diff}, t_{Q,k+1} + \tau^+_{rem}\} + \tau^+_{PC} + \tau^+_{TH}$.*

The following corollary, deduced from Lemma 4.11 and Lemma 4.12, provides us with an upper bound on when p sends transition $k \geqslant 1$:

Corollary 4.5 (C1,C2). *Consider a non-ε-blocking, $\varepsilon \geqslant 0$, execution of a $GJ_{Imp}(J)$ module p with input channels, where Assumption 1 holds for p. Then p sends transition $k \geqslant 1$ at time $t_{p,k}$ with*

$$t_{p,1} \leqslant \tau^+_{PC} + \tau^+_{TH} \text{ and}$$
$$t_{p,k+1} \leqslant \min \left\{ \begin{array}{l} \max\{t^{GEQ}_k, t_{p,k} + \tau^+_{loc}\} + \tau^+_{Diff} + \varepsilon \\ \max\{t^{GR}_{k+1}, t_{p,k} + \tau^+_{loc} + \tau^+_{Diff}\} + \varepsilon \end{array} \right\} + \tau^+_{PC} + \tau^+_{TH}$$

Lemma 4.13 (C1,C2). *Let p' be a General Join module with input channels and p the corresponding $GJ_{Imp}(p')$ module with input channels. Let e be a non-ε-blocking execution, $\varepsilon \geqslant 0$, of p for which Assumption 1 holds. Then there exists an execution e' of p' where the delay bounds $\tau^{\pm}_{Diff}$ of p' are replaced by the bounds $\tau^{\pm}_{Diff} + [0, \varepsilon]$, such that, for all $k \geqslant 0$, $t \geqslant 0$ and q in $\mathcal{Q}$,*

- *$\#reqO(t) = k$ in e' iff $\#b_{p'}(t) = k$ in e.*
- *$\#b_q(t) = k$ in e' iff $\#b_q(t) = k$ in e.*
- *$\#r_q(t) = \#r'_q(t) + 1 = k$ in e' iff $\#r^{rem}_{p,q}(t) = k$ in e.*

Since the detailed proof is rather technical, we only give the proof idea here:

We first construct e' from e and show that it is an execution of General Join module p with input channels.

Choose the behavior of the input and output ports of e' identical to those in e, i.e., for all $k \geqslant 1$,

- p sends transition k in e' at the same time, p sends transition k in e,
- for all q in $\mathcal{Q}$, whenever q sends transition k in e, q sends transition k in e', and
- for all q in $\mathcal{Q}$, whenever transition k is received at the remote pipe of pipepair (p, q) in e, both ports q^r and q'^r make transition k in e'.

We then make use of an inductive argument over $k \geqslant 1$. From the above choice, it follows that for each $k \geqslant 1$, (i) the earliest time t by which transition k is received from a set of $Q \subseteq \mathcal{Q}$, $|Q| \geqslant \alpha$, predecessor modules in e' is identical to t_k^{GEQ} in e and (ii) the earliest time t by which transition $k+1$ is received from a set of $Q \subseteq \mathcal{Q}$, $|Q| \geqslant \beta$, predecessor modules in e' is identical to t_{k+1}^{GR} in e. Further, p must send transition $k+1$ in e within the bounds given by Lemma 4.7 and Corollary 4.5. By analogous arguments as given in the proof of Lemma 4.2, 4.3 and 4.4, we can thus let transition k arrive at the X^l and X'^l, for each X in $\mathcal{Q}$, such that p sends transition $k+1$ in e' at the same time, p sends transition $k+1$ in e.

From Lemma 4.13 we deduce the main result of this section, namely:

Theorem 4.1 (**C1, C2**). *Let p be a $GJ_{Imp}(J)$ module with input channels, where Assumption 1 holds for p. Then p with input channels implements the corresponding General Join module J with input channels, where the delay bounds $\tau_{Diff}^{\pm}$ of J are replaced by the bounds $\tau_{Diff}^{\pm} + [0, \varepsilon]$, for an arbitrarily small $\varepsilon > 0$.*

4.7 Performance of clockless algorithms

Given a clockless algorithm it is usually of interest to know its worst-case performance. While classical methods from distributed computing can be used to analyze the worst-case timing behavior of highly regular distributed systems (e.g., those with fully connected or regular communication graphs), stating worst case performance bounds for systems with irregular communication graphs is typically difficult.

In this section it is shown that executions of non fault-tolerant clockless algorithms built from bounded queue modules of size 1 (registers), forks and non-buffering joins, can be mapped to executions of the simple and elegant distributed algorithm called Full Reversal (FR) that was introduced by Gafni and Bertsekas [42]. FR is an algorithm on directed graphs[5] that transforms an initial graph G_0 with nodes from the set $N = \{0, \ldots, n\}$, $n \geqslant 0$, and links between nodes, in discrete time steps $t \geqslant 0$, yielding a series of graphs $G_0, G_1, \ldots, G_t, \ldots$ as follows: a node i in N is said to be *enabled in* G_t, $t \geqslant 0$, iff it only has incoming links in G_t. The graph G_t at time $t \geqslant 1$ is obtained from G_{t-1} by letting a non empty subset of nodes from the set $N \setminus \{0\}$ that are enabled in G_{t-1} perform a step at time $t-1$. We will denote the set of all these nodes by S_{t-1}. The result of node p *performing a*

[5] We further demand that the graph's support is connected.

step at time $t-1$ is that it reverts the directions of all its links in G_{t-1}. In case no nodes are enabled at time t, the algorithm terminates. In the original algorithm, node 0 plays a special role and never performs a step. Removal of node 0 leads to an algorithm that never terminates in case G_0 is acyclic. This adaption of FR has been studied in the context of mutual exclusion by Chandy and Misra [12] as well as Barbosa and Gafni [3] and will be used throughout this section. We will thus assume in the following that $N = \{1, \ldots, n\}$, for some $n \geqslant 1$. An *execution of FR with initial graph* G_0 is a (possibly infinite) alternating sequence $G_0, S_0, G_1, S_1, \ldots$ of graphs G_t, $t \geqslant 0$, and node sets $S_t \subseteq N$, $t \geqslant 0$, where G_t and S_t are defined as above. An execution where for all $t \geqslant 0$, S_t is the set of all nodes enabled in G_t is called *greedy*. We will next establish a relation between greedy FR executions and executions of a general class of non fault-tolerant clockless algorithms.

Relation to Executions of Clockless Algorithms. Consider a compound module $M(D_0, s)$, comprising only registers, non-buffering joins and forks, that is specified by a directed graph $D_0 = \langle N', E' \rangle$, called *data flow graph*[6], and an initial state function $s : E' \to \{P, S\}$, where P (standing for "predecessor") and S (standing for "successor") are constants. Function s determines the initial state of the communication interface, as described in Section 4.4, between two nodes p and q, where $\langle p, q \rangle \in E'$: in case $s(p, q) = S$, p is initially waiting for q's (q is among p's successor modules) acknowledge and in case $s(p, q) = P$, q is initially waiting for p's (p is among q's predecessor modules) request (and data). Module M then is obtained by the following means: for each p in N':

(i) There is exactly one register p in M.

(ii) For each $\langle q, p \rangle \in E'$, $q \in N'$, the output communication interface of q is connected to the input communication interface of p via channels: in case there is only one such q, directly, and otherwise, via a non-buffering join module.

(iii) For each $\langle p, q \rangle \in E'$, $q \in N'$, the output communication interface of p is connected to the input communication interface of q via channels: in case there is only one such q, directly, and otherwise, via a fork module.

Throughout this section we make use of the following simplifying assumption[7]:

Assumption 2. *For each* $\langle p, q \rangle \in E'$*: (i) the delay between* p *issuing a request (and applying the data on the bus) and* q *receiving the request (and the data) is exactly* 1 *and (ii) the delay between* q *issuing an acknowledge and* p *receiving it is exactly* 1*.*

Given a module $M(D_0, s)$ specified by $D_0 = \langle N', E' \rangle$ and function s, we will next construct an initial FR graph $G_0(D_0, s) = \langle N, E \rangle$ by: (i) $N = N'$ and (ii) $\langle p, q \rangle \in E$ iff: $\langle p, q \rangle \in E'$ and $s(p, q) = S$, or $\langle q, p \rangle \in E'$ and $s(q, p) = P$.

[6] Again, we demand that the support is connected.

[7] Getting rid of this assumption is central interest in future work. A refined delay assumption, e.g., with the possibility to specify different delay bounds for REQ and ACK signals thus is subject to current research.

Interestingly, the following correspondence holds:

Theorem 4.2. *If Assumption 2 holds in an execution of $M(D_0, s)$, p in N makes a step at time $t \geqslant 0$ iff p makes a step in the greedy FR execution with initial graph $G_0(D_0, s)$ at time t.*

Theorem 4.2 provides means to answer performance questions of executions of $M(D_0, s)$ by answering related performance questions of executions of FR with initial graph $G_0(D_0, s)$. For the latter, important measures are either known or have recently been discovered by means of a min-plus algebraic [47] approach.[8] Before stating one such measure, namely the average rate, we introduce some notation: a *chain* c (of length $\ell(c) = k \geqslant 1$) in graph $G_0 = \langle N, E\rangle$ is a finite sequence $c = (p_i)_{0 \leqslant i \leqslant k}$ of nodes p_i from the set N, such that for two successors p_i and p_{i+1}, either $\langle p_i, p_{i+1}\rangle \in E$ or $\langle p_{i+1}, p_i\rangle \in E$. Its *start node* is p_0, its *end node* is p_k and its *weight* $\omega_{G_0}(c)$ the number of links traversed in the direction of E, that is the number of $\langle p_i, p_{i+1}\rangle \in E$, where $0 \leqslant i \leqslant k$. For both FR executions with initial graph $G_0(D_0, s)$ and executions of module $M(D_0, s)$ let the *work* $\#w_p(t)$ of node $p \in N$ until and including time $t \geqslant 0$ in an execution be the number of steps made by p at times no greater than t in the execution, and the *average rate of* p in the execution be $\lim_{t\to\infty} \#w_p(t)/t$.

Average Rate. From Theorem 4.2, we deduce that, given a module $M(D_0, s)$ with graph $D_0 = \langle N', E'\rangle$ and function s, for each $p \in N'$, the average rate of p in the execution of module $M(D_0, s)$ is equal to the average rate of p in the greedy FR execution with initial graph $G_0(D_0, s)$.

From [3] (Theorem 16), it follows that the average rate of $p \in N$ in a greedy FR execution e_{G_0} with initial graph $G_0(D_0, s) = \langle N, E\rangle$ is equal to $\min\{\omega_{G_0}(c)/\ell(c) \mid c \in C(G_0)\}$, where $C(G_0)$ is the set of chains in G_0 with identical start and end node. From this, we obtain the following results: (i) If G_0 is cyclic, for each $p \in N$, the average rate of p in e_{G_0} is 0. If G_0 is acyclic, for each $p \in N$, the average rate of p in e_{G_0} is greater than 0. (ii) In case G_0 is acyclic and the shadow of G_0 is a tree, the average rate of any node $p \in N$ is 1/2.

4.8 Related Work

Further reading: The $GJ_{\mathrm{Imp}}(J)$ module was developed in the DARTS project[9] to solve the tick generation problem (see next chapter). The work by Függer *et al.* [36, 41] contains its formal analysis dedicated to the specific tick generation problem and stated in terms of a predecessor formal model. Further, [41] contains a description of a physical gate-level implementation of a $GJ_{\mathrm{Imp}}(J)$ module. In [29] implementation details are discussed to a greater extent and a field programmable gate array (FPGA) implementation is presented.

[8] A joint paper of the author with Bernadette Charron-Bost, Jennifer Welch, and Josef Widder is submitted for review at the time of writing.

[9] see *ti.tuwien.ac.at/ecs/research/projects/darts*

In [35], alternatives to $GJ_{\text{Imp}}(J)$ were investigated and their timing constraints analyzed. In [24, 37] part of the system model as well as a specification and formal analysis of an adapted $GJ_{\text{Imp}}(J)$ module have been published. Again, both with respect to solving the tick generation problem. The adaption consists of a further generalization of (4.6) which restricts the threshold k_i^{th}, $k \geqslant 1$, to an offset i in $\{0, 1\}$. In [24, 37], it was shown how to implement General Join modules with arbitrary offset $i \geqslant 0$. Fuchs' thesis [32] contains a thorough investigation of the implementation of a system of $GJ_{\text{Imp}}(J)$ modules for solving the tick generation problem. An approach to measure the performance of some (fault-tolerant) clockless control structures introduced in this chapter using min-max-plus algebra [45, 47] was presented by Függer *et al.* in [38].

Existing work: Different approaches for designing fault-tolerant clockless circuits are described in literature. Most of them, however, either deal with finding defects after fabrication, typically gate level stuck-at faults[10], or tolerating single event upsets (SEU) happening during mission time. Those, however, do not permanently destroy a circuit's components.

Verdel and Makris [95] propose a technique to detect errors in clockless circuits, whose components communicate by 4-phase handshaking. Their idea is to duplicate the D-elements that perform the 4-phase handshaking, and compare their output signals. However, the point in time of performing the comparison for equality is not clear in clockless circuits, unlike in synchronous circuits. The presented error detection circuit thus tolerates inequality of the output signal status of duplicated D-Elements within timing windows of predefined length, and raises an error signal otherwise. Clearly, though, detailed knowledge of the time window length is required. Further, no permanent faults, but only faults leading to soft-errors can be tolerated by resetting (part of) the circuit after a detected error.

LaFrieda and Manohar [52] present a method to make QDI circuits less susceptible to faults and, further, to map faults to stuck-at faults where possible. The latter are easier to detect than other faults as they typically result in deadlock after some time. A method proposed in their work is to duplicate all synchronization channels forming system's control structure (yielding a top and a bottom channel) and let each handshaking component not only perform handshaking with its predecessor and successor components, but also with its duplicated instance. Unfortunately, however, this introduces significant additional delay: because of the handshaking between the duplicated top and bottom components, they do not communicate in parallel, but one after the other with their predecessor and successor components. Further, this approach does not allow to tolerate permanent faults.

Monnet *et al.* [71] consider hardening of QDI circuits and propose duplicating the data path and crosschecking its memory cells (C-gates). In case there is a memory mismatch an alarm is triggered. Clearly, however, this method is restricted to SEUs only, that is, does not allow to tolerate permanent faults.

Jang and Martin [49] proposed a simple and efficient method to tolerate soft-errors. They transform a gate G into a soft-error tolerant gate, by duplicating it and introducing synchronizing cross-coupled C-Elements after both duplicates of G, say G_1 and G_2.

[10] A stuck-at fault at port x at some time t results in a constant 0 or 1 status of x from t on. Note that this may produce an extra transition at time t.

Thereby each C-Element has its inputs connected to the outputs of both G_1 and G_2. In case a spurious transition occurs on either G_1's or G_2's output, or in case a transition is lost, both C-Elements do not propagate the gate's outputs, until they both match again. Thereby, the technique makes use of the stability property of correctly designed QDI circuits: both gates' production rules remain enabled until they are acknowledged. Thus, after the spurious pulse has decayed, the faulty gate, whose production rule is still enabled by stability, will change its output to the correct output, resulting in matching outputs of G_1 and G_2. Now the output can propagate, possibly leading to a deactivation of the gates' production rules. While this method is powerful and has successfully been used to implement a soft-error tolerant microcontroller [50], again, it does not allow to tolerate permanent faults.

OR causality, which is central in the specification of the presented General Join module, has also been investigated by the asynchronous VLSI community, albeit with a focus of early decision joins: a join module with $\geqslant 2$ predecessor modules that receives data from one of its predecessor modules, which makes the reception of data from the other predecessor modules superfluous (the first data allows for an early decision), is not required to wait for these before handing over the data to its successor module. In order, however, not to loose synchrony with the other modules, the early deciding join generates an anti-token at all its inputs corresponding to the other predecessor modules. In case a data item (represented by a token) and an anti-token collide, they cancel each other. Typically, control structures are extended by structures that propagate anti-tokens (against the token direction) to predecessor modules.

As already stated in Section 3.9, Yakovlev *et al.* [98] investigated OR causality and separated it into joint and disjoint OR causality. In [10], Bystrov *et al.* extend this classification by the *slack*: consider two ports a and b whose transitions cause a third port c to make a transition by means of joint or disjoint OR causality. Then slack is the maximum number of transitions, the two ports a and b can be off at any time $t \in \mathcal{T}$. As an example consider ports a and b, whose transitions cause transitions of port o with joint OR causality. In case the slack is 1, the k^{th}, $k \geqslant 1$, occurrence of a transition of either a or b whichever first occurs (say a) causes a transition of o. However, before the $(k+1)^{\text{th}}$ transition of a can happen, b must make a transition. Typically, slack is enforced by handshaking with the components with output port a and b, respectively. In [10], implementations of a component (called a joint merge) with joint OR causality of its input ports with respect to its output port are presented: one with bounded slack and one with "almost" unbounded slack. While the solution with bounded slack cannot be used in fault-tolerant circuits whose components may permanently fail (leading to deadlock in case of bounded slack), the "almost" unbounded slack joint merge module could be used as an implementation for a General Join module as presented in the thesis.[11] However, here, we allow faulty components to act arbitrarily, making it unnecessary to keep synchronization with these. Thus, we do not profit from the benefits of the "almost" unbounded joint merge and can

[11] Clearly, the OR semantics, that is, waiting for the first transition, had to be changed to a more general waiting for the k^{th} transition, with $k \geqslant 1$.

use the simpler $GJ_{\mathrm{Imp}}(J)$ implementation.

Another approach making use of joint OR causality has been described in [21]. In [22] Cortadella *et al.* proposed to use elastic handshaking protocols between synchronously clocked pipeline stages. Later, in [21] the design method was extended to allow not only for tokens, but also for anti-tokens flowing into the opposite direction. The idea is to double the pipelines and add combinatorial logic between corresponding pipeline stages, such that a token and an anti-token annihilate each other. Unfortunately, the possibility of metastability during normal operation[12] and the central clock are prohibitive for use in fault-tolerant systems.

Another approach intended to speed-up asynchronous circuits by early evaluation and using anti-tokens is proposed by Brej [8]. The idea is as follows: in case a module M does not need a predecessor module's data any more, it may acknowledge and thereby disable the predecessor module's request port at any time. The approach, however, is susceptible to produce glitches on the request port, by race conditions between the predecessor enabling it and M disabling it. This may lead to metastability during normal operation, unless a detailed timing analysis and optimization are carried out.

Ampalam and Sigh [1] presented building blocks (stage, fork, join) that support circulation of tokens and anti-tokens. The blocks can be connected with each other to form an asynchronous circuit that supports early evaluation and preemption of superfluous computations by feeding back the anti-tokens. While this method is close to the one presented in this thesis, there are significant differences: unlike [1] we do not excessively make use of stages that can propagate and merge tokens and anti-tokens. In fact, the Diff-gate can be seen as such a merging gate (of far smaller size), which merges tokens stored in the remote pipeline with anti-tokens stored in the local pipeline. This approach (i) allows to store tokens and anti-tokens locally at the General Join module (which is necessary when building fault-tolerant circuits), rather than propagate them back to the predecessor modules and (ii) yields shorter token passage times (and by this higher throughput) through the remote pipelines of the General Join, since inside the General Join conventional pipeline stages are used instead of merging pipeline stages.

[12] Unless conservative clock periods are used such that under normal operation, transient effects have decayed before the clock transition occurs.

Chapter 5

Fault-tolerant Tick Generation

In this chapter an application of the framework from Chapters 3 and 4 is presented. The application is both sufficiently simple in order not to distract from the major points but complex enough to show the applicability for practical systems.

Because of physical environmental effects like ionizing particle hits inducing current on one of their ports [4, 100], components of digital systems are subject to faults. In order to protect a system from failing due to a faulty component, different mitigation techniques exist on different levels of abstraction: well-known methods to obtain fault-tolerant systems range from sizing transistors [99] and layouting techniques (see [86] for an example application) for making cells less susceptible to radiation faults to introducing error correction mechanisms for memory and computation by redundant logic, just to name a few. A typical approach at system level is to replicate a single module n times and let the replica communicate via channels in order to remain consistent in their state. In the following we will refer to the i^{th}, $1 \leqslant i \leqslant n$, module as *functional unit* Fu_i. The problems arising from this replication have been extensively studied for classic distributed system models in the past decades, see e.g., [56, 83]. It is well known that solving important problems arising in this context (like the consensus problem) require a certain degree of synchrony on the computations performed by the replicated modules [30]. Synchrony is thereby typically obtained by restricting message and computation delays. Distributed algorithms that directly rely on these timing constraints are usually complex. A commonly used approach thus is (i) to provide a synchronized notion of time to the functional units Fu_i and (ii) run algorithms that maintain a consistent global state of the Fu_i's on top of the synchronized functional units. In case of a digital system, a notion of time is typically provided by a clock signal to a functional unit. A measure of how good any two clock signals are synchronized can be defined by different means: (i) By its *skew*, that is the maximum difference in real-time two clock signals produce their k^{th} transition, over all $k \geqslant 1$. (ii) By a less accurate synchrony measure, their *precision*, that is the maximum difference in the number of transitions of every two clock signals at some real-time t, over all $t \in \mathcal{T}$.

One possible approach to supply a set of replicated functional units with synchronized clock signals is to provide a global clock signal from a single oscillator to all functional units. Although becoming more and more difficult due to decreasing feature sizes and increasing transistor densities, there exist clock tree engineering techniques by which the maximum skew can be minimized. Clearly, however, these designs bear a single point of failure, namely the oscillator itself.

An alternative approach without this weakness is to replicate the clock sources, too, one for each functional unit. It remains to specify the means by which the functional units communicate with each other: (i) In case of asynchronous communication, we arrive at the well known GALS (globally asynchronous, locally synchronous) designs [13]. (ii) In case of synchronous communication, an underlying fault-tolerant clock synchronization protocol must be available, which ensures that the functional units' local clocks do not arbitrarily drift apart, which would lead to arbitrarily large skews between corresponding (k^{th}) clock transitions. Unfortunately, both solutions (i) and (ii) are susceptible to metastability during communication, even if no single component is faulty.

Fortunately there is a further possibility to provide the functional units with a local clock signal: to generate the clock signal in a distributed manner. The idea is to make the k^{th} clock transition of a correct functional unit's clock at about the same time all other correct functional units generate the k^{th} clock transition of their clock signals. We will call this problem the *tick generation problem*. An interesting question thus is to find a set of modules that solve the tick generation problem in the presence of faults and prove them correct.

In this chapter it will be shown that a module comprising of (i) a set of General Join modules, with identical, well-chosen thresholds $\langle \alpha_0^{\text{th}}, \beta_1^{\text{th}} \rangle$ and (ii) a set of channels interconnecting all these General Join modules solves the tick generation problem, in spite of a fraction of Byzantine faulty general join modules. The proposed architecture is called DARTS and was developed in the DARTS project[1] in our group.

5.1 The DARTS Architecture

As shown in Figure 5.1, the basic idea of DARTS is to replace the common quartz oscillator and the clock tree by a fully distributed GALS-like approach [13]: every functional unit Fu_i has attached a dedicated fault-tolerant tick generation node (TG node) here, which generates Fu_i's local clock signal. To accomplish this, all TG nodes communicate with each other over a simple "network" by broadcasting their clock signals (TG-Net). In contrast to standard GALS, however, DARTS ensures that the local clock signals of different Fu's are synchronized to each other to within a few clock cycles (sometimes called multi-synchronous clocking [85,94]). In [78], Polzer *et al.* have shown that even such loose synchrony suffices for implementing metastability-free high-speed communication between

[1] ti.tuwien.ac.at/ecs/research/projects/darts/

different Fu's. Thus SoCs built atop of DARTS do not need asynchronous communication mechanisms.

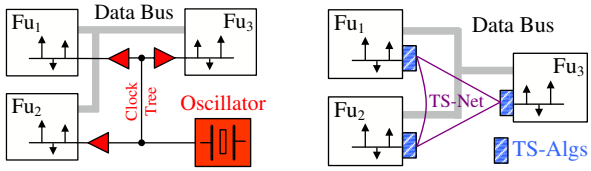

Figure 5.1: Replacing synchronous clocking by fault-tolerant distributed tick generation.

DARTS clocks (patented in [82]) provide a number of additional advantages, which makes them particularly promising for critical applications in the aerospace domain: First of all, the approach entirely circumvents quartz oscillators, which are fairly big and sensitive devices (shock, vibration, temperature etc.), as well as the cumbersome clock tree engineering issue [6, 31, 70, 79]. It is fault-tolerant, in the sense that clock signals supplied by correct TG-Algs are not affected by transient and permanent failures occurring in other TG-Algs and/or in the TG-Net. DARTS clocks can also be guaranteed to indeed start operating at booting time, a feature that is crucial to ensure for oscillator-based clocking approaches in difficult to maintain applications, like space applications. Moreover, the clock always runs at the maximum speed and adapts to the current operating conditions. And last but not least, since different Fu's are driven by slightly different clock signals, DARTS clocks alleviate EM radiation and ground bouncing problems [66] that typically plague devices using synchronous clocking.

We will next describe the idea of the TG-Alg's algorithm: the TG-Algs derive from a simple synchronizer algorithm for the Θ-Model [57, 96, 97] introduced in [97]. Consider a system of $n \geqslant 1$ nodes in a classical message-passing system, where nodes make atomic receive-compute-send steps whenever they receive a message. Further consider the code shown in Figure 5.2.

```
1: /* Initialization */
2: VAR k : integer := 0;
3: initially send tick(0) to all [once];
4:
5: if received tick(k) from all n processes then
6:     send tick(k + 1) to all [once]; k := k + 1;
7: end if
```

Figure 5.2: Simple algorithm for generating approximately simultaneous tick(k) messages.

Assume that all processes are correct and boot at time 0. Processes are connected by a reliable point-to-point message-passing network: no spurious messages are ever generated, no messages are lost or altered, and all messages sent at time t are eventually received at

some time in $t + [\tau^-, \tau^+]$, where τ^- resp. τ^+ denote the (possibly unknown) lower resp. upper bound on the end-to-end delay of messages exchanged between processes.

All processes will send tick(k), $k \geqslant 0$, within $\tau^+ - \tau^-$ of each other and thereby generate synchronized ticks: assume that a process sends tick(k), $k \geqslant 1$, at time $t_k \in \mathcal{T}$. Then it must have received tick($k-1$) from all processes by time t_k. Thus, all processes must have sent tick($k-1$) by time $t_k - \tau^-$ and all processes will receive tick($k-1$) by time $t_k - \tau^- + \tau^+$. Thus all processes will send tick(k) by then. The proposition follows. Clearly, however, the above algorithm fails to make progress in case of even a single crashed process that stops sending ticks after some point in time.

A fault-tolerant variant of this algorithm, whose core is based on Srikanth & Toueg's consistent broadcasting primitive [88], is shown in Figure 5.3. It again assumes a message-driven system of $n = 3f + 1$ processes (i.e. TG-Alg instances), but this time, at most f of those may behave arbitrarily faulty, i.e., Byzantine. The processes are connected by a reliable[2] point-to-point message-passing network (i.e. the TG-Net) with the same restrictions as before: there are no spurious, lost or altered messages and all messages sent from correct nodes to correct nodes are received within $[\tau^-, \tau^+]$. Let $\varepsilon = \tau^+ - \tau^-$ be the maximum uncertainty of the message delay, and $\Theta = \tau^+/\tau^-$ the maximum delay ratio.

```
 1: /* Initialization */
 2: VAR k : integer := 0;
 3: initially send tick(0) to all [once];
 4:
 5: if received tick(ℓ) from at least f + 1 distinct processes with ℓ > k then
 6:     send tick(k + 1), ..., tick(ℓ) to all [once]; k := ℓ;
 7: end if
 8: if received tick(k) from at least 2f + 1 distinct processes then
 9:     send tick(k + 1) to all [once]; k := k + 1;
10: end if
```

Figure 5.3: Fault-tolerant algorithm for generating approximately simultaneous tick(k) messages.

The algorithm works as follows: initially, every process broadcasts tick(0) in `line 3`. If a correct process p receives $f + 1$ tick(ℓ) messages (`line 5`), it can be sure that at least one of those was broadcast by a correct process. Therefore, p can safely catch up and send tick($k + 1$), ..., tick(ℓ). If some process p receives $2f + 1$ tick(k) messages (`line 8`) and thus send tick($k + 1$), one can be sure that all tick(k) messages broadcast by correct processes, i.e., at least $f + 1$, will be received within ε by every other correct process. Hence, every correct process will execute `line 5` and send tick(k), if it has not already done so. Consequently, at most τ^+ later, every other correct process q will receive $2f + 1$ tick(k) messages and thus send tick($k + 1$).

[2]Note that this reliable network assumption is not unduly restrictive, since communication failures can be mapped to failures of the sending process.

In conjunction with the fact that the fastest correct process cannot send consecutive tick(k), tick($k+1$), ... arbitrarily fast, this implies a bound on the synchronization precision: The detailed analysis will reveal that correct nodes generate a sequence of consecutive messages tick(k), $k \geqslant 1$, in a synchronized way: if $\#b_p(t)$ denotes the number of tick(k) messages broadcast by process p by real-time t (which is identical to the value of variable k at real-time t, cf. Figure 5.3), it will turn out that $(t_2 - t_1)\alpha_{\min} \leqslant \#b_p(t_2) - \#b_p(t_1) \leqslant (t_2 - t_1)\alpha_{\max}$ for any correct node p and $t_2 - t_1$ sufficiently large ("accuracy"); the constants $\alpha_{\min}$ and $\alpha_{\max}$ depend on τ^-, τ^+. Moreover, every two correct nodes p, q maintain $|\#b_p(t) - \#b_q(t)| \leqslant \pi$ ("precision"), for a small constant π that depends on Θ only. Note carefully that the algorithm automatically adapts to the instantaneous timing characteristics of any involved computation and communication.

Since the algorithm in Figure 5.3 looks very simple, it is tempting to conclude that it is easily translated into a hardware description language: Node p's TG-Alg just needs to drive a boolean-valued clock signal, where it outputs the k-th signal transition when the algorithm sends its tick(k) message; the TG-Net is formed by feeding all clock signals to all TG-Algs. It turns out, however, that a number of challenging issues must be solved to actually accomplish this:

- *How to implement the TG-Net efficiently?*

  The algorithm assumes a fully connected network, consisting of n^2 links,[3] so anything beyond a single wire per link is unacceptable [35]. Moreover, for implementation simplicity and performance, the information transmitted via the TG-Net must be kept to a minimum. Ideally, and almost mandatory, the TG-Net should just feed the emitted clock ticks, i.e., signal transitions, of every TG-Alg to every other TG-Alg.

- *How to adapt the original algorithm for zero-bit messages?*

  By just sending signal transitions, no information except the occurrence time can be conveyed over the TG-Net. Thus, the tick number k contained in the messages of Figure 5.3 must be maintained at every receiver, individually for every sender.

- *How to map hardware faults to process failures?*

  Given that the algorithm shown in Figure 5.3 tolerates Byzantine failures, we are on the safe side here. Interestingly, there is evidence [43] that assuming less severe failures is inappropriate in the presence of real hardware faults: Even simple stuck-at faults could produce early and/or inconsistently perceived signal transition, and cannot hence be modeled as a crash or omission process failure. More severe hardware faults, like delay faults or early/spurious clock transitions induced e.g. by particle hits or crosstalk, can also easily lead to Byzantine failures.

[3] Note that a bus of n broadcast links that provide every TG-Alg with the messages from all other TG-Algs is in fact sufficient here.

- *How to ensure atomicity of actions in a VLSI implementation?*

  This turned out to be the most demanding challenge, which actually triggered this thesis: All fault-tolerant distributed computing models assume atomic computing steps at the level of a single process. For example, the algorithm presented in Figure 5.3 assumes that: (i) messages are received, (ii) the number of received messages is checked with respect to a threshold, and (iii) possibly a new message is broadcast and variable k is updated; all this happens in one atomic computing step. This abstraction does not apply when the algorithm is implemented via asynchronous digital logic gates, which concurrently and continuously compute their outputs based on their inputs. Explicit synchronization (serialization of actions/interlocking) must be introduced if two local computations must not interfere with each other.

Recalling Chapter 4, it should become clear that the algorithms stated in Figure 5.2 and 5.3 are just abstract (zero-time) descriptions of the behavior of General Join modules: the first algorithm corresponds to a General Join module J with $\Theta(J) = n_0^{\text{th}}$ and the second algorithm corresponds to a General Join J with $\Theta(J) = \langle (2f+1)_0^{\text{st}}, (f+1)_1^{\text{st}} \rangle$. It is exactly this correspondence which will allow us to elegantly express and prove correct the DARTS solution for the tick generation problem.

In the following we will describe the distributed system under consideration as well as the assumed failure model in detail.

5.1.0.6 Distributed system

The distributed system is a compound module comprising of (i) a set P of TG nodes, where $|P| = n = 3f + 2$, for some $f \geqslant 0$, and (ii) the interconnection formed by the TG-Net. Each TG node p in P has n input ports $\text{req}_{p,q}$, one for each q in $P \setminus \{p\}$, and 1 output port, b_p, over which it broadcasts its locally generated clock signal. Each output port b_p, p in P, is connected to exactly one input port, namely $\text{req}_{q,p}$, of each remote TG node q in $P \setminus \{p\}$, via a channel with delay bounds $\tau_{rem}^{\pm}$. The channels are forming the TG-Net and are all initialized to 0.

5.1.0.7 Failure model

We partition our system into multiple fault-containment regions (FCRs) as follows: let FCR p in P consist of the single TG node p together with all its outgoing channels, as depicted in Figure 5.4. Since every FCR is associated with exactly one TG node, we will also use these terms interchangeably. As already stated in Chapter 3, we assume each of the FCRs to fail in an arbitrary way: the adverse power of Byzantine failures in our context lies in the ability of a faulty TG node to generate wrong clock ticks (early timing failures or even spurious) that are perceived inconsistently at different remote TG nodes.

Recapitulate from Chapter 3 that the set of FCRs is partitioned in to the correct FCRs C and the faulty FCRs F. We will prove that there exist modules with the same input and

output ports as the TG nodes, which solve the tick generation problem specified below in Section 5.1.1 in the presence of up to f Byzantine faulty FCRs, provided that the total number of TG nodes is $n \geqslant 3f+2$. Note that this is slightly more than the required lower bound of $n \geqslant 3f+1$ for clock synchronization [25], but facilitates a considerably better precision and accuracy (attained by counting only remote messages when calculating the $f+1$ resp. $2f+1$ thresholds; including self-reception would lead to $\tau^-_{rem} = \tau^-_{loc}$ in Theorem 5.2 and hence spoil the achievable worst-case precision).

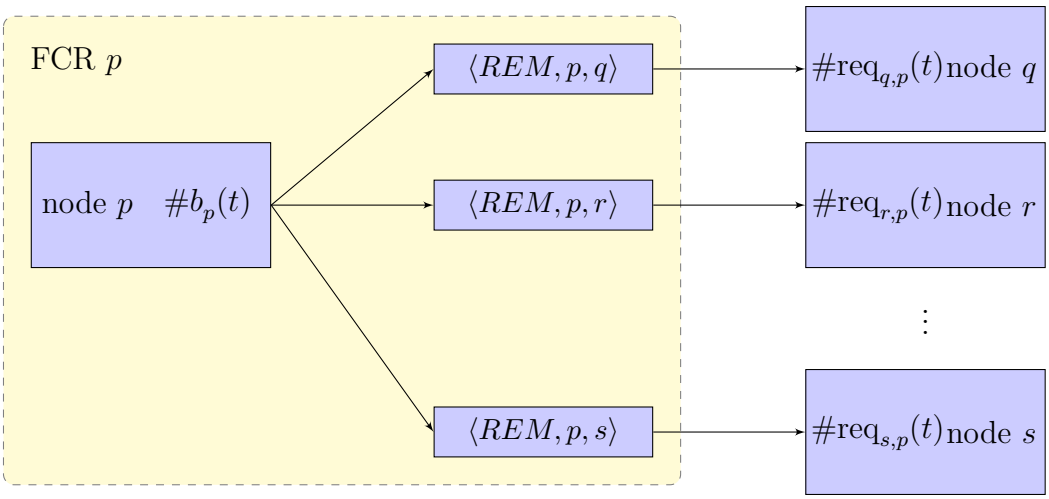

Figure 5.4: Fault-containment region for TG node p

5.1.0.8 Booting

We assume that the whole system is simultaneously reset at time $t = 0$. However, we allow the modules to complete booting at (slightly) different times: If $t_{p,b}$ denotes the time by which all modules in a correct FCR p in P have completed booting, we require that $t_{p,b} \in [0, B]$ for some constant $B \leqslant \tau^-_{rem}$. The latter condition ensures that messages sent by p are never lost at a different correct module q in P because of late booting.

5.1.1 Tick Generation Problem

We will next formally state the tick generation problem: in all executions complying to the system and failure model introduced in the previous sections, the set P of TG nodes must guarantee the following properties:

(P) **Precision:** There is a constant π, such that for every pair of correct TG nodes $p, q \in C$:

$$\forall t \in \mathcal{T} : \ |\#b_q(t) - \#b_p(t)| \leqslant \pi. \tag{5.1}$$

(A) **Accuracy:** There are constants R^-, O^-, R^+, $O^+ > 0$, such that for every correct TG node $p \in C$:

$$\forall t_1, t_2 \in \mathcal{T}, t_2 \geqslant t_1 : \; O^-(t_2 - t_1) - R^- \leqslant \#b_p(t_2) - \#b_p(t_1) \leqslant O^+(t_2 - t_1) + R^+. \quad (5.2)$$

Informally, the *precision requirement* (P) just states that the difference of the number of clock ticks generated by any two different correct TG nodes is bounded, whereas the *accuracy requirement* (A) guarantees some relation of the progress of the clock ticks with respect to the progress of real-time. Note that (A) is also called *envelope requirement* in literature, and effectively bounds the frequency of the generated clock ticks.

5.1.2 A Solution with General Join modules

A *solution* of the tick generation problem is an assignment of modules to TG nodes P, such that the obtained distributed system solves the tick generation problem in spite of up to f faults. We will next present two solutions: a high-level and a low-level one.

5.1.2.1 TG Node Implementation

High-level solution. The proposed solution is fairly simple: let each TG node p, p in P be a General Join module p with thresholds $\Theta(p) = \langle (2f+1)^{\text{st}}_0, (f+1)^{\text{st}}_1 \rangle$. A correct solution additionally has to fulfill Constraint 3, defined later on.

Low-level solution. In a second step, we will give a more refined module of a TG node p. Since we know how to implement a General Join with input channels in terms of even simpler building blocks, namely by module $GJ_{\text{Imp}}(J)$ with input channels, we can simply let each TG node p in P, be a $GJ_{\text{Imp}}(J)$ module. This solution is well suited to be mapped to physical hardware. To underline the importance of $GJ_{\text{Imp}}(J)$ modules being a solution for the tick generation problem, we will also call module $GJ_{\text{Imp}}(J)$ with parameters $\alpha = 2f+1$ and $\beta = f+1$ a *TG-Alg* module.

Clearly, for a physical implementation of a TG-Alg, we have to further restrict module $GJ_{\text{Imp}}(J)$: up to now $GJ_{\text{Imp}}(J)$ comprises of a set of $+/-$ counters and we assumed $+/-$ counters to have unbounded capacity. To get rid of this unduly restrictive assumption, we require the following *size requirement* (S) to hold for all executions of the low-level solution complying to the system and failure model.

(S) **Size:** There are constants S_{rem} and S_{loc}, such that for every pair or correct TG nodes $p, q \in C$;

$$\forall t \in \mathcal{T} : \; \#s^{loc}_{p,q}(t) \leqslant S_{loc} \text{ and } \#s^{rem}_{p,q}(t) \leqslant S_{rem}.$$

In the following section, we will show that the distributed system with TG-Algs as TG nodes indeed satisfy the above properties (P), (A) and (S) in all executions complying to

our system and failure model, provided that the already introduced Constraints 1–2 and the additional Constraint 3, introduced later, hold. Our Theorems 5.2, 5.3, 5.4 and 5.6 will also establish numerical values for all the constants introduced in (P), (A) and (S), which only depend on the delay parameters of the TG-Net and those introduced in the specifications of the TG-Alg basic modules.

5.2 Correctness Proofs

Unless otherwise stated, we consider the *high-level solution* first. Its correctness proof can be partitioned into three layers of abstraction. The *bottom proof layer* is concerned with deriving basic results on how a single TG node behaves. We thereby heavily make use of the results derived for General Join modules and restate them in terms of the given distributed system.

The *intermediate proof layer* builds upon the results derived in the bottom proof layer. Its main result is the proof that a set of global properties, so called synchronization properties, hold for the distributed system.

Finally, properties (P) and (A) are derived from the synchronization properties in the *top proof layer*.

In a second step, we will show that (P) and (A) also hold for the low-level solution, and prove that (S) also holds for the latter.

5.2.1 Bottom Proof Layer

We first prove a minimum time between two ticks sent by a correct TG node:

Lemma 5.1. *If correct node p in C sends tick $k \geqslant 1$ at time $t_{p,k}$, then it cannot send tick $k+1$ before $t_{p,k} + T_{min}$.*

Proof. Since node p is a General Join, the result follows from Corollary 4.1. ■

Next, we derive, that the $2f+1$ threshold performs its duty:

Lemma 5.2. *For all correct nodes p in P and ticks $k \geqslant 1$: if there exists a set $Q \subseteq C$ of correct nodes with $|Q| \geqslant 2f+1$, such that, for all $q \in Q$:*

(i) for all $q \in Q$, q sends tick k by $t_{Q,k}$,

(ii) p sends tick k at $t_{p,k}$,

then p sends tick $k+1$ by time $\max\{t_{p,k} + \tau^+_{loc}, t_{Q,k} + \tau^+_{rem}\} + \tau^+_{Diff} + \tau^+_{PC} + \tau^+_{TH}$.

Proof. Node p and nodes q in Q all are General Joins with threshold $\langle (2f+1)_0^{\text{st}}, (f+1)_1^{\text{st}} \rangle$. Since all nodes q in Q are correct, node p receives transition k from each node q by time $t_{Q,k} + \tau_{rem}^+$. Since further $|Q| \geqslant 2f+1$, we may apply Lemma 4.2 and obtain that p sends transition $k+1$ by time $\max\{t_{p,k} + \tau_{loc}^+, t_{Q,k} + \tau_{rem}^+\} + \tau_{Diff}^+ + \tau_{PC}^+ + \tau_{TH}^+$. The lemma follows. ■

Further, the $f+1$ threshold performs its duty:

Lemma 5.3. *For all correct nodes p in C and ticks $k \geqslant 1$: If there exists a set $Q \subseteq C$ of correct nodes with $|Q| \geqslant f+1$, such that,:*

(i) for all $q \in Q$, node q sends tick $k+1$ by $t_{Q,k+1}$, and

(ii) p sends tick k at $t_{p,k}$,

then p sends tick $k+1$ by time $\max\{t_{p,k} + \tau_{loc}^+ + \tau_{Diff}^+, t_{Q,k+1} + \tau_{rem}^+\} + \tau_{PC}^+ + \tau_{TH}^+$.

Proof. The proof follows the proof of Lemma 5.2 in large parts: node p and nodes q in Q all are General Joins with threshold $\langle (2f+1)_0^{\text{st}}, (f+1)_1^{\text{st}} \rangle$. By the correctness of the nodes in Q, node p receives transition $k+1$ from each node q by time $t_{Q,k} + \tau_{rem}^+$. Since further $|Q| \geqslant f+1$, we may apply Lemma 4.3 and obtain that p sends transition $k+1$ by time $\max\{t_{p,k} + \tau_{loc}^+ + \tau_{Diff}^+, t_{Q,k} + \tau_{rem}^+\} + \tau_{PC}^+ + \tau_{TH}^+$. The lemma follows. ■

5.2.2 Intermediate Proof Layer

Based on the results of the bottom proof layer, we can now establish elementary synchronization properties of the ticks generated by different correct nodes. The following Theorem 5.1 corresponds to well-known classic results on consistent broadcasting [57,80,88,97], which are expressed and proved in our new modeling framework. For the theorem to hold, Constraint 3 must hold, which essentially guarantees that even the slowest local channel is faster than the fastest remote channel.

Constraint 3. [C3] *With T_{first}^- defined by*

$$T_{first}^- := \tau_{rem}^- + \tau_{Diff}^- + \tau_{PC}^- + \tau_{TH}^-, \tag{5.3}$$

the relation $T_{first}^- \geqslant \tau_{loc}^+ + \tau_{Diff}^+ + \tau_{PC}^+ + \tau_{TH}^+$ must hold.

Theorem 5.1 (C3, but only in (QS)). (Synchronization Properties). *The algorithm satisfies the synchronization properties Progress (P), Unforgeability (U), Quasi-Simultaneity (QS) and Booting-Simultaneity (BS), if $n \geqslant 3f+2$ holds.*

(P) *Progress. If all correct nodes send tick $k \geqslant 1$ by time t, then every correct node sends at least tick $k+1$ by time $t + T_P$, with*

$$T_P := \max\{\tau_{loc}^+, \tau_{loc}^+\} + \tau_{Diff}^+ + \tau_{PC}^+ + \tau_{TH}^+. \tag{5.4}$$

(U) *Unforgeability. If no correct node sends tick $k \geqslant 1$ by time t, then no correct node sends tick $k+1$ by time $t + T^-_{first}$ or earlier, where T^-_{first} is given by* (5.3).

(QS) *Quasi-Simultaneity. If some correct node p sends tick $k+1 \geqslant 2$ by time t, then every correct node (p included) sends at least tick k by time $t + T_{QS}$, with*

$$T_{QS} := \max \left\{ \begin{array}{l} (\tau^+_{rem} - \tau^-_{rem}) - \tau^-_{Diff}, \\ B - T^-_{first} \end{array} \right\} + (\tau^+_{PC} - \tau^-_{PC}) + (\tau^+_{TH} - \tau^-_{TH}). \tag{5.5}$$

(BS) *Booting-Simultaneity. If some correct node sends tick $k \geqslant 1$ by time t, then every correct node sends at least tick k by time $t + T_{BS}(k)$, with*

$$T_{BS}(k) := B + (\tau^+_{PC} - \tau^-_{PC}) + (\tau^+_{TH} - \tau^-_{TH}) + (T_P - T^-_{first})(k-1). \tag{5.6}$$

We will show the properties Progress (P), Unforgeability (U), Quasi-Simultaneity (QS) and Booting-Simultaneity (BS) one after the other.

Progress (P)

Proof. Assume that all correct nodes C, with $|C| \geqslant 2f+2$, sent tick $k \geqslant 1$ by time t. Now focus on a correct node $p \in C$: We can apply Lemma 5.2 with $Q = C \setminus \{p\}$ and $t_{p,k} = t_{Q,k} = t$. Thus, p must send tick $k+1$ by

$$\begin{aligned} t' &= \max\{t_{p,k} + \tau^+_{loc}, t_{Q,k} + \tau^+_{rem}\} + \tau^+_{Diff} + \tau^+_{PC} + \tau^+_{TH} \\ &= t + \max\{\tau^+_{loc}, \tau^+_{rem}\} + \tau^+_{Diff} + \tau^+_{PC} + \tau^+_{TH} \\ &= t + T_P. \end{aligned}$$

The lemma follows. ■

Unforgeability (U)

Proof. Let p be the first correct node that sends tick $k+1 \geqslant 2$ at time $t_{p,k+1}$. Since p is a General Join, we can apply Lemma 4.4 and consider its two possible implications (i) and (ii), one of which must hold:

(i) Since $|Q| \geqslant 2f+1$, there must be a subset $C' \subseteq Q$ of correct nodes of size $|C'| \geqslant f+1$. Clearly, it must hold that, for $t' = t_{p,k+1} - (\tau^-_{Diff} + \tau^-_{PC} + \tau^-_{TH} + \tau^-_{loc})$ and for all r in C',

$$\#\mathrm{req}_{p,r}(t') \geqslant k.$$

By the remote channel properties, this implies $\#b_r(t'') \geqslant k$ with

$$t'' := t' - \tau^-_{rem} = t_{p,k+1} - T^-_{first},$$

i.e., r — and by this the first correct node which sent tick k — has sent tick k by $t_{p,k+1} - T^-_{first}$. The lemma follows.

(ii) Since $|Q| \geqslant f+1$, there must be at least one correct node $r \neq p$ among Q for which

$$\#\text{req}_{p,r}(t') \geqslant k+1,$$

with $t' = t_{p,k+1} - \tau_{TH}^{-} - \tau_{PC}^{-}$. For r, this implies $\#b_r(t') \geqslant k+1$ — a contradiction to the assumption that p was the first correct node to send tick $k+1$. The lemma follows.

The lemma follows in both cases. ■

Before turning to the proof of (QS), we proceed with two technical lemmas.

Lemma 5.4. *If the first correct node p sends tick $k+1 \geqslant 2$ at time $t_{p,k+1}$, then at $t' := t_{p,k+1} - \tau_{TH}^{-} - \tau_{PC}^{-} - \tau_{Diff}^{-}$ it must hold that there exists a set $Q \subseteq P \setminus \{p\}$ of size $|Q| \geqslant 2f+1$, such that*

$$\forall q \in Q : \#req_{p,q}(t') \geqslant k.$$

Proof. Analogous to the proof of Unforgeability (U), we apply Lemma 4.4 and consider its two possible implications (i) and (ii):

(i) This case exactly matches the implication of our lemma. The lemma follows.

(ii) Since, $|Q| \geqslant f+1$, there must be at least one correct node $r \neq p$ among Q that has already sent tick $k+1$ before p did; this contradicts the assumption that p is the first correct node to send tick $k+1$. The lemma follows.

The lemma follows in both cases. ■

Lemma 5.5. *Let t_b be the time when the first correct node boots. Let $t_{first,1}$ (respectively $t_{last,1}$) be the time the first (respectively last) correct node sends tick 1. Then,*

$$t_{first,1} \geqslant t_b + \tau_{PC}^{-} + \tau_{TH}^{-}, \text{ and}$$
$$t_{last,1} \leqslant t_b + B + \tau_{PC}^{+} + \tau_{TH}^{+}.$$

Proof. Correct nodes can boot only within $t_b + [0, B]$ by assumption. Further, from Lemma 4.1, we know that a correct node p that boots at time $t_{b,p}$ sends transition 1 within $t_{b,p} + \tau_{PC}^{\pm} + \tau_{TH}^{\pm}$. The lemma follows. ■

Quasy Simultaneity (QS)

Proof. The proof is by induction on the number of ticks $k+1 \geqslant 2$ sent by the first correct node.

Begin $(k+1=2)$: By Lemma 5.5, the first correct node must send tick 1 at $t_{first,1}$ with

$$t_{first,1} \geqslant t_b + \tau^-_{PC} + \tau^-_{TH}.$$

Let $t_{first,2}$ be the time when the first correct node sends tick 2. By Unforgeability (U),

$$\begin{aligned} t_{first,2} &\geqslant t_{first,1} + T^-_{first} \\ &\geqslant t_b + \tau^-_{PC} + \tau^-_{TH} + T^-_{first}. \end{aligned}$$

By Lemma 5.5, all other correct nodes must send tick 1 by $t_{last,1}$ with

$$t_{last,1} \leqslant t_b + B + \tau^+_{PC} + \tau^+_{TH}.$$

Thus we obtain the bound,

$$\begin{aligned} t_{last,1} - t_{first,2} &\leqslant B + (\tau^+_{PC} - \tau^-_{PC}) + (\tau^+_{TH} - \tau^-_{TH}) - T^-_{first} \\ &\leqslant T_{QS}. \end{aligned}$$

Step $(k+1 \geqslant 3)$**:** Let $t_{first,k}$ be the time when the first correct process sends tick k. As our induction hypothesis, we assume that all correct nodes send tick $k-1$ by $t_{last,k-1} \leqslant t_{first,k} + T_{QS}$.

Let $t_{first,k+1}$ be the time the first correct node, say p, sends tick $k+1 \geqslant 3$. By Lemma 5.4, there exists a set Q of size $|Q| \geqslant 2f+1$, such that at time $t' = t_{first,k+1} - \tau^-_{TH} - \tau^-_{PC} - \tau^-_{Diff}$,

$$\forall q \in Q : \#\mathrm{req}_{p,q}(t') \geqslant k. \tag{5.7}$$

Clearly, there is a subset $\widetilde{Q} \subseteq Q$ of correct nodes among Q of size at least $|\widetilde{Q}| \geqslant f+1$. Let $\widetilde{Q}' := C \setminus (\widetilde{Q} \cup \{p\})$. Now consider the partitioning of correct nodes $C = \widetilde{Q} \cup \{p\} \cup \widetilde{Q}'$. We will prove the lemma separately for each of the three partitions: In case $q \in \widetilde{Q}$, by (5.7) and the remote channel properties, q has sent tick k by $t' - \tau^-_{rem} < t_{first,k+1}$. In case $q = p$, it has sent tick k by $t_{first,k+1}$. For the only non-trivial case $q \in \widetilde{Q}'$, it follows from the remote channel properties that any $\widetilde{q} \in \widetilde{Q}$, as a correct node, must have sent tick k by

$$t_{\widetilde{Q},k} := t_{first,k+1} - \tau^-_{TH} - \tau^-_{PC} - \tau^-_{Diff} - \tau^-_{rem}. \tag{5.8}$$

Now consider an arbitrary correct node $r \in \widetilde{Q}'$. By the induction hypothesis and (U),

$$\begin{aligned} t_{r,k-1} &\leqslant t_{first,k} + T_{QS} \\ &\leqslant t_{first,k+1} - T^-_{first} + T_{QS}. \end{aligned} \tag{5.9}$$

We may now apply Lemma 5.3 to node r and the set of correct nodes $\widetilde{Q}$ with $t_{r,k-1}$ and $t_{\widetilde{Q},k}$ from (5.9) and (5.8). This yields: r sends tick k at time $t_{r,k}$ with

$$\begin{aligned}
t_{r,k} &\leqslant \max\left\{ \begin{array}{l} t_{r,k-1} + \tau^+_{loc} + \tau^+_{Diff}, \\ t_{\widetilde{Q},k} + \tau^+_{rem} \end{array} \right\} + \tau^+_{PC} + \tau^+_{TH} \\
&\leqslant \max\left\{ \begin{array}{l} t_{first,k+1} - T^-_{first} + T_{QS} + \tau^+_{loc} + \tau^+_{Diff}, \\ t_{first,k+1} - \tau^-_{TH} - \tau^-_{PC} - \tau^-_{Diff} - \tau^-_{rem} + \tau^+_{rem} \end{array} \right\} + \tau^+_{PC} + \tau^+_{TH} \\
&= t_{first,k+1} + \max\left\{ \begin{array}{l} -T^-_{first} + T_{QS} + \tau^+_{loc} + \tau^+_{Diff}, \\ -\tau^-_{TH} - \tau^-_{PC} - \tau^-_{Diff} - \tau^-_{rem} + \tau^+_{rem} \end{array} \right\} + \tau^+_{PC} + \tau^+_{TH} \\
&\leqslant t_{first,k+1} + \max\left\{ \begin{array}{l} T_{QS}, \\ -\tau^-_{TH} - \tau^-_{PC} - \tau^-_{Diff} - \tau^-_{rem} + \tau^+_{rem} + \tau^+_{PC} + \tau^+_{TH} \end{array} \right\} \qquad (5.10) \\
&\leqslant t_{first,k+1} + T_{QS}, \qquad (5.11)
\end{aligned}$$

where (5.10) follows by applying Constraint 3 and (5.11) from the definition of T_{QS}. ■

Booting Simultaneity (BS)

Proof. The proof is by induction on $k \geqslant 1$.

Begin ($k = 1$): From Lemma 5.5, we deduce, that

$$\begin{aligned}
t_{last,1} - t_{first,1} &\leqslant B + (\tau^+_{PC} - \tau^-_{PC}) + (\tau^+_{TH} - \tau^-_{TH}) \\
&\leqslant T_{BS}(1).
\end{aligned}$$

Step ($k > 1$): Assume the Lemma is true for k. From (U) and (P), it follows that

$$\begin{aligned}
t_{first,k+1} - t_{first,k} &\geqslant T^-_{first} \\
t_{last,k+1} - t_{last,k} &\leqslant T_P.
\end{aligned}$$

In combination with the induction hypothesis, this yields:

$$\begin{aligned}
t_{last,k+1} - t_{first,k+1} &= (t_{last,k+1} - t_{last,k}) + (t_{last,k} - t_{first,k}) + (t_{first,k} - t_{first,k+1}) \\
&\leqslant T_P + T_{BS}(k) - T^-_{first} \\
&= T_{BS}(k+1).
\end{aligned}$$

This completes the induction step and the lemma follows. ■

Lemma 5.6. (Fastest Progress). *Assume that p is the first correct node that sends tick number $k \geqslant 1$ at time $t_{first,k}$. Then no correct node can send tick $k' \geqslant k$ before time $t_{first,k} + (k' - k)T^-_{first}$.*

Proof. The proof is by induction on $k' - k$.

Begin ($k' = k$): The lemma trivially holds since the first correct node cannot send tick k before time $t_{first,k}$.

Step ($k' \geqslant k+1$): Assume that p is the first correct node that sends tick k. The first correct node $q \in C$, by the induction hypothesis, does not send tick k' before $t + (k' - k)T^-_{first}$. Because of (U), no other correct node can send tick $k'+1$ by time $t+(k'-k)T^-_{first}+T^-_{first} = t + (k' + 1 - k)T^-_{first}$. ■

Having completed the proof of our major Theorem 5.1, we proceed with Lemmas 5.7, 5.8 and 5.9 that bound the progress of the ticks generated by the fastest node. For this purpose, we define $b^{max}(t)$ as the maximum of $\#b_p(t)$ over all correct nodes C, i.e., $b^{max}(t) := \max\{\#b_p(t) \mid p \in C\}$. Similarly, we define $b^{min}(t) := \min\{\#b_p(t) \mid p \in C\}$. Furthermore, we abbreviate the left limit of a function $f(\xi)$ at point t as $\lim_{\xi \to t^-} f(\xi) = f(t^{\rightarrow})$. For example, if node p sends tick $k \geqslant 1$ at time $t_{p,k}$, then $\#b_p(t^{\rightarrow}_{p,k}) = k - 1$ (whereas it holds that $\#b_p(t_{p,k}) = k$).

Lemma 5.7. (Maximum Increase of $b^{max}(\xi)$ in $(t_{first,k}, t)$). *If the first correct node sends tick $k \geqslant 1$ at $t_{first,k}$, then*

$$\forall t > t_{first,k} : b^{max}(t^{\rightarrow}) - b^{max}(t_{first,k}) \leqslant \left\lceil \frac{t - t_{first,k}}{T^-_{first}} \right\rceil - 1$$

or, equivalently: The number N of ticks sent by the correct first node in the interval $\mathcal{I} = (t_{first,k}, t)$ is upper bounded by $\left\lceil \frac{t-t_{first,k}}{T^-_{first}} \right\rceil - 1$.

Proof. Let $t_{first,j}$ be the time when the first correct node sends tick j, and assume by contradiction that

$$N \geqslant \left\lceil \frac{t - t_{first,k}}{T^-_{first}} \right\rceil . \tag{5.12}$$

According to the definition of N,

$$t_{first,k+N} < t. \tag{5.13}$$

By applying Lemma 5.6 to $t_{first,k}$ and $t_{first,k+N}$, we find

$$\begin{aligned} t_{first,k+N} - t_{first,k} &\geqslant NT^-_{first} \\ &\geqslant \left\lceil \frac{t - t_{first,k}}{T^-_{first}} \right\rceil T^-_{first} \\ &\geqslant T^-_{first} \frac{t - t_{first,k}}{T^-_{first}} \\ &= t - t_{first,k}. \end{aligned} \tag{5.14}$$

Clearly, (5.14) contradicts (5.13). ■

Lemma 5.8. (Maximum Increase of $b^{max}(\xi)$ in $(t_{first,k}, t]$). *If the first correct node sends tick $k \geqslant 1$ at $t_{first,k}$, then*

$$\forall t > t_{first,k} : b^{max}(t) - b^{max}(t_{first,k}) \leqslant \left\lfloor \frac{t - t_{first,k}}{T^-_{first}} \right\rfloor$$

or, equivalently: The number N of ticks sent by the correct first node in the interval $\mathcal{I} = (t_{first,k}, t]$ is upper bounded by $\left\lfloor \frac{t-t_{first,k}}{T^-_{first}} \right\rfloor$.

Proof. Let $t_{first,j}$ be the time when the first correct node sends tick j, and assume by contradiction that

$$N \geqslant \left\lfloor \frac{t - t_{first,k}}{T^-_{first}} \right\rfloor + 1. \tag{5.15}$$

According to the definition of N,

$$t_{first,k+N} \leqslant t. \tag{5.16}$$

By applying Lemma 5.6 to $t_{first,k}$ and $t_{first,k+N}$, and recalling $x < \lfloor x \rfloor + 1$ for all real x, we find

$$\begin{aligned} t_{first,k+N} - t_{first,k} &\geqslant N T^-_{first} \\ &\geqslant \left(\left\lfloor \frac{t - t_{first,k}}{T^-_{first}} \right\rfloor + 1 \right) T^-_{first} \\ &> T^-_{first} \frac{t - t_{first,k}}{T^-_{first}} \\ &= t - t_{first,k}. \end{aligned} \tag{5.17}$$

Clearly, (5.17) contradicts (5.16). ∎

The following Lemma 5.9 is a weaker form of Lemma 5.8, where the beginning of the interval $\mathcal{I}$ not necessarily coincides with the sending of a tick by the first correct node, i.e., where $\mathcal{I}$ is not "aligned" with $t_{first,k}$.

Lemma 5.9. (Maximum Increase of $b^{max}(\xi)$ in $(t, t']$).

$$\forall t' > t : b^{max}(t') - b^{max}(t) \leqslant \left\lceil \frac{t' - t}{T^-_{first}} \right\rceil$$

or, equivalently: The number N of ticks sent by the correct first node in the interval $\mathcal{I} = (t, t']$ is upper bounded by $\left\lceil \frac{t'-t}{T^-_{first}} \right\rceil$.

Proof. Let $t_{first,j}$ the time when the first correct node sends tick j. We distinguish two cases: (i) $N \geqslant 1$, i.e., $\exists k : t_{first,k+1} \in \mathcal{I}$ and (ii) $N = 0$.

ad (i): Assume by contradiction that

$$N \geqslant \left\lceil \frac{t' - t}{T^-_{first}} \right\rceil + 1. \tag{5.18}$$

According to the definition of N and the assumption $N \geqslant 1$, there must be some k s.t.

$$t_{first,k+1} > t \tag{5.19}$$
$$t_{first,k+N} \leqslant t'. \tag{5.20}$$

By applying Lemma 5.6 to $t_{first,k+1}$ and $t_{first,k+N}$, we find

$$\begin{aligned} t_{first,k+N} - t &> t_{first,k+N} - t_{first,k+1} \\ &\geqslant (N-1)T^-_{first} \\ &\geqslant \left\lceil \frac{t' - t}{T^-_{first}} \right\rceil T^-_{first} \\ &\geqslant T^-_{first} \frac{t' - t}{T^-_{first}} \\ &= t' - t. \end{aligned} \tag{5.21}$$

Clearly, (5.21) contradicts (5.20).

ad (ii): Obviously $N = 0 \leqslant \left\lceil \frac{t'-t}{T^-_{first}} \right\rceil$ holds for $t' > t$. ■

The following Lemma 5.10 is an analogon to Lemma 5.9 for any correct node p.

Lemma 5.10. (Maximum Increase of $\#b_p(\xi)$ in $(t, t']$).

$$\forall t' > t : \#b_p(t') - \#b_p(t) \leqslant \left\lceil \frac{t' - t}{T_{min}} \right\rceil$$

or, equivalently: The number N of ticks sent by the correct node p in the interval $\mathcal{I} = (t, t']$ is upper bounded by $\left\lceil \frac{t'-t}{T_{min}} \right\rceil$.

Proof. Let $t_{p,j}$ the time when node p sends tick j. We distinguish two cases: (i) $N \geqslant 1$, i.e., $\exists k : t_{p,k+1} \in \mathcal{I}$ and (ii) $N = 0$.

ad (i): Assume by contradiction that

$$N \geqslant \left\lceil \frac{t' - t}{T_{min}} \right\rceil + 1. \tag{5.22}$$

According to the definition of N and the assumption $N \geqslant 1$, there must be some k s.t.

$$t_{p,k+1} > t \tag{5.23}$$
$$t_{p,k+N} \leqslant t'. \tag{5.24}$$

By applying Lemma 5.1 to $t_{p,k+1}$ and $t_{p,k+N}$, we find

$$\begin{aligned} t_{p,k+N} - t &> t_{p,k+N} - t_{p,k+1} \\ &\geqslant (N-1)T_{min} \\ &\geqslant \left\lceil \frac{t'-t}{T_{min}} \right\rceil T_{min} \\ &\geqslant T_{min} \frac{t'-t}{T_{min}} \\ &= t' - t. \end{aligned} \tag{5.25}$$

Clearly, (5.25) contradicts (5.23).

ad (ii): Obviously $N = 0 \leqslant \left\lceil \frac{t'-t}{T_{min}} \right\rceil$ holds for $t' > t$. ■

The next Lemma 5.11 bounds the progress of the last correct node.

Lemma 5.11. (Last Progress) *If the last correct node sends tick $k \geqslant 1$ at $t_{last,k}$, then the last correct node sends tick $k+N$, $N \geqslant 0$, by $t_{last,k+N}$, with*

$$t_{last,k+N} - t_{last,k} \leqslant NT_P.$$

Proof. The proof is by induction on N:

Begin ($N = 0$): Clearly $t_{last,k} - t_{last,k} \leqslant 0$ is true.

Step ($N > 0$): As induction hypothesis, assume that the Lemma is true for $N-1$. By applying (P), we immediately obtain

$$\begin{aligned} t_{last,k+N} - t_{last,k} &= (t_{last,k+N} - t_{last,k+N-1}) + (t_{last,k+N-1} - t_{last,k}) \\ &\leqslant T_P + (N-1)T_P \\ &= NT_P. \end{aligned}$$

■

The following two simple technical lemmas complete the intermediate proof layer.

Lemma 5.12 (C3). (Progress by (QS)). *If p is a correct node which sends tick $k \geqslant 2$ at $t_{p,k}$, then p sends tick $k+N$, $N \geqslant 1$, at $t_{p,k+N}$ with*

$$t_{p,k+N} - t_{p,k} \leqslant (N+1)T_P + T_{QS}.$$

Proof. With Lemma 5.11 and (QS), it follows that

$$\begin{aligned} t_{p,k+N} - t_{p,k} &\leqslant t_{last,k+N} - t_{first,k} \\ &= (t_{last,k+N} - t_{last,k-1}) + (t_{last,k-1} - t_{first,k}) \\ &\leqslant (N+1)T_P + T_{QS}. \end{aligned}$$

■

Lemma 5.13. (Progress by (BS)) *If p is a correct node which sends tick $k \geqslant 1$ at $t_{p,k}$, then p sends tick $k+N$, $N \geqslant 1$, at $t_{p,k+N}$ with*

$$t_{p,k+N} - t_{p,k} \leqslant NT_P + T_{BS}(k).$$

Proof. With Lemma 5.11 and (BS), it follows that

$$\begin{aligned} t_{p,k+N} - t_{p,k} &\leqslant t_{last,k+N} - t_{first,k} \\ &= (t_{last,k+N} - t_{last,k}) + (t_{last,k} - t_{first,k}) \\ &\leqslant NT_P + T_{BS}(k). \end{aligned}$$

■

5.2.3 Top Proof Layer

We are now ready for establishing our major results. The first one, Theorem 5.2, bounds the precision π of our algorithm, i.e., shows that for every pair of correct nodes $p, q \in C$: $\forall t : |\#b_q(t) - \#b_p(t)| \leqslant \pi$.

Theorem 5.2 (C3). (Precision). $\pi := \left\lceil \frac{T_{QS}}{T^-_{first}} \right\rceil + 1$ *is a valid precision-bound.*

Proof. Let p, q be two distinct correct nodes. Clearly for all t, $|\#b_q(t) - \#b_p(t)| \leqslant b^{max}(t) - b^{min}(t)$. We will bound this term by distinguishing three cases for t: (i) $t \in [0, t_{last,1})$, (ii) $t = t_{last,k}$ for some $k \geqslant 1$ and (iii) $t \in (t_{last,k}, t_{last,k+1})$ for some $k \geqslant 1$.

ad (i): Since $t \in [0, t_{last,1})$, we have $b^{max}(t) \leqslant b^{max}(t^{\rightarrow}_{last,1})$ and $b^{min}(t) = 0$. Thus,

$$\begin{aligned}
b^{max}(t) - b^{min}(t) &= b^{max}(t) \\
&\leqslant b^{max}(t^{\rightarrow}_{last,1}) \\
&= \left(b^{max}(t^{\rightarrow}_{last,1}) - b^{max}(t_{first,1})\right) + b^{max}(t_{first,1}) \\
&\leqslant \left\lceil \frac{t_{last,1} - t_{first,1}}{T^{-}_{first}} \right\rceil && (5.26) \\
&\leqslant \left\lceil \frac{B + \tau^{+}_{PC} + \tau^{+}_{TH} - (\tau^{-}_{PC} + \tau^{-}_{TH})}{T^{-}_{first}} \right\rceil && (5.27) \\
&\leqslant \left\lceil \frac{T_{QS} + T^{-}_{first}}{T^{-}_{first}} \right\rceil \\
&= \left\lceil \frac{T_{QS}}{T^{-}_{first}} \right\rceil + 1, && (5.28)
\end{aligned}$$

where (5.26) follows from Lemma 5.7 and (5.27) from Lemma 5.5.

ad (ii):

$$\begin{aligned}
b^{max}(t_{last,k}) - b^{min}(t_{last,k}) &= (b^{max}(t_{last,k}) - b^{max}(t_{first,k+1})) + \left(b^{max}(t_{first,k+1}) - b^{min}(t_{last,k})\right) \\
&= (b^{max}(t_{last,k}) - b^{max}(t_{first,k+1})) + 1 \\
&\leqslant \left\lfloor \frac{t_{last,k} - t_{first,k+1}}{T^{-}_{first}} \right\rfloor + 1 && (5.29) \\
&\leqslant \left\lfloor \frac{T_{QS}}{T^{-}_{first}} \right\rfloor + 1 && (5.30)
\end{aligned}$$

where (5.29) follows from Lemma 5.8 and (5.30) from (QS).

ad (iii): Since $t \in (t_{last,k}, t_{last,k+1})$, we have $b^{max}(t) \leqslant b^{max}(t^{\rightarrow}_{last,k+1})$. Thus,

$$\begin{aligned}
b^{max}(t) - b^{min}(t) &= (b^{max}(t) - b^{max}(t_{first,k+2})) + \left(b^{max}(t_{first,k+2}) - b^{min}(t)\right) \\
&= (b^{max}(t) - b^{max}(t_{first,k+2})) + (k + 2 - k) \\
&\leqslant \left(b^{max}(t^{\rightarrow}_{last,k+1}) - b^{max}(t_{first,k+2})\right) + 2 \\
&\leqslant \left\lceil \frac{t_{last,k+1} - t_{first,k+2}}{T^{-}_{first}} \right\rceil + 1 && (5.31) \\
&\leqslant \left\lceil \frac{T_{QS}}{T^{-}_{first}} \right\rceil + 1, && (5.32)
\end{aligned}$$

where (5.31) follows from Lemma 5.7 and (5.32) from (QS).

Combining (5.28), (5.30) and (5.32) provides a precision bound π for arbitrary t:

$$\begin{aligned}\pi &:= \max\left\{\left\lfloor\frac{T_{QS}}{T^-_{first}}\right\rfloor, \left\lceil\frac{T_{QS}}{T^-_{first}}\right\rceil\right\} + 1\\ &= \left\lceil\frac{T_{QS}}{T^-_{first}}\right\rceil + 1\end{aligned}$$

The statement holds in all three cases. ■

Our next major result, Theorem 5.3 (Accuracy), bounds the number of ticks generated locally at a correct node p during some real-time interval $(t_1, t_2]$, i.e., allows to make statements about the local clock frequency. For example, it reveals that the long-term frequency is within $\left[1/T_P, 1/T^-_{first}\right]$.

Theorem 5.3 (C3). (Accuracy). *Given t_1 and t_2 with $t_2 > t_1 \geqslant t_{p,1}$, the accuracy $\#b_p(t_2) - \#b_p(t_1)$ of any correct node p is bounded by*

$$\begin{aligned}&\max\left\{0, \left\lfloor\frac{t_2 - t_1 - \max\left\{T_{BS}(1), \min\left\{T_{BS}(k), T_{QS} + T_P\right\} \mid k \geqslant 2\right\}}{T_P}\right\rfloor + 1\right\}\\ &\quad\leqslant \#b_p(t_2) - \#b_p(t_1) \leqslant\\ &\min\left\{\left\lceil\frac{t_2 - t_1}{T^-_{first}}\right\rceil + \pi, \left\lceil\frac{t_2 - t_1}{T_{min}}\right\rceil\right\}.\end{aligned}$$

Proof. To prove the accuracy upper bound, we start from

$$\begin{aligned}\#b_p(t_2) - \#b_p(t_1) &\leqslant b^{max}(t_2) - b^{min}(t_1)\\ &\leqslant \left(b^{max}(t_2) - b^{max}(t_1)\right) + \left(b^{max}(t_1) - b^{min}(t_1)\right).\end{aligned} \tag{5.33}$$

Both terms of (5.33) can be bounded by applying Lemma 5.9 and Theorem 5.2, respectively, which gives

$$b^{max}(t_2) - b^{max}(t_1) \leqslant \left\lceil\frac{t_2 - t_1}{T^-_{first}}\right\rceil \tag{5.34}$$

$$b^{max}(t_1) - b^{min}(t_1) \leqslant \pi, \tag{5.35}$$

thus yielding

$$\#b_p(t_2) - \#b_p(t_1) \leqslant \left\lceil\frac{t_2 - t_1}{T^-_{first}}\right\rceil + \pi. \tag{5.36}$$

Moreover, from Lemma 5.10, it follows that

$$\#b_p(t_2) - \#b_p(t_1) \leqslant \left\lceil \frac{t_2 - t_1}{T_{min}} \right\rceil . \tag{5.37}$$

A combination of both bounds (5.36) and (5.37) leads to

$$\#b_p(t_2) - \#b_p(t_1) \leqslant \min \left\{ \left\lceil \frac{t_2 - t_1}{T^-_{first}} \right\rceil + \pi, \left\lceil \frac{t_2 - t_1}{T_{min}} \right\rceil \right\}$$

To prove the accuracy lower bound, let $k = \#b_p(t_1) \geqslant 1$ and $N \geqslant 0$, s.t., $k + N = \#b_p(t_2)$. Clearly such k and N exist since p has sent tick 1 by t_1. By the definition of k and N,

$$t_{p,k} \leqslant t_1 \tag{5.38}$$

$$t_{p,k+N+1} > t_2. \tag{5.39}$$

For $k \geqslant 2$ we can apply Lemma 5.12 together with (5.38) and (5.39), yielding

$$\begin{aligned} t_2 - t_1 &< t_{p,k+N+1} - t_{p,k} && (5.40)\\ &\leqslant (N+2)T_P + T_{QS} \Rightarrow \\ N &> \frac{t_2 - t_1 - T_{QS} - 2T_P}{T_P} \Rightarrow \\ N &\geqslant \left\lfloor \frac{t_2 - t_1 - T_{QS} - T_P}{T_P} \right\rfloor + 1. && (5.41) \end{aligned}$$

For $k \geqslant 1$, we apply Lemma 5.13 to (5.40), which yields

$$\begin{aligned} t_2 - t_1 &< (N+1)T_P + T_{BS}(k) \Rightarrow \\ N &> \frac{t_2 - t_1 - T_{BS}(k) - T_P}{T_P} \Rightarrow \\ N &\geqslant \left\lfloor \frac{t_2 - t_1 - T_{BS}(k)}{T_P} \right\rfloor + 1. && (5.42) \end{aligned}$$

Combining the bounds (5.41), (5.42) and the trivial bound 0 yields (with $\#b_p(t_1) = k$)

$$\#b_p(t_2) - \#b_p(t_1) = N \geqslant \begin{cases} \max\left\{0, \left\lfloor \frac{t_2 - t_1 - T_{BS}(1)}{T_P} \right\rfloor + 1\right\} & \text{if } k = 1, \\ \max\left\{0, \left\lfloor \frac{t_2 - t_1 - \min\left\{T_{BS}(k), T_{QS} + T_P\right\}}{T_P} \right\rfloor + 1\right\} & \text{if } k \geqslant 2. \end{cases} \tag{5.43}$$

In case k is not known, a valid bound is the minimum of all lower bounds, i.e.,

$$\#b_p(t_2) - \#b_p(t_1) \geqslant \max\left\{0, \left\lfloor \frac{t_2 - t_1 - \max\left\{T_{BS}(1), \min\left\{T_{BS}(k), T_{QS} + T_P\right\} \mid k \geqslant 2\right\}}{T_P} \right\rfloor + 1\right\}. \tag{5.44}$$

■

Note that the term $\min\{T_{BS}(k), T_{QS} + T_P\}$ for $k \geqslant 2$ in both (5.43) and (5.44) accounts for the fact that correct nodes may be synchronized very tightly after booting (within $T_{BS}(k)$), such that $T_{QS} + T_P$ would be too conservative. From the definition of $T_{BS}(k)$ in (5.6), however, it follows that the initial synchrony from booting cannot usually be maintained: When $T^-_{first} < T_P$, which is typically the case in real systems, we obtain $\lim_{k\to\infty} T_{BS}(k) = \infty$. Thus, for some k_0, $\forall k \geqslant k_0 : T_{QS} + T_P < T_{BS}(k)$, i.e., the constant bound from (QS) will be tighter, which prevents the nodes from drifting apart further.

This ends the analysis of the high-level solution. Now we will prove the low-level solution correct.

5.2.3.1 Low-level solution

We will first show that *Precision* and *Accuracy*, hold, too:

Lemma 5.14 (C1). *Assumption 1 holds for each correct node p in C.*

Proof. Consider an arbitrary p in C. Let Q be the set of correct predecessor modules of p. Since all q in Q are $GJ_{\mathrm{Imp}}(J)$ modules, we can apply Corollary 4.2 to each of them. It follows that Assumption 1 holds for p. ■

We are now in the position to state and prove correct a result that allows us to take over the bounds for Precision and Accuracy derived in Theorems 5.2 and 5.3:

Lemma 5.15 (C1,C2). *Each correct node p in C with input channels implements a General Join (with respective thresholds $\langle (2f+1)^{th}_0, (f+1)^{th}_1 \rangle$) with input channels, where the delay bounds $\tau^{\pm}_{Diff}$ of the General Join are replaced by the bounds $\tau^{\pm}_{Diff} + [0, \varepsilon]$, for an arbitrarily small $\varepsilon > 0$.*

Proof. Consider a correct node p in C. By Lemma 5.14, Assumption 1 holds for p. We may thus apply Theorem 4.1. The lemma follows. ■

From Lemma 5.15, it follows that Theorems 5.2 and 5.3 hold for the low-level solution as well, provided that (i) each occurrence of τ^+_{Diff} has to be replaced by $\tau^+_{Diff} + \varepsilon$ for an arbitrarily small $\varepsilon > 0$, and (ii) Constraints C1, C2 and C3 hold.

Our final Theorems 5.4 and 5.6 establish bounds on the size of the local and remote pipeline. We start with a technical Lemma 5.16, which bounds the maximum time some tick can exist in the system before it is eliminated by the Diff-Gate in all pipepairs associated with correct nodes.

Lemma 5.16 (C1, C2, C3). *If the first correct node has sent tick $k \geqslant 2$ by time t, then every correct node $p \in C$ has removed tick $k-2$ from all its pipepairs corresponding to correct nodes $q \in C$ by time $t + T_{del}$, with*

$$T_{del} := T_{QS} + \max\{\tau^+_{loc}, \tau^+_{rem}\} + \tau^+_{Diff}, \tag{5.45}$$

or, which is equivalent

$$\#d_{p,q}(t + T_{del}) \geqslant b^{max}(t) - 2.$$

Proof. Consider a pair of pipes (p, q) located at $p \in C$, corresponding to a different $q \in C$. Furthermore, assume that $b^{max}(t) \geqslant k$ holds at time t. We are interested in how much later tick $k-2$ is removed from this pipepair:

Clearly, $t_{first,k} \leqslant t$, and by (QS),

$$\begin{aligned} t_{p,k-1} &\leqslant t_{last,k-1} \\ &\leqslant t_{first,k} + T_{QS} \text{ and} \qquad (5.46) \\ t_{q,k-1} &\leqslant t_{first,k} + T_{QS}. \qquad (5.47) \end{aligned}$$

We can now apply Lemma 4.8 in combination with (5.46) and (5.47), which reveals that tick $k-2$ is removed from the pipepair at p corresponding to q by time $t_{rmv,k-2}$, with

$$\begin{aligned} t_{rmv,k-2} &\leqslant \max\{t_{p,k-1} + \tau^+_{loc}, t_{q,k-1} + \tau^+_{rem}\} + \tau^+_{Diff} \\ &\leqslant t_{first,k} + T_{QS} + \max\{\tau^+_{loc}, \tau^+_{rem}\} + \tau^+_{Diff} \\ &= t_{first,k} + T_{del} \\ &\leqslant t + T_{del}. \end{aligned}$$

■

Theorem 5.4 (C1, C2, C3). (Local pipeline size bound). *For every pair of distinct correct nodes $p, q \in C$, $\#s^{loc}_{p,q}(t)$ is bounded by S_{loc}, with*

$$S_{loc} := \max\left\{ \left\lceil \frac{T_{del} - \tau^-_{loc}}{T^-_{first}} \right\rceil + 2, \left\lfloor \frac{T_{del} - \tau^-_{loc}}{T^-_{first}} \right\rfloor + 3 \right\}. \qquad (5.48)$$

Proof. Choose two arbitrary distinct correct node p, q and consider $\#s^{loc}_{p,q}(t)$. We distinguish between two cases for t: (i) $t \geqslant t_{first,2} + T_{del}$ and (ii) $t < t_{first,2} + T_{del}$.

ad (i):

$$\begin{aligned} \#s^{loc}_{p,q}(t) &= \#r^{loc}_{p,q}(t) - \#d_{p,q}(t) \\ &\leqslant \#b_p(t - \tau^-_{loc}) - \#d_{p,q}(t) \\ &\leqslant b^{max}(t - \tau^-_{loc}) - \#d_{p,q}(t) \\ &= \left(b^{max}(t - \tau^-_{loc}) - b^{max}(t - T_{del})\right) + \left(b^{max}(t - T_{del}) - \#d_{p,q}(t)\right) \\ &\leqslant \left\lceil \frac{T_{del} - \tau^-_{loc}}{T^-_{first}} \right\rceil + 2 \qquad (5.49) \\ &\leqslant S_{loc}, \end{aligned}$$

where (5.49) follows from applying Lemma 5.9 and Lemma 5.16.

ad (ii): Since $\#s^{rem}_{p,q}(t) \leqslant \#r^{rem}_{p,q}(t) + 1$ must always hold, because the local pipe may contain at most all ticks received so far plus the initial tick 0, we obtain

$$\begin{aligned}
\#s^{loc}_{p,q}(t) &\leqslant \#r^{loc}_{p,q}(t) + 1 \\
&\leqslant \#b_p(t - \tau^-_{loc}) + 1 \\
&\leqslant b^{max}(t - \tau^-_{loc}) + 1 \\
&\leqslant b^{max}(t_{first,2} + T_{del} - \tau^-_{loc}) + 1 \\
&\leqslant \left(b^{max}(t_{first,2} + T_{del} - \tau^-_{loc}) - b^{max}(t_{first,2})\right) + b^{max}(t_{first,2}) + 1 \qquad (5.50) \\
&\leqslant \left\lfloor \frac{T_{del} - \tau^-_{loc}}{T^-_{first}} \right\rfloor + 3 \qquad (5.51) \\
&\leqslant S_{loc},
\end{aligned}$$

where (5.51) follows from Lemma 5.8. ■

In order to derive a bound for the size of the remote pipelines, we can use exactly the same derivation based on Lemma 5.16 as used in Theorem 5.4 (with remote delays instead of local delays) to obtain the following Theorem 5.5.

Theorem 5.5 (C1, C2, C3). (Remote pipeline size bound). *For every pair of distinct correct nodes $p, q \in C$, $\#s^{rem}_{p,q}(t)$ is bounded by S_{rem}, with*

$$S_{rem} := \max\left\{ \left\lceil \frac{T_{del} - \tau^-_{rem}}{T^-_{first}} \right\rceil + 2, \left\lfloor \frac{T_{del} - \tau^-_{rem}}{T^-_{first}} \right\rfloor + 3 \right\}. \qquad (5.52)$$

However, the resulting bound is overly large in most parameter settings: In order to maximize $\#s^{rem}_{p,q}(t)$, we need a scenario where the local node p is slow and the remote node q is fast. The time T_{del} established by Lemma 5.16 is too conservative for this case, however, since it actually considers q being slow. The following Lemma 5.17 provides a refined result.

Lemma 5.17 (C1, C2, C3). *If $k := b^{max}(t) \geqslant 2$, then for every pair of correct nodes $p, q \in C$, with*

$$\begin{aligned}
T^{loc}_{del} &:= T_{QS} + \tau^+_{loc} + \tau^+_{Diff}, \qquad (5.53) \\
t' &:= t + T^{loc}_{del} - \tau^+_{rem} - \tau^+_{Diff}, \\
k' &:= \#b_q(t'),
\end{aligned}$$

the following holds:

(a) If $k' \geqslant k - 1$, then tick $k - 2$ is removed from pipepair (p, q) by time $t + T^{loc}_{del}$, i.e.,

$$\#d_{p,q}(t + T^{loc}_{del}) \geqslant b^{max}(t) - 2.$$

(b) If $k' \leqslant k-2$, then tick $k'-1$ is removed from pipepair (p,q) by time $t + T^{loc}_{del}$, i.e.,

$$\#d_{p,q}(t + T^{loc}_{del}) \geqslant \#b_q(t') - 1.$$

Proof. To prove case (a), assume $k' \geqslant k-1$, which implies $t_{q,k-1} \leqslant t'$. Hence,

$$t_{q,k-1} + \tau^+_{rem} + \tau^+_{Diff} \leqslant t' + \tau^+_{rem} + \tau^+_{Diff} = t + T^{loc}_{del},$$

and, by (QS) and our assumption $t_{first,k} \leqslant t$,

$$t_{p,k-1} + \tau^+_{loc} + \tau^+_{Diff} \leqslant t_{first,k} + T_{QS} + \tau^+_{loc} + \tau^+_{Diff} \leqslant t + T^{loc}_{del}. \tag{5.54}$$

We can now apply Lemma 4.8, which reveals that tick $k-2$ is removed from the pipepair (p,q) by time $t_{rmv,k-2}$, with

$$\begin{aligned} t_{rmv,k-2} &\leqslant \max\{t_{p,k-1} + \tau^+_{loc}, t_{q,k-1} + \tau^+_{rem}\} + \tau^+_{Diff} \\ &\leqslant t + T^{loc}_{del} \end{aligned} \tag{5.55}$$

as asserted.

To prove case (b), assume $k' \leqslant k-2$, which implies $t_{q,k-1} > t'$ and $t_{q,k'} \leqslant t'$. Hence,

$$\begin{aligned} t_{q,k'} + \tau^+_{rem} + \tau^+_{Diff} &\leqslant t' + \tau^+_{rem} + \tau^+_{Diff} = t + T^{loc}_{del}, \\ t_{q,k-1} + \tau^+_{rem} + \tau^+_{Diff} &> t' + \tau^+_{rem} + \tau^+_{Diff} = t + T^{loc}_{del}. \end{aligned}$$

Since (5.54) also holds in case (b) and trivially $t_{p,k'} \leqslant t_{p,k-1}$, we find

$$t_{p,k'} + \tau^+_{loc} + \tau^+_{Diff} \leqslant t + T^{loc}_{del}.$$

We can again apply Lemma 4.8, which reveals that tick $k'-1$ is removed from the pipepair (p,q) by time $t_{rmv,k'-1}$, with

$$\begin{aligned} t_{rmv,k'-1} &\leqslant \max\{t_{p,k'} + \tau^+_{loc}, t_{q,k'} + \tau^+_{rem}\} + \tau^+_{Diff} \\ &\leqslant t + T^{loc}_{del}. \end{aligned}$$

as asserted. ∎

Now we can establish our final major Theorem 5.6.

Theorem 5.6 (C1, C2, C3). (Remote pipeline size bound). *For every pair of distinct correct nodes $p, q \in C$, $\#s^{rem}_{p,q}(t)$ is bounded by S_{rem}, with*

$$S_{rem} := \max\left\{ \left\lceil \frac{T^{loc}_{del} - \tau^-_{rem}}{T^-_{first}} \right\rceil + 2, \left\lfloor \frac{T^{loc}_{del} - \tau^-_{rem}}{T^-_{first}} \right\rfloor + 3, \left\lceil \frac{\tau^+_{rem} - \tau^-_{rem} + \tau^+_{Diff}}{T_{min}} \right\rceil + 1 \right\}. \tag{5.56}$$

Proof. Choose two arbitrary distinct correct nodes p, q and consider $\#s^{rem}_{p,q}(t)$. Since we will apply Lemma 5.17, we distinguish the following cases:

Case (a): Suppose $t \geqslant t_{first,2} + T^{loc}_{del}$ and $\#b_q(t - \tau^+_{rem} - \tau^+_{Diff}) \geqslant b^{max}(t - T^{loc}_{del}) - 1$. Then,

$$\begin{aligned}
\#s^{rem}_{p,q}(t) &= \#r^{rem}_{p,q}(t) - \#d_{p,q}(t) \\
&\leqslant \#b_q(t - \tau^-_{rem}) - \#d_{p,q}(t) \\
&\leqslant b^{max}(t - \tau^-_{rem}) - \#d_{p,q}(t) \\
&= \left(b^{max}(t - \tau^-_{rem}) - b^{max}(t - T^{loc}_{del})\right) + \left(b^{max}(t - T^{loc}_{del}) - \#d_{p,q}(t)\right) \\
&\leqslant \left\lceil \frac{T^{loc}_{del} - \tau^-_{rem}}{T^-_{first}} \right\rceil + 2 \qquad (5.57) \\
&\leqslant S_{rem},
\end{aligned}$$

where (5.57) follows from applying Lemma 5.9 and Lemma 5.17.

Case (b): Suppose $t \geqslant t_{first,2} + T^{loc}_{del}$ and $\#b_q(t - \tau^+_{rem} - \tau^+_{Diff}) \leqslant b^{max}(t - T^{loc}_{del}) - 2$. Then, we find

$$\begin{aligned}
\#s^{rem}_{p,q}(t) &= \#r^{rem}_{p,q}(t) - \#d_{p,q}(t) \\
&\leqslant \#b_q(t - \tau^-_{rem}) - \#d_{p,q}(t) \\
&= \#b_q(t - \tau^-_{rem}) - \#b_q(t - \tau^+_{rem} - \tau^+_{Diff}) \\
&\quad + \#b_q(t - \tau^+_{rem} - \tau^+_{Diff}) - \#d_{p,q}(t) \\
&\leqslant \left\lceil \frac{\tau^+_{rem} - \tau^-_{rem} + \tau^+_{Diff}}{T_{min}} \right\rceil + 1 \qquad (5.58) \\
&\leqslant S_{rem},
\end{aligned}$$

where (5.58) follows from Lemma 5.10. Note that $\tau^+_{rem} - \tau^-_{rem} + \tau^+_{Diff}$ is always non-negative.

Case (c): Suppose $t < t_{first,2} + T^{loc}_{del}$. Since $\#s^{rem}_{p,q}(t) \leqslant \#r^{rem}_{p,q}(t) + 1$ must always hold, because the remote pipe may contain at most all ticks received so far plus the initial tick 0 we obtain

$$\begin{aligned}
\#s^{rem}_{p,q}(t) &\leqslant \#r^{rem}_{p,q}(t) + 1 \\
&\leqslant \#b_q(t - \tau^-_{rem}) + 1 \\
&\leqslant b^{max}(t - \tau^-_{rem}) + 1 \\
&\leqslant b^{max}(t_{first,2} + T^{loc}_{del} - \tau^-_{rem}) + 1 \\
&\leqslant \left(b^{max}(t_{first,2} + T^{loc}_{del} - \tau^-_{rem}) - b^{max}(t_{first,2})\right) + b^{max}(t_{first,2}) + 1 \qquad (5.59) \\
&\leqslant \left\lfloor \frac{T^{loc}_{del} - \tau^-_{rem}}{T^-_{first}} \right\rfloor + 3 \qquad (5.60) \\
&\leqslant S_{rem},
\end{aligned}$$

where (5.60) follows from Lemma 5.8. ∎

5.3 Remarks on the TG-Alg Solution

At the beginning of this chapter, it was argued that the DARTS distributed tick generation solution provides synchronously clocked functional units with local clock signals that are synchronized to each other in spite of up to f faulty TG-Algs. Different functional units can also exchange data over a communication medium distinct from the TG-Net either (i) by appropriately downscaling their local clock to non-overlapping macro-tick rounds or (ii) by communication via bounded-size ring-buffers. This is enabled by bounded precision which both allows constant factor clock division and read/write-conflict-free communication via the shared buffers. The latter approach has been described and analyzed in detail by Polzer *et al.* in [78].

There is, however, a different solution for the functional units to communicate with each other, which combines the data communication with the proposed TG-Alg solution. Remember that Chapter 4 was concerned with the modeling of asynchronous fault-tolerant circuits. There it was argued that one possible solution to perform the modeling is to partition the circuit under consideration into a control part and a data-dependent part. With this in mind, the proposed TG-Alg solution is not only a circuit that solves the tick generation problem, but can be viewed as a set of interconnected General Joins, forming the control part of a fault-tolerant asynchronous design. The idea now is to add the data-dependent circuit to the TG-Alg solution, i.e., to attach data word $k \geqslant 1$ to tick k. This can be done in a way analogous to a method that has been described by Sutherland in [92] for micropipeline structures:

- Each stage of a micropipeline provides a clock signal for an m-bit wide register, where m is the bit width of the data path. That is, beneath each of the local and remote pipes of the $+/-$ counters a data path is added.
- Each point-to-point connection of the TG-Net is extended by an m-bit wide bus over which the data of the sending TG-Alg is provided to the respective remote pipe of the receiving TG-Alg.
- The Tick Generation unit is extended by a Data Generation unit, which merges all the data attached to the remote k received the other TG-Algs, and generates the data attached to the newly broadcast tick k (respectively $k+1$, depending on whether the $f+1$ or $2f+1$ rule fired).

Short Note on Transient Failures. A straightforward simplification of the resulting circuit is to remove the data registers from all the local pipes of the $+/-$ counters, since the local pipes' sole purpose is to memorize how many ticks have already been locally broadcast (that is, to store the anti-tokens). Thus, attaching the locally generated data to all of these ticks seems to be unnecessary.

There is an important generalization, however, where one could profit from data attached to the ticks residing in the local pipes: In this work we only considered *static failure*

models, that is, (i) modules in a faulty FCR may behave arbitrarily during the whole execution and (ii) the number of faulty FCRs is upper bounded by f per execution. We therefore do not benefit form the difference of transient and permanent faults, but treat them equally. While this is appropriate for small mission-times, long-mission-time systems, where a large number of transient failures can violate any f-bound, require *dynamic failure models*, where all FCRs may be faulty during an execution, albeit not at the same time.

Unfortunately the proposed TG-Alg solution is non-straightforward to adapt to a dynamic fault model: in contrast to the high-level algorithm stated in Figure 5.3, the TG-Algs only send event (differential) information in form of anonymous tick transitions, without tick numbers attached, over the TG-Net. Thus the conversion back to the state information, i.e., the tick number, has to be performed by the receiving TG-Alg. While this approach is cheaper (we need only a single wire from each TG-Alg to each other TG-Alg), it is the root of the "anonymity" problem. When two ticks are lost, say from node p to node q, (either because of a temporarily faulty TG-Alg p or a temporarily faulty link from p to q), the corresponding TG-Algs p and q remain unsynchronized forever, i.e., the Diff-Gate of pipe (q, p) will match p's remote tick $k-2$ and q's local tick k. Consequently, transient faults are in fact converted into permanent faults. If we would, however, be able to attach data to both ticks, the Diff-Gate could note the difference in the tick numbers. A possible solution thus is to periodically use the data path for sending a tick identifier attached to a tick. Since the identifiers are fed back into the local pipes, the Diff-gate can detect transient faults when matching a tick with identifier in one pipe and a tick without identifier in the corresponding pipe. A detailed discussion of a system of TG-Algs within a dynamic failure model is outside the scope of this work and will be subject to future work.

5.4 Related Work

Further Reading: All work that has been published on the General Join Module has been done in the context of solving the tick generation problem. Thus one could cite all the work presented in Section 4.8, here, too. A short overview of the DARTS approach can be found in [89]. The reader interested in the non-trivial implementation testing of DARTS is pointed to the work of Steininger *et al.* [90] and Függer *et al.* [39].

Existing work: The publications on (fault-tolerant) clock synchronization are numerous. Since one of the advantages of DARTS is to avoid external clock sources, we do not consider the sizeable body of work on hardware-assisted fault-tolerant clock synchronization (see [81] for an overview) here. The few approaches for distributed clock generation without external clock sources we are aware of are essentially based on a (distributed) ring oscillator, which is formed by gates arranged in a feedback loop. Instead of being dictated by a quartz, the frequency of the generated clock signal is determined by the end-to-end delay of the feedback loop. In [67], a regular structure of closed loops of an odd number of inverters is used for distributed clock generation. Similarly, [27, 28] employs local tick generation cells, arranged in a two-dimensional grid, with each cell inverting its output signal when

its four inputs (from the up, down, left and right neighbor) match the current clock output value. Since clock synchronization theory [25] reveals that high connectivity is required for bounded synchronization tightness in the presence of failures, however, the sparsely connected designs proposed in [27, 28, 67] are not fault-tolerant.

Chapter 6

Fault Containment

In CHAPTER 4, an implementation of a general join module with input channels was given. Although the presented implementation uses only low-level building blocks, which are easier to implement in hardware, one could imagine implementing its basic modules by even more refined modules, comprising only very simple standard design building blocks like combinatorial logic gates (NAND, NOR, etc.) and channels. It is the topic of this chapter to investigate whether such very low-level implementations, which will be called *circuit implementations*, exist for a given module like, e.g., the pivotal $+/-$ counter module of the TG-Alg.

Rather than searching for positive answers, Section 6.1 will be concerned with showing that a Byzantine fault-tolerant circuit implementation does not exist for the so-called short-pulse-filter module, which is important for building stable storage. Interestingly, the degree of freedom of the faulty module that generates the input to be harmful is quite modest: The capability to generate a single pulse of arbitrary length suffices. From there it will be proven by reduction that a $+/-$ counter module, as well as a TG-Alg module, does not have a circuit implementation if just one of the inputs are allowed to be driven by a Byzantine faulty module: a Byzantine faulty module can produce an erroneous extra transition at a correct TG-Alg's output and thus violate fault-containment. Clearly, this negative answer is disappointing. However, there is still hope that the probability that a Byzantine faulty module can produce a glitch of a correct TG-Alg module is very low. This topic is investigated in Section 6.2.

6.1 Deterministic Fault Containment

In this section, we focus on the *short-pulse-filter* problem. We thus start with its definition:

6.1.1 The SPF Problem

We define the *parametrized short-pulse-filer* problem, abbreviated by $\mathbf{SPF}(T_{min}, T_L)$, with parameters $T_{min} > 0$ and $T_L > 0$, as follows: Consider a module with one input port i and one output port o. We will first define an environment for the input port.

Definition 6.1. *Let Env_{SPF} be the set of executions of input port i that fulfill at least one of the properties:*

- *There are constants $t_b > 0$ and $t_e \geqslant t_b$ such that $\widetilde{i}$ is well defined and*

$$\widetilde{i}(t) = \begin{cases} 0 & \text{if } t \in [0, t_b) \\ 1 & \text{if } t \in [t_b, t_e] \\ 0 & \text{if } t \in (t_e, \infty) \end{cases} . \tag{6.1}$$

  *We will call this input $\widetilde{i}(t)$ a $\langle t_b, t_e \rangle$-*pulse*.*

- *The input is a constant-0 signal, i.e., for all $t \geqslant 0$,*

$$\widetilde{i}(t) = 0.$$

□

Definition 6.2. *The $\boldsymbol{SPF}(T_{min}, T_L)$ problem is defined by: For every single input execution $\widehat{i}$ in Env_{SPF}, let the set of valid output executions $E_O(\widehat{i})$ for input $\widehat{i}$ be those $\widehat{o}$, which are non-Zeno (thus we may use state functions for their description) and further fulfill the properties:*

(No short pulse) *The output does not contain a short pulse, i.e.,*

$$\forall t_1 < t_2 < t_3 \in \mathcal{T} : \ (\widetilde{o}(t_1) = 0 \ \wedge \ \widetilde{o}(t_2) = 1 \ \wedge \ \widetilde{o}(t_3) = 0) \Rightarrow (t_3 - t_1 \geqslant T_{min})$$

(No generation) *If $\widetilde{i}(t) = 0$ for all $t \in \mathcal{T}$, then $\widetilde{o}(t) = 0$ for all $t \in \mathcal{T}$.*

(No muteness) *If the input pulse length $\Delta := t_e - t_b \geqslant T_L$, then the output must not be mute, i.e.,*

$$\exists t \in \mathcal{T} : \ \widetilde{o}(t) = 1. \tag{6.2}$$

□

We say that *module M solves the **SPF** problem in environment Env_{SPF}* iff there exist $T_{min} > 0$ and $T_L > 0$ such that M solves the $\mathbf{SPF}(T_{min}, T_L)$ problem in environment Env_{SPF}. Further, since we are only interested in solving the **SPF** problem in environment Env_{SPF} and no other environments, we abbreviate "M solves **SPF** in environment Env_{SPF}" with simply: "M solves **SPF**".

Note that the properties in Definition 6.2 do not prevent the output to oscillate, i.e., make transitions arbitrarily often, or steadily stay at 1 after a transition has occurred at the input signal. Clearly the above properties are an insufficient restriction of the allowed behavior of real-world short-pulse-filters. Those are typically intended to filter not only one short pulse, but multiple short pulses through its mission time, and are thus required neither to oscillate nor to produce a constant 1 from some $t \in \mathcal{T}$ on. Here, however, we are interested in impossibility results and we will show that even a short-pulse-filter that must adhere to only the above properties is does not admit a circuit implementation.

6.1.2 Circuit Implementation of a Module

The main goal of this chapter will be to show that there is no circuit implementation of the **SPF** module in environment Env_{SPF}. For this purpose, we formally define a circuit implementation module.

Given a module M, we are interested in whether there exists a physical implementation of M. Typically, a positive answer is given if the hardware designer comes up with a correct implementation — that is a collection of well-known basic gates (like NAND and NOT) and their wiring that behaves as the module M. In this section we are mainly concerned with impossibility results on the physical implementability of a module M, i.e., if we are able to deduce that no circuit implementation for M exists, this should suggest that there does not exist a physical implementation. By contrast, if we came up with a circuit implementation in our framework, this would not necessarily tell us whether there really exists a feasible physical implementation: Our solution might require ideal components that are not available in practice: constant delay channels (that do not jitter in time) and arbitrarily large (zero-delay) Boolean function modules.

We start our formalization of the concept of "circuit implementability" by introducing the *circuit implementation graph*, which will be helpful later on in specifying circuit implementations.

Circuit Implementation Graph. A *circuit implementation graph* $G = \langle I, L, O, f_B, E, f_0 \rangle$ is a directed graph with a finite set of nodes $V = I \cup L \cup O$ (input nodes I, local nodes L and output nodes O) and a finite set of weighted edges $E \subseteq V \times V \times (0, \infty)$ between its nodes. The intended meaning of $\langle a, b, \delta \rangle \in E$ is that there is a channel with delay (exactly) δ from node (port) a to node (port) b. We demand that nodes in I have in-degree 0 and nodes in $L \cup O$ have in-degree at least 1. Further, every node $v \in (L \cup O)$ has assigned a finite propositional formula $f_B(v)$ to it with its propositional variables from the set of

incoming edges $E_v = \{\langle x, v, \delta\rangle \mid \langle x, v, \delta\rangle \in E\}$. The meaning of $f_B(v)$ is the instantaneous output of v computed from the current output of its incoming channels. Function f_0 is a from E to $\mathbb{B}$ and assigns every edge e an initial value $f_0(e)$. The meaning of $f_0(e)$ is the initial value of the channel e.

Example 6.1. *An example circuit implementation graph* $G = \langle\{i\}, \{a\}, \{b\}, f_B, E, f_0\rangle$ *has nodes* $V = \{i, a, b\}$ *and edges* $E = \{\langle i, a, 1\rangle, \langle a, a, 2\rangle, \langle a, b, 1\rangle\}$. *Further let* $f_B(a) = \langle a, a, 2\rangle \wedge \langle i, a, 1\rangle$, *as well as* $f_B(b) = \neg\langle a, b, 1\rangle$. *The initial values are* $f_0(\langle i, a, 1\rangle) = 0$, $f_0(\langle a, a, 2\rangle) = 1$ *and* $f_0(\langle a, b, 1\rangle) = 1$. *See Figure 6.1 for a visualization of the graph.*

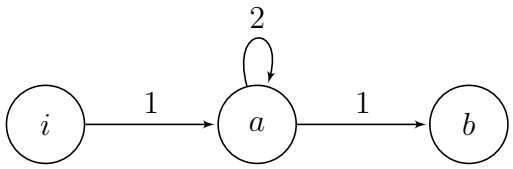

Figure 6.1: Circuit implementation graph of Example 6.1

Circuit Implementation Module. We define a *circuit implementation module* to be a composed module $M_I = \mathrm{Cm}(I, L', O, A)$, specified by a circuit implementation graph $G = \langle I, L, O, f_B, E, f_0\rangle$, where the set of local ports L' and the architecture A (comprising only of channels and Boolean function modules) are defined as follows:

(i) $L' := L \cup E$, i.e., the set of local ports contains all local ports of G and additionally one port per edge.

(ii) For every edge $e = \langle a, b, \delta\rangle \in E$ there is a channel $\langle a, \langle a, b, \delta\rangle, \delta, \delta, f_0(e)\rangle$ in A, with delay exactly δ and initial value determined by the function f_0.

(iii) For every node $v \in (L \cup O)$ there is a Boolean function module $\langle E_v, v, f_B(v)\rangle$, with input ports from the set E_v of incoming edges of v and Boolean function $f_B(v)$, in A.

Example 6.2. *A visualization of the circuit implementation module specified by the circuit implementation graph of Example 6.1, is depicted in Figure 6.2: ports are displayed by full dots together with their names. A channel with delay* δ *and initial value* v_0 *is drawn as a rounded rectangle with caption* $\delta_{(v_0)}$ *and Boolean function blocks are visualized as rectangles.*

We are now ready to specify what it means for a module M_I to be a *circuit implementation* of module M (in environment Env): it must hold that both M_I is a circuit implementation module and M_I implements M (in environment Env).

Given a module M (and environment Env), we say, that M *has a circuit implementation*, iff there exists a module M_I that is a circuit implementation of M (in environment

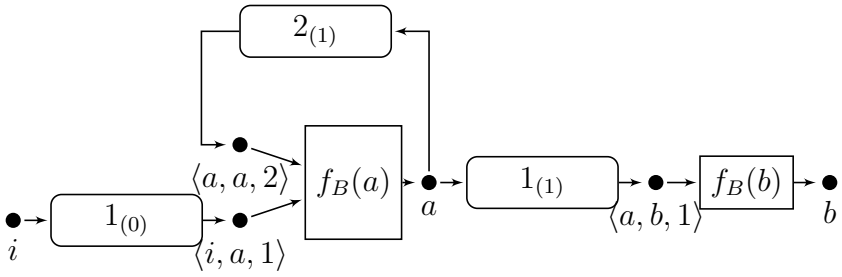

Figure 6.2: Circuit implementation module of Example 6.1

Env). Carefully note that the concept of a circuit implementation captures physical implementability in terms of purely binary building blocks, but not in terms of analogue components. Further, it is important to notice that the environment plays a key role when analyzing whether a given module has a circuit implementation module or not. Consider the following example:

Example 6.3. *The perfect C-Element problem, namely* $\boldsymbol{CELEMENT}(v_0, \delta)$*, where* $v_0 \in \mathbb{B}$ *is the initial value and* $\delta > 0$ *is the propagation delay is the problem* $\langle \{a, b\}, \{o\}, E\rangle$*, with* E *defined by: For all non-Zeno input executions* e_{in} *of ports* $\{a, b\}$*, the set of possible output executions* $E_O(e_{in})$ *consists of those* $\widehat{o}$*, for which: For all* $t \in \mathcal{T}$*:*

$$\langle 0, v_0\rangle \in \widehat{o} \text{ and}$$
$$\widetilde{a}(t) \wedge \widetilde{b}(t) \Leftrightarrow \langle t + \delta, 1\rangle \in \widehat{o} \text{ and}$$
$$(\neg\widetilde{a}(t) \wedge \neg\widetilde{b}(t)) \Leftrightarrow \langle t + \delta, 0\rangle \in \widehat{o}.$$

Notice that in the environment Env_1 *defined by*

$$\forall t \in \mathcal{T} : \widetilde{a}(t) = \widetilde{b}(t) = 0 \tag{6.3}$$

it can easily be proven that a Boolean function module that always produces 0 *implements* $\boldsymbol{CELEMENT}(0, \delta)$ *for arbitrary* δ*. Further, considering the environment* Env_2*, where* $\widetilde{a}$ *always changes before (and remains steady until)* $\widetilde{b}$ *changes, a single identity gate with input* b *and output* y *(Boolean function module* $Bfm(\{b\}, o, \widetilde{y}(t) = \widetilde{b}(t))$*) succeeded by a constant delay channel* $Ch(y, o, \delta, \delta, 0)$ *implements* $\boldsymbol{CELEMENT}(0, \delta)$*. Clearly for more complex environments, more complex implementation modules will be needed.*

An interesting question that will be answered later on in this chapter is whether **CELEMENT**$(0, \delta)$ has a circuit implementation for less restricted input environments at all. A positive answer here is of severe importance when trying to physically build a module like the **CELEMENT**$(0, \delta)$ module that can cope with faulty (Byzantine) input environments.

A first step towards this direction is the characterization of a circuit implementation module's properties.

Every Circuit Implementation Module is non-Zeno. We will now deduce that a circuit implementation module cannot produce Zeno traces out of non-Zeno traces. We start with auxiliary lemmata which state that channels and Boolean function modules cannot introduce Zeno behavior.

Lemma 6.1. *Let $C = Ch(i, o, \delta, \delta, v_0)$ be a channel with constant delay $\delta \in (0, \infty)$ and some initial value $v_0 \in \{0, 1\}$. If the input event trace $\widehat{i}$ is non-Zeno until time $t \in \mathcal{T}$, then the output event trace $\widehat{o}$ is non-Zeno until time $t + \delta$.*

Proof. Assume by contradiction that the input event trace is non-Zeno until t, but the output event trace is Zeno by time $t + \delta$. By definition, this implies that infinitely many transitions occur on port o during time $[0, t + \delta]$, that is, there is an infinite series $((a_j)_{j\in\mathbb{N}})$ of successive alternating-value events $a_j = \langle t_j, v_j\rangle$, with $t_j \in [d(0), t + \delta]$, occurring at o. Because of (3.10) and d being a bijection, an infinite series of successive, alternating-value, events $b_j = \langle t'_j, v'_j\rangle$ with $j \in \mathbb{N}$ and $b_j.t'_j \in [d^{-1}(d(0)), d^{-1}(t + \delta)] = [0, t]$ must have occurred at i, a contradiction to the assumption. ■

Lemma 6.1 enables us to safely make use of the state function in the channel's behavioral description if its input is non-Zeno. In the case where we do not mind about the exact input trace and corresponding output traces, as long as they behave equivalently in terms of state, (3.9) and (3.10) become

$$\widetilde{o}(t) = \begin{cases} v_0 & \text{if } t \in [0, d(0)) \\ \widetilde{i}(d^{-1}(t)) & \text{if } t \geqslant d(0) \end{cases}. \tag{6.4}$$

An analogous result for Boolean function modules, namely, that Boolean function modules cannot produce Zeno behavior out of non-Zeno behavior holds, too:

Lemma 6.2. *Let $B = Bfm(I, o, f)$ be a Boolean function module. If all input event traces $\widehat{i}$, where $i \in I$, are non-Zeno until time $t \in \mathcal{T}$, then the output event trace $\widehat{o}$ is non-Zeno until time t.*

Proof. Assume that all input event traces are non-Zeno until t. Thus for every input in $I = \{x_1, \ldots, x_n\}$ there exists a counting function $\#x_j$ until time t. By induction on $\sum_{1\leqslant j\leqslant n} \#x_j(t)$ it can be shown that

$$\#o(t) \leqslant \sum_{1\leqslant j\leqslant n} \#x_j(t), \tag{6.5}$$

i.e., that the number of transitions at the output port is at most the number of transitions occurring at all input ports. By definition of the non-Zeno property, $\#x_j(t)$ is finite for each $1 \leqslant j \leqslant n$. Thus by (6.5), $\#o(t)$ is finite or, which is equivalent, o is non-Zeno until time t. ■

Combining Lemmata 6.1 and 6.2 we can deduce the following result:

Lemma 6.3. *Let M_I be a circuit implementation module with circuit implementation graph G and let δ_{min} be the minimum of all its channels' delay times, that is, $\delta_{min} := \min\{\delta \mid \langle a, b, \delta\rangle \in G.E\}$. If the input execution e_{in} is non-Zeno until time $t \in \mathcal{T}$, then the output execution e_{out} is non-Zeno until time $t + \delta_{min}$.*

Proof. Assume that the input execution is non-Zeno until time t. Now assume by contradiction that the module's execution is Zeno by time $t + \delta_{min}$. Thus at least one of the non-input ports has an event trace that is Zeno by t. Let o be among the "earliest" becoming Zeno, in the sense that for some $t_{Zen} \in (0, t]$ and some $\varepsilon \in (0, \delta_{min})$ it holds that: all signals of non-input ports are non-Zeno until time $t_{Zen} - \varepsilon \geqslant 0$ and $\widehat{o}$ is Zeno by time t_{Zen}. Such a t_{Zen} and ε must exist, since all signals of non-input ports are non-Zeno until time 0 and at least one is Zeno by time t. To determine ε, we approach from both sides the point where at least one port's signal becomes Zeno and stop approaching when the difference between the left and right bound is $\varepsilon < \delta_{min}$.

Since o is a non-input port, it is the output of either (i) a Boolean function module or (ii) a channel. In case of (i), we apply Lemma 6.2, yielding that at least one of its inputs must have been non-Zeno by time t_{Zen}. Because every input of a Boolean function module is the output of a channel, (i) has been reduced to (ii). In case of (ii), let $o = \langle a, b, \delta\rangle$. Application of the negation of Lemma 6.1 implies that $\widehat{a}$ must have been Zeno by time $t_{Zen} - \delta \leqslant t_{Zen} - \delta_{min} < t_{Zen} - \varepsilon$, a contradiction. ■

Lemma 6.3 is of great importance for the subsequent analysis, as it allows to handle circuit implementation modules and their components by means of status functions rather than event traces. This considerably simplifies our proofs.

6.1.3 Dependence graph

We have introduced the concept of a circuit implementation graph, which allows to formally describe implementation modules. In this section we will concentrate on a technique for arguing about executions of circuit implementation modules. For this purpose we introduce the *dependence graph*. Given a circuit implementation module M (with output port o) and an input execution, a natural question is now to determine the value of $\widetilde{o}(\tau)$ for a given τ in $\mathcal{T}$. Intuitively, the dependence graph provides some means to answer this question, by providing us with all input signal values $\widetilde{o}(\tau)$ depends on:

Consider a circuit implementation module M with output o (possibly among others) specified by the circuit implementation graph G. Further let $\tau \in \mathcal{T}$ be an arbitrary instant in time. Then the *dependence graph* $D(G, o, \tau) = \langle V, K\rangle$ is a directed graph with the set of nodes V and the set of edges K defined as

$$V := \bigcup_{i \geqslant 0} \{\langle x, i\rangle \mid x \in V_i\} \qquad K := \bigcup_{i \geqslant 0} \{\langle\langle x, i+1\rangle, \langle y, i\rangle\rangle \mid \langle x, y\rangle \in K_{i+1}\}, \tag{6.6}$$

where the sets V_i, K_i make up the layers i of the dependence graph and are constructed by the rules defined below. In order to disambiguate between two identical nodes x in two different layers i and i', we extend the nodes in V with the layer number, obtaining $\langle x, i\rangle$ and $\langle x, j\rangle$. Every x actually is a tuple $\langle p, \delta\rangle$ where p is a port (output of a channel or function module) and δ is a time. In the following we define the sets V_i and K_i, for $i \geqslant 0$, as the smallest sets for which

- Base case (V_0, K_0):
  - $V_0 = \{\langle o, \tau\rangle\}$
  - $K_0 = \emptyset$
- Function module iteration step (V_i, K_i with $i \in \{1, 3, \dots\}$): For all $\langle s, \tau'\rangle \in V_{i-1}$ with arbitrary τ' and $s \in (G.L \cup G.O)$:
  - For all edges $e = \langle a, s, \delta\rangle \in G.E$ with arbitrary a and δ: $\langle e, \tau'\rangle \in V_i$ and $\langle\langle e, \tau'\rangle, \langle s, \tau'\rangle\rangle \in K_i$.
- Channel iteration step (V_i, K_i with $i \in \{0, 2, \dots\}$): For all $\langle e, \tau'\rangle \in V_{i-1}$ with arbitrary τ' and $e = \langle a, s, \delta\rangle \in G.E$:
  - If $\tau' - \delta \geqslant 0$: $\langle a, \tau' - \delta\rangle \in V_i$ and $\langle\langle a, \tau' - \delta\rangle, \langle e, \tau'\rangle\rangle \in K_i$.

Given a circuit implementation module $M = \langle I, O, E\rangle$ with output $o \in O$ and its dependence graph $D(G, o, \tau) = \langle V, K\rangle$, we call a node $x \in V$ a *leaf*, denoted by $x \in \mathbb{L}_\tau$, iff x has no incoming edges in K. Nodes that are not leafs are called *intermediate nodes*. We further call node x an *input leaf*, denoted by $x \in \mathbb{L}_\tau^I$, iff $x \in \mathbb{L}$ and x is of the form $x = \langle\langle s, \tau'\rangle, i\rangle$, with $s \in I$ (i.e., it is a module input) and arbitrary i and τ'. Note that non-input-leaf nodes are those where a further channel iteration step would have led to $\tau' - \delta < 0$. We further observe:

Observation 6.1. *Consider a circuit implementation module M and a dependence graph $D = \langle V, K\rangle$. For each intermediate node $x = \langle s, \tau\rangle$ in V_i, with arbitrary port s, $\tau \geqslant 0$ and $i \geqslant 0$, the value $\widetilde{s}(\tau)$ is a function of the values $\widetilde{s'}(\tau')$, where $\langle\langle s', \tau'\rangle, i+1\rangle$ is a child of $\langle\langle s, \tau\rangle, i\rangle$ in graph D.*

Observation 6.1 states that the value of an intermediate node just depends on the value of its children—justifying the term "dependence graph". We will next consider the value of the leafs: notice that leafs $\langle x, i\rangle$, $i \geqslant 0$, are always of one of two kinds: either (i) $\langle x, i\rangle = \langle\langle s, \tau\rangle, i\rangle$ is an input leaf, or (ii) $\langle x, i\rangle = \langle\langle e, \tau\rangle, i\rangle$, for $e = \langle a, b, \delta\rangle$ in $G.E$ and $\tau < \delta$ and a, b being ports of the circuit implementation module M.

In case of (i), the value of $\langle x, i\rangle$ is given by $\widetilde{s}(\tau)$ and in case of (ii), the value of $\langle x, i\rangle$ is given by $\widetilde{\langle a, b, \delta\rangle}(\tau)$. Since $\tau < \delta$, the value is equal to the initial value of the channel with input port a, output port b and delay δ. We thus observe:

Observation 6.2. *Consider a circuit implementation module M and a dependence graph $D = \langle V, K \rangle$. For each leaf $\langle x, i \rangle = \langle \langle s, \tau \rangle, i \rangle$ in V, with arbitrary port s, $\tau \geqslant 0$ and $i \geqslant 0$, the value $\widetilde{s}(\tau)$ is either (i) the status of input port s at time τ, or (ii) the initial value of one of M's channels.*

We will exemplify all this by means of a simple dependence graph:

Example 6.4. *An example dependence graph is depicted in Figure 6.3. It is the dependence graph $D(G, b, 7)$ with G from Example 6.1. The leafs are depicted as filled nodes, while intermediate nodes are empty. To enhance readability, we denote node $\langle a, \tau \rangle \in V_i$ with the string $\langle a(\tau), i \rangle$ (although related, not to confuse with the value $\widetilde{a}(\tau)$).*

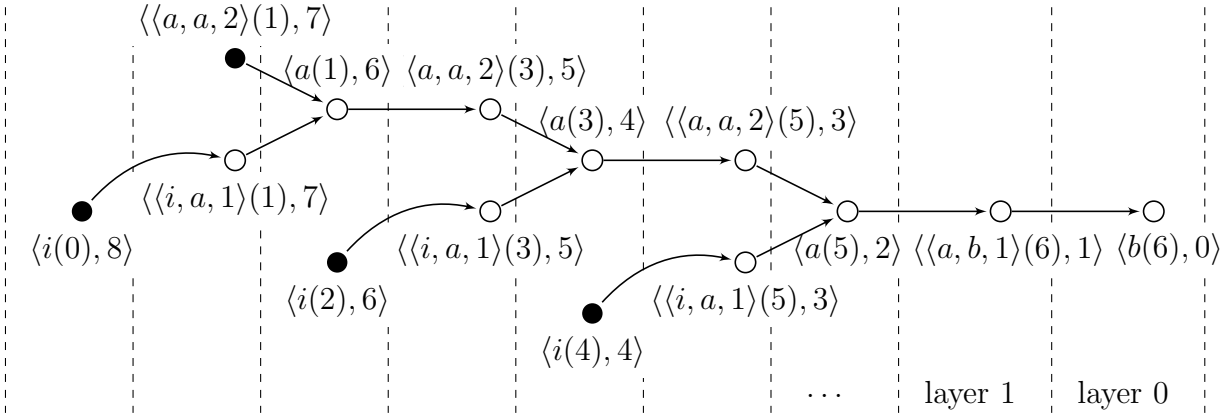

Figure 6.3: Dependence graph $D(G, b, 7)$ of Example 6.4

We will next prove a series of lemmata, which provide us with basic properties of dependence graphs. We start with:

Lemma 6.4. *For every node $\langle x, i \rangle \in V$ of the dependence graph $D(G, o, \tau)$, the set of paths $P = \{p_1, p_2, \dots\}$ from $\langle x, i \rangle$ to the root $\langle \langle o, \tau \rangle, 0 \rangle$ is non empty and the length of all paths is i.*

Proof. The proof is by induction on the layer $i \geqslant 0$.

Begin ($i = 0$): Node $\langle \langle o, \tau \rangle, 0 \rangle$ clearly is reachable from $\langle \langle o, \tau \rangle, 0 \rangle$ over the empty path of length 0. Since, by construction of D, there is no edge from a node $\langle x, i \rangle \in V$, with $i \geqslant 1$ to node $\langle \langle o, \tau \rangle, 0 \rangle$ the statement follows.

Step ($i \to i + 1$) with $i \geqslant 0$: Assume that, for all nodes $\langle x, i \rangle \in V$, the lemma holds. We will show that for all $\langle y, i + 1 \rangle \in V$, the proposition holds, too.

By construction of V, since $\langle y, i + 1 \rangle \in V$, $y \in V_{i+1}$. By the minimality of V_{i+1}, y has been added by one of the rules (i) or (ii). By these rules, there must be a node $x \in V_{i+1}$ with $\langle y, x \rangle \in K_{i+1}$. Thus $\langle \langle y, i+1 \rangle, \langle x, i \rangle \rangle \in K$. Therefore there is at least one path to the

root of length $i+1$. Since $\langle y, i+1\rangle$ has only outgoing edges to nodes $\langle a, i\rangle \in V$, and all $\langle a, i\rangle$ have paths to the root of length exactly i, $\langle y, i+1\rangle$ has paths to the root of length exactly $i+1$ only. ■

Let $D(G, o, \tau) = \langle V, K\rangle$ be a dependence graph. For all layers $V_i = \{\langle s, \tau'\rangle \mid \langle\langle s, \tau'\rangle, i\rangle \in V\}$, we define maxtime$(i)$ as the maximum time-instant of any of the nodes in layer V_i, that is,

$$\text{maxtime}(i) := \max\{\tau' \mid \langle s, \tau'\rangle \in V_i\}. \tag{6.7}$$

Recall the definition of δ_{min} from Lemma 6.3, which is the minimum of all the channels' propagation times. We now prove:

Lemma 6.5. *Let $D(G, o, \tau) = \langle V, K\rangle$ be a dependence graph. For any non empty layer V_i, with $i \geqslant 0$:*

(i) If $i = 0$, then maxtime$(i) = \tau$.

(ii) If $i \in \{1, 3, \dots\}$, then maxtime$(i) \leqslant$ maxtime$(i-1)$.

(iii) If $i \in \{0, 2, \dots\}$, then maxtime$(i) \leqslant$ maxtime$(i-1) - \delta_{min}$.

Proof. *(i)* is trivial.

(ii) In case $i \in \{1, 3, \dots\}$, if node $\langle e, \tau''\rangle \in V_i$, by the minimality of V_i, there must be a node $\langle s, \tau''\rangle \in V_{i-1}$. By (6.7), $\tau'' \leqslant \text{maxtime}(i-1)$.

(iii) In case $i \in \{0, 2, \dots\}$, if node $\langle a, \tau''\rangle \in V_i$, by minimality of V_i, there must be a node $\langle e, \tau'''\rangle \in V_{i-1}$, with $\tau''' = \tau'' + \delta$, for some channel delay δ. By (6.7), $\tau''' \leqslant \text{maxtime}(i-1)$ and thus $\tau'' + \delta \leqslant \text{maxtime}(i-1)$, i.e., $\tau'' \leqslant \text{maxtime}(i-1) - \delta_{min}$. ■

Lemma 6.6. *Any dependence graph $D(G, o, \tau) = \langle V, K\rangle$ has a finite number of nodes V and edges K.*

Proof. The number of nodes V_i and edges K_i is finite for every finite $i \geqslant 0$: V_0 and K_0 are finite and every node $x \in V_i$ adds only finitely many nodes y (which are all connected to x) to V_{i+1} by rules (i), resp., (ii). Thus the finiteness of a single layer follows by induction.

Further because of Lemma 6.5 in conjunction with $\delta_{min} > 0$ and the fact that nodes always have time values $\geqslant 0$, D has only finitely many non empty layers. Thus the total number of nodes V and edges K is finite. ■

Dependence Graphs with equivalent Structures. Consider a circuit implementation module M_I with circuit implementation graph G. We call two of its dependence graphs, $D(G,o,\tau) = \langle V, K\rangle$, for $\tau \geqslant 0$, and $D(G,o,\tau+\varepsilon) = \langle V', K'\rangle$, for $\varepsilon \in \mathbb{R}$, *equivalent in structure*, denoted by $D(G,o,\tau) \overset{\varphi_\varepsilon}{\approx} D(G,o,\tau+\varepsilon)$ or shortly $D(G,o,\tau) \approx D(G,o,\tau+\varepsilon)$, iff there exists a bijection $\varphi_\varepsilon : V \to V'$ such that,

(i) $\varphi_\varepsilon(\langle\langle a,\tau'\rangle, i\rangle) = \langle\langle a,\tau'+\varepsilon\rangle, i\rangle$ and

(ii) $\langle x,y\rangle \in K \Leftrightarrow \langle\varphi_\varepsilon(x),\varphi_\varepsilon(y)\rangle \in K'$.

We are now ready to prove the central lemma on equivalence, namely the *indistinguishability* of two input executions at two successive times $\tau \in \mathcal{T}$ and $\tau+\varepsilon$, for $\varepsilon \in \mathbb{R}$, at some output port o of a circuit implementation module:

Lemma 6.7. *Let $M_I = \langle I, O, E\rangle$ be a circuit implementation module with $o \in O$ and $D(G,o,\tau) = \langle V,K\rangle$ respectively $D(G,o,\tau+\varepsilon) = \langle V',K'\rangle$ its dependence graphs at times τ respectively $\tau+\varepsilon$. Further let e_{in} and e'_{in} be two input executions of ports I, and $e_{out} \in E_O(e_{in})$ and $e'_{out} \in E_O(e'_{in})$ be the corresponding output executions. If*

(i) $D \overset{\varphi_\varepsilon}{\approx} D'$ and

(ii) for all input leafs $\langle\langle s,\tau'\rangle, i\rangle \in V$ and $\varphi_\varepsilon(\langle\langle s,\tau'\rangle,i\rangle) = \langle\langle s,\tau'+\varepsilon\rangle,i\rangle \in V'$, it holds that $\widetilde{s}(\tau')$ in e_{in} is identical to the value $\widetilde{s}(\tau'+\varepsilon)$ in e'_{in},

then $\widetilde{o}(\tau)$ in e_{out} is identical to $\widetilde{o}(\tau+\varepsilon)$ in e'_{out}.

Since the proof is rather technical and lengthy, only the proof idea will be presented. Because of Lemma 6.4 and Lemma 6.6 we may use induction on the depth $i \geqslant 0$ to show that, for all nodes $\langle x,i\rangle$ in V the value of node x in the execution of M corresponding to input execution e_{in} is identical to the value of node $\varphi_\varepsilon(\langle x,i\rangle)$ in the execution of M corresponding to input execution e'_{in}.

The induction base ($i = \max\{i \mid \langle x,i\rangle \in V\}$) thereby makes use of Observation 6.2 and the induction step (from $i \to i-1 \geqslant 0$) uses both Observation 6.2 for leafs and Observation 6.1 together with the induction hypothesis for intermediate nodes.

From Lemma 6.7 we immediately obtain the following result:

Corollary 6.1. *Let $M_I = \langle I,O,E\rangle$ be a circuit implementation module with $o \in O$ and $\tau \in \mathcal{T}$ a time. For an input execution e_{in} of ports I, the set of possible output state values $\widetilde{o}(\tau)$, M_I might produce on e_{in}, is a singleton.*

Proof. Let $D(G,o,\tau) = \langle V,K\rangle$ respectively $D(G,o,\tau+\varepsilon) = \langle V',K'\rangle$ be M_I's dependence graphs at times τ respectively $\tau+\varepsilon$. Application of Lemma 6.7 with $\varepsilon = 0$ and input executions $e'_{in} = e_{in}$, yields, that the set of possible output state values $\widetilde{o}(\tau)$ is a singleton. ■

Furthermore we observe from Lemmata 6.6, 6.5 and $\delta_{min} > 0$ that a dependence graph can always be shifted by a small $\varepsilon > 0$ such that the resulting dependence graph has no additional nodes.

Observation 6.3. *Let $M_I = \langle I, O, E\rangle$ be a circuit implementation module with $o \in O$ and $D(G, o, \tau)$ its dependence graphs for some time $\tau \in \mathcal{T}$. There exists a $\varepsilon > 0$ such that $D(G, o, \tau + \varepsilon)$ has no leafs at time 0 and $D(G, o, \tau) \overset{\varphi_\varepsilon}{\approx} D(G, o, \tau + \varepsilon)$.*

6.1.4 Circuit Implementation Module Characterization

A circuit implementation module can be further characterized with respect to determinacy and causal behavior. These characterizations then lead to first impossibility results, which support the intuition on the non-existence of circuit implementations of modules that are not deterministic in state or not causal.

We start with a formal definition: module $M = \langle I, O, E\rangle$ *is deterministic* (in environment *Env*), iff (i) in case it has no inputs ($I = \emptyset$), $E = \{e_{out}\}$ is a singleton and (ii) in case it has inputs I, for each input executions $e_{in} \in Env$, there is only one corresponding output execution — that is, $E_O(e_{in}) = \{e_{out}\}$ is a singleton.

In case we do not care about the specific event trace but solely about the state of the execution, we are not interested in determinism but rather determinism "modulo state". Formally, we thus define: module $M = \langle I, O, E\rangle$ *is deterministic in state* (in environment *Env*), iff (i) in case it has no inputs ($I = \emptyset$), for each o in O, any two $e_{out}(o)$ and $e'_{out}(o)$, where e_{out} and e'_{out} in E, are elements of the same equivalence class $Event(\widetilde{o})$, for some status function $\widetilde{o}$; and (ii) in case it has inputs I, for each o in O and input execution $e_{in} \in Env$, any two $e_{out}(o)$ and $e'_{out}(o)$, where e_{out} and e'_{out} in $E_O(e_{in})$, are elements of the same equivalence class $Event(\widetilde{o})$, for some status function $\widetilde{o}$.

Example 6.5. *Let $M = \langle \emptyset, \{o\}, E\rangle$ be a module with no inputs and output o only. Assume that $E = \{e_{out}, e'_{out}\}$ with $e_{out}(o) = \{\langle 0, 0\rangle, \langle 1, 0\rangle, \langle 2, 1\rangle\}$ and $e'_{out}(o) = \{\langle 0, 0\rangle, \langle 2, 1\rangle\}$. Then M is not deterministic, but deterministic in state.*

Lemma 6.8. *Any circuit implementation module $M_I = \langle I, O, E\rangle$ is deterministic in state if $I = \emptyset$ and in the environment Env consisting of all non-Zeno input executions for $I \neq \emptyset$.*

Proof. Assume that $I \neq \emptyset$: consider an arbitrary output o in O and time $\tau \geqslant 0$. Further, since *Env* consists of all non-Zeno input executions, the input's state functions are well defined. Application of Corollary 6.1 reveals that the set of possible output state values $\widetilde{o}(\tau)$ is a singleton. Since this holds for arbitrary $\tau \geqslant 0$, it follows that M is deterministic in state in *Env*.

Otherwise, assume that $I = \emptyset$: by analogous arguments as above, it follows that M is deterministic. ■

We further say that module $M = \langle I, O, E\rangle$ with non empty I is *causal in environment* Env iff there exists a function $F_O(\cdot,\cdot)$, that maps an input execution prefix $e_{in}[\leqslant t]$ and a time $t \geqslant 0$ to a set $\{e_{out}[\leqslant t], e'_{out}[\leqslant t], \dots\}$ of output execution prefixes, such that, for all $t \geqslant 0$,

$$\forall e_{in} \in Env : \forall e_{out} \in E_O(e_{in}) : e_{out}[\leqslant t] \in F_O(e_{in}[\leqslant t], t) \tag{6.8}$$

and

$$\begin{aligned}&\forall e_{in} \in Env : \forall e'_{out}[\leqslant t] \in F_O(e_{in}[\leqslant t], t) :\\ &\exists e_{out} \in E_O(e_{in}) : e_{out}[\leqslant t] = e'_{out}[\leqslant t].\end{aligned} \tag{6.9}$$

Both that causality is defined with respect to event traces only and not modulo state. Intuitively, (6.8) makes sure that for a given input execution e_{in}, F_O contains at least the prefixes (until time t) of all the possible output executions e_{out} that M might produce when fed with input e_{in}. Thus (6.8) is a "minimum size" requirement on the set $F_O(e_{in}[\leqslant t], t)$ and it can be simply fulfilled by letting $F_O(e_{in}[\leqslant t], t)$ be the set of all output execution prefixes (until time t) with arbitrary behavior. By contrast, (6.9) is a "maximum size" requirement on $F_O(e_{in}[\leqslant t], t)$. Given an input execution e_{in}, it specifies that those output execution prefixes that are in $F_O(e_{in}[\leqslant\ t], t)$ must be the prefix of an allowed output execution for e_{in}. As such, (6.9) alone can be simply fulfilled by letting $F_O(e_{in}[\leqslant t], t) = \emptyset$.

We are now ready to prove that all circuit implementation modules are causal in non-Zeno environments:

Lemma 6.9. *Any circuit implementation module $M_I = \langle I, O, E\rangle$ with non-empty I is causal in the environment Env consisting of all non-Zeno input executions.*

Proof. We define F_O by its graph, namely

$$\begin{aligned}F_O &:= \{\langle\langle e_{in}[\leqslant t], t\rangle, Y(e_{in}, t)\rangle \mid e_{in} \in Env, t \geqslant 0\}, \text{ where}\\ Y(e_{in}, t) &:= \{e_{out}[\leqslant t] \mid e_{out} \in E_O(e_{in})\}.\end{aligned}$$

In case F_O is a function, both (6.8) and (6.9) are fulfilled by the construction of F_O. It thus remains to show that F_O indeed is a function.

Assume, by contradiction, that F_O is not a function. Then there exists some $\langle e_{in}[\leqslant t], t\rangle$ that is both mapped to set Y_1 and set $Y_2 \neq Y_1$. Thus there exist two input executions e_{in} and e'_{in} as well as an output execution e_{out} such that (i) e_{out} in $E_O(e_{in})$, but e_{out} not in $E_O(e'_{in})$ and (ii) $e_{in}[\leqslant t] = e'_{in}[\leqslant t]$.

By (i), there must exist an o in O and a time $t' \leqslant t$, such that the value $\widetilde{o}(t') = x$ for some x in $\mathbb{B}$ is allowed by module M when fed with e_{in} but not allowed by M when fed with e'_{in}[1], i.e., $\widetilde{o}(t') = \neg x$ must hold for M when fed with e'_{in}. Clearly, for all input

[1]Note that this must be the case, because o is the output of a Boolean function module, which allows arbitrary behavior of $\widehat{o}$ as long as $\widehat{o}$ has the correct status. By Observation 3.1, $E_O(e_{in})$ and $E_O(e'_{in})$ do not only differ by output executions that are equivalent in status.

executions in Env, the status function is well-defined. Application of Lemma 6.7 with $\tau = t'$ and $\varepsilon = 0$ reveals that the value $\widetilde{o}(t')$ is a function of t' and the status of the input ports at times $\leqslant t'$ only. However, the status of e_{in} and e'_{in} does not differ until time t' by (ii) — a contradiction to the different values $\widetilde{o}(t')$ for e_{in} and e'_{in}. ■

6.1.5 Impossibility results

At the beginning of this chapter, the naturally arising question whether any module M has a circuit implementation module M_I that implements M (in a given environment Env) was stated. With the characterizations of circuit implementation modules carried out in the former section, we know that this is not the case: a module M that is non-deterministic in state or acausal does not have a circuit implementation. As a simple example consider the following module:

Example 6.6. *Let* $\boldsymbol{PRED}(T) = \langle\{i\}, \{o\}, E\rangle$ *be the "predictive step" problem with one input* i*, one output* o *and prediction range* $T > 0$*. The input environment* Env_T *is defined to consist of only those* $\widetilde{i}$*, for which either (i) or (ii) below holds:*

(i) there exists a time $t_b \geqslant 2T$*, such that,*

$$\widetilde{i}(t) = \begin{cases} 0 & \text{if } t \in [0, t_b) \\ 1 & \text{if } t \in [t_b, \infty) \end{cases} \tag{6.10}$$

(ii) for all $t \geqslant 0$*,* $\widetilde{i}(t) = 0$*.*

For every input execution $e_{in} \in Env_T$ *of type (i), we define the set of valid output executions* $E_O(e_{in})$ *as those for which*

$$\exists t_o \in [0, t_b - T] : \widetilde{o}(t_o) = 1 \tag{6.11}$$

and for the input executions of type (ii), the only valid output execution is

$$\widetilde{o}(t) = 0. \tag{6.12}$$

PRED does not have a circuit implementation. By Lemma 6.8 and Lemma 6.9 we know that all circuit implementation modules behave deterministically in state and causally in the environment of non-Zeno input executions. Thus all that remains to show for the impossibility is to prove acausality:

Lemma 6.10. *For any* $T > 0$*,* $\boldsymbol{PRED}(T)$ *is not causal in the environment* Env_T*.*

Proof. Assume by contradiction that $\mathbf{PRED}(T) = \langle I, O, E\rangle$ is causal in environment Env_T. Clearly, Env_T is non-Zeno.

Then there exists a function $F_O(e_{in}[\leqslant t], t)$ as defined in (6.8) and (6.9). Now consider the two (non-Zeno) input executions e_{in} and e'_{in}:

$$e_{in}(i) = \widehat{i},\ \text{with } \widetilde{i}(t) = \begin{cases} 0 & \text{if } t \in [0, 2T) \\ 1 & \text{if } t \in [2T, \infty) \end{cases}, \text{ and}$$
$$e'_{in}(i) = \widehat{i'},\ \text{with } \widetilde{i'}(t) = 0$$

together with the output executions

$$e_{out}(o) = \widehat{o},\ \text{with } \widetilde{o}(t) = \begin{cases} 0 & \text{if } t \in [0, T) \\ 1 & \text{if } t \in [T, \infty) \end{cases}, \text{ and}$$
$$e'_{out}(o) = \widehat{o'},\ \text{with } \widetilde{o'}(t) = 0.$$

By definition of **PRED**, $e_{out} \in E_O(e_{in})$. Thus by (6.8), $e_{out}[\leqslant T] \in F_O(e_{in}[\leqslant T], T)$. Since $e_{in}[\leqslant T] = e'_{in}[\leqslant T]$, $e_{out}[\leqslant T] \in F_O(e'_{in}[\leqslant T], T)$.

By (6.9), there must exist an output execution $e''_{out} \in E_O(e'_{in})$, for which, $e''_{out}[\leqslant T] = e_{out}[\leqslant T]$. Since $\widetilde{e_{out}(o)}(T) = 1$, however, this contradicts (6.12). ■

SPF does not have a circuit implementation. After having shown the not too surprising result that **PRED** does not have a circuit implementation in environment Env_T, we proceed with proving the main result of this chapter, namely, that **SPF** does not have a circuit implementation in the single-pulse environment Env_{SPF} of Definition 6.1. Since the environment Env_{SPF} consists only of (t_b, t_e)-input pulses and the 0 signal, we are able to make some simplifications first.

The impossibility proof for **SPF** will be by means of contradiction. To this end we assume that a circuit implementation module M_I exists, such that M_I solves **SPF** in environment Env_{SPF}. We will next characterize M_I and introduce some useful notation: let G be M_I's circuit implementation graph and $D_\tau = \langle G, o, \tau\rangle$ be the dependence graph for some time $\tau \geqslant 0$. Recall that we denote the set of leafs of D_τ as $\mathbb{L}_\tau$ and the set of input leafs as $\mathbb{L}^I_\tau$. Given that the input signal is a $\langle t_b, t_e\rangle$-pulse, we partition the input leafs $\mathbb{L}_\tau$ into three sets: $\mathbf{0}_B$, $\mathbf{1}$ and $\mathbf{0}_A$, where,

$$\mathbf{0}_B := \{\langle i, \tau'\rangle \in \mathbb{L}^I_\tau \mid \tau' < t_b\},$$
$$\mathbf{1} := \{\langle i, \tau'\rangle \in \mathbb{L}^I_\tau \mid \tau' \in [t_b, t_e]\},$$
$$\mathbf{0}_A := \{\langle i, \tau'\rangle \in \mathbb{L}^I_\tau \mid \tau' > t_e\}.$$

The intended meaning of the sets $\mathbf{0}_B$, $\mathbf{1}$ and $\mathbf{0}_A$ is to contain those input leafs with value 0 before the pulse, those with value 1 during the pulse, and those with value 0 after the pulse, respectively.

Example 6.7. *As an example consider Figure 6.4, which depicts a $\langle 2, 5\rangle$-pulse together with the positions of the input leafs of some dependence graph. Here, $\mathbf{0}_B = \{\langle i, 0\rangle, \langle i, 1\rangle\}$, $\mathbf{1} = \{\langle i, 2.5\rangle, \langle i, 4\rangle, \langle i, 5\rangle\}$ and $\mathbf{0}_A = \{\langle i, 5.9\rangle\}$.*

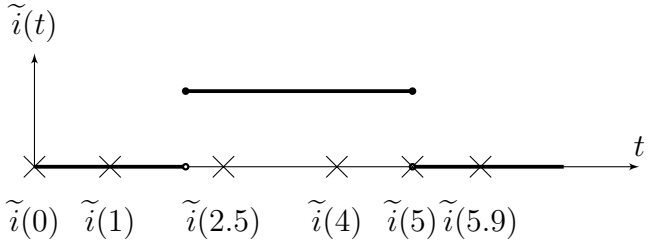

Figure 6.4: Input pulse with positions of input leafs marked.

We are now ready to prove:

Theorem 6.1. ***SPF*** *does not have a circuit implementation in environment Env_{SPF}.*

Proof. Assume by contradiction that **SPF** has a circuit implementation in environment Env_{SPF}, namely module M_I. The proof proceeds by the following steps: (i) A time $t \geqslant 0$, where output $\widetilde{o}(t) = 1$, is determined that will be used to construct the glitch (arbitrarily short pulse) of $\widetilde{o}$ around t. (ii) The shortest input pulse that still produces $\widetilde{o}(t) = 1$ is constructed. It is then shown by an indistinguishability argument that (iii) at an arbitrarily short time before t and (iv) at an arbitrarily short time after t, the status of the output must have been 0, contradicting the *(No short pulse)* property of the **SPF** problem in Definition 6.2. The theorem follows.

(i) Finding a time where $\mathbf{M}_I$ will glitch. Choose an arbitrary $\tilde{t}_b > 0$, and choose $\tilde{t}_e > \tilde{t}_b$ such that M_I reacts to input pulse $\langle \tilde{t}_b, \tilde{t}_e\rangle$ with $\widetilde{o}$, in the sense that there exists a $t' \in \mathcal{T}$ with $\widetilde{o}(t') = 1$. Such a $\tilde{t}_e$ must exist because of *(No muteness)*. Fix this t' and observe the dependence graph $D_{t'} = D(G, o, t')$. The point of time where we intend to generate a glitch will be called t'' and is obtained as follows:

- In case there is no leaf (may it be an input leaf or not) $\langle s, \tau'\rangle \in \mathbb{L}_{t'}$ with time $\tau' = 0$, we simply set $t'' := t'$ and $t_b := \tilde{t}_b$ as well as $t_e := \tilde{t}_e$.
- Otherwise, we add a small $\gamma > 0$ to t', i.e., $t'' := t' + \gamma$, such that $D_{t''} = D\langle G, o, t''\rangle$ has no leaf at time 0 and $D_{t''} \approx D_{t'} = \langle G, o, t'\rangle$. Such a γ exists by Observation 6.3. Further we shift the input pulse $\widetilde{i}$ by the same amount γ to the right, such that the $\langle \tilde{t}_b, \tilde{t}_e\rangle$-pulse becomes a $\langle t_b, t_e\rangle$-pulse, with $t_b := \tilde{t}_b + \gamma$ and $t_e := \tilde{t}_e + \gamma$. By definition of $D_{t''}$, $D_{t'} \approx D_{t''}$ and by the right shift of the input signal, both $D_{t'}$ with input pulse $\langle \tilde{t}_b, \tilde{t}_e\rangle$ and $D_{t''}$ with input pulse $\langle t_b, t_e\rangle$ have the same input leaf values. Thus by Lemma 6.7, it holds that $\widetilde{o}(t'') = \widetilde{o}(t') = 1$.

The existence of an input $\widetilde{i}$, being a $\langle t_b, t_e\rangle$-pulse with $0 < t_b < t_e$, and a time $t \geqslant 0$, such that $D_t = D(G, o, t)$ has no leafs with time 0 and produces $\widetilde{o}(t) = 1$ when fed with $\widetilde{i}$ follows.

In the following we call every input pulse that produces $\widetilde{o}(t) = 1$ a *one-pulse* and every input pulse that produces $\widetilde{o}(t) = 0$ a *zero-pulse*. Clearly, by construction, our $\langle t_b, t_e\rangle$-pulse is a one-pulse.

(ii) Searching for the shortest one-pulse. We now consider the set $P \subset \mathcal{T} \times \mathcal{T}$ of all tuples $\langle t'_b, t'_e\rangle$ (representing $\langle t'_b, t'_e\rangle$-pulses) with $t'_b \geqslant t_b$ and $t'_e \leqslant t_e$. For each such pulse let $\Delta := t'_b - t'_e$ be the *pulse length*.

Since $\langle t_b, t_e\rangle \in P$ and $\langle t_b, t_b\rangle \in P$, P consists of at least a one-pulse and a zero-pulse[2]. We denote the set of zero-pulses with P_0 and the set of one-pulses with P_1.

We are interested in the shortest (w.r.t. pulse length) pulse (or if there are more fulfilling this requirement: one of the shortest) that is a one-pulse. Let

$$\Delta^- := \inf\{t'_e - t'_b \mid \langle t'_b, t'_e\rangle \in P_1\}. \tag{6.13}$$

We first prove that the infimum is a minimum: since the value $\widetilde{o}(t)$ only depends on the structure of the graph D_t and the values of its finitely many input leafs, the shortest (infimum) pulse $\langle t'_b, t'_e\rangle$ must be one with an input-leaf at time t'_b and an input leaf at time t'_e; otherwise we arrive at a contradiction:

- If there is no input leaf at either t'_b or t'_e but an input leaf with time in $[t'_b, t'_e]$, then there is a shorter $\langle t''_b, t''_e\rangle$-pulse, with $t''_b \geqslant t'_b$ and $t''_e \leqslant t'_e$ and input leafs at both time t''_b and time t''_e such that: the input leaf values are identical for both the $\langle t'_b, t'_e\rangle$-input pulse and the $\langle t''_b, t''_e\rangle$-input pulse. Thus, by Lemma 6.7, $\widetilde{o}(t) = 1$ must hold in both cases — a contradiction to the minimal length of the $\langle t'_b, t'_e\rangle$-pulse.

- If there is no input leaf at either t'_b or t'_e and there is no input leaf with time in $[t'_b, t'_e]$, then the values of the input leafs are identical for both the $\langle t'_b, t'_e\rangle$-input pulse and the constant 0 input. By Lemma 6.7, $\widetilde{o}(t) = 1$ in both cases — a contradiction to *(No generation)*.

Since there are only finitely many input leafs $\mathbb{L}_t^I$ by Lemma 6.6 and by the above arguments, the shortest pulse's t'_b and t'_e must have times identical to input leaf times. As there are only finitely many combinations t'_b and t'_e, the minimum exists.

[2]By analogous but simplified indistinguishability arguments as in the proof of this theorem it can be easily shown that any $\langle t_b, t_b\rangle$-pulse must be a zero-pulse: If a $\langle t_b, t_b\rangle$-pulse were a one-pulse, the output o would produce zero-length pulses, contradicting *(No short pulse)* of **SPF**. For sake of better readability of the proof's key message we leave the proof of this simpler case to the reader.

(iii) Left-shift. We now proceed by observing the output value after a small left shift: we define

$$\begin{aligned}\tau_a &:= \min\{\tau' - 0 \mid \langle\langle s, \tau'\rangle, i\rangle \in \mathbb{L}_t \text{ and } \tau' \geqslant 0, i \geqslant 0\},\\ \tau_b &:= \min\{\tau' - t'_b \mid \langle\langle s, \tau'\rangle, i\rangle \in \mathbb{L}_t^I \text{ and } \tau' \in (t'_b, t'_e], i \geqslant 0\},\\ \tau_c &:= \min\{\tau' - t'_e \mid \langle\langle s, \tau'\rangle, i\rangle \in \mathbb{L}_t \text{ and } \tau' > t'_e, i \geqslant 0\}.\end{aligned}$$

In case any of these values is not well defined, we set it to $+\infty$. The meaning of τ_a, τ_b and τ_c is: τ_a is the maximum amount $D_t = \langle G, o, t\rangle$ can be shifted to the left, by say $\varepsilon_L > 0$, obtaining $D_{t-\varepsilon_L} = \langle G, o, t - \varepsilon_L\rangle$, without violating $D_t \approx D_{t-\varepsilon_L}$. Clearly, by a left shift, the input leafs with time t'_b are moved from $\mathbf{1}$ to $\mathbf{0}_B$. τ_b is the maximum amount D_t can be shifted to the left, without moving other leafs in $\mathbf{1}$ into $\mathbf{0}_B$. τ_c is the maximum amount D_t can be shifted to left, without moving any leafs in $\mathbf{0}_A$ into $\mathbf{1}$. Now choose an arbitrarily small

$$\varepsilon_L \in (0, \min\{\tau_a, \tau_b, \tau_c\})\,.$$

Clearly, $D_{t-\varepsilon_L} \approx D_t$. Further, by the definition of ε_L, the input leaf sets $\mathbf{0}_B{}', \mathbf{1}'$ and $\mathbf{0}_A{}'$ after the shift are:

$$\begin{aligned}\mathbf{0}_B{}' &= \{\langle s, \tau - \varepsilon_L\rangle \mid \langle s, \tau\rangle \in (\mathbf{0}_B \cup \{\langle i, t'_b\rangle\})\},\\ \mathbf{1}' &= \{\langle s, \tau - \varepsilon_L\rangle \mid \langle s, \tau\rangle \in (\mathbf{1} \setminus \{\langle i, t'_b\rangle\})\},\\ \mathbf{0}_A{}' &= \{\langle s, \tau - \varepsilon_L\rangle \mid \langle s, \tau\rangle \in \mathbf{0}_A\}.\end{aligned}$$

We are now interested in $\widetilde{o}(t - \varepsilon_L)$. It turns out that the input leaf sets $\mathbf{0}_B{}', \mathbf{1}'$ and $\mathbf{0}_A{}'$ are equivalent to those of D_t with the input forming a $\langle t'_b + \tau_b, t'_e\rangle$-pulse. Clearly the size of the latter pulse is smaller than Δ^- and thus by Lemma 6.7,

$$\widetilde{o}(t - \varepsilon_L) = 0. \tag{6.14}$$

(iv) Right-shift. We further observe the output value after a small right shift:

$$\begin{aligned}\tau'_a &:= \min\{0 - \tau \mid \langle s, \tau\rangle \in \mathbb{L}_t \text{ and } \tau \leqslant 0\},\\ \tau'_b &:= \min\{t'_b - \tau \mid \langle s, \tau\rangle \in \mathbb{L}_t \text{ and } \tau \in (0, t'_b)\},\\ \tau'_c &:= \min\{t'_e - \tau \mid \langle s, \tau\rangle \in \mathbb{L}_t \text{ and } \tau \in (t'_b, t'_e)\}.\end{aligned}$$

In case any of these values is not well defined, we set it to ∞. Further choose an arbitrarily small

$$\varepsilon_R \in (0, \min\{\tau'_a, \tau'_b, \tau'_c\})\,.$$

Clearly, $D_{t+\varepsilon_R} = D(G, o, t + \varepsilon_R) \approx D_t$. By the definition of ε_R, the leaf-sets $\mathbf{0}_B{}', \mathbf{1}'$ and $\mathbf{0}_A{}'$ after the shift are:

$$\begin{aligned}\mathbf{0}_B{}' &= \{\langle s, \tau + \varepsilon_R\rangle \mid \langle s, \tau\rangle \in \mathbf{0}_B\},\\ \mathbf{1}' &= \{\langle s, \tau + \varepsilon_R\rangle \mid \langle s, \tau\rangle \in (\mathbf{1} \setminus \{\langle i, t'_e\rangle\})\},\\ \mathbf{0}_A{}' &= \{\langle s, \tau + \varepsilon_R\rangle \mid \langle s, \tau\rangle \in (\mathbf{0}_A \cup \{\langle i, t'_e\rangle\})\}.\end{aligned}$$

The value $\widetilde{o}(t+\varepsilon_R)$ is obtained as follows: the leaf-sets $\mathbf{0}_B{}', \mathbf{1}'$ and $\mathbf{0}_A{}'$ are equivalent to those of D_t with the input forming a $\langle t'_b, t'_e - \tau'_c\rangle$-pulse. Clearly the size of the latter pulse is smaller than Δ^- and thus by Lemma 6.7,

$$\widetilde{o}(t+\varepsilon_R) = 0. \tag{6.15}$$

Since ε_L and ε_R may be chosen arbitrarily small, equations (6.14) and (6.15) violate property *(No short pulse)* of **SPF**. Thus M_I is not a correct circuit module implementation of **SPF** in environment Env_{SPF}. ■

Theorem 6.1 tells us that a certain problem, namely **SPF** does not have a circuit implementation even in quite a restricted environment Env_{SPF}, which allows only for single pulses of arbitrary length. The result is noteworthy, as it might have been suspected that for very complicated circuit implementation modules M_I, only very complicated input behaviors can make M_I's output glitch. This, however, is not the case. By the constructive nature of Theorem 6.1's proof, we even obtained an algorithm that, given any such module M_I, can construct a malicious input pulse, producing a glitch. Interestingly, as soon as we further restrict Env_{SPF} to contain only pulses of minimum length $\Delta_1 > 0$, or, more generally, does not contain pulses with length in $[\Delta_1, \Delta_2]$ for arbitrary $\Delta_1 < \Delta_2$, **SPF** turns out to have a circuit implementation. Let us call the latter environment $Env_{\mathrm{SPF}}(\Delta_1, \Delta_2)$. The circuit implementation module $M_{\mathbf{SPF}}$, with input port i and output port o, which solves **SPF** in $Env_{\mathrm{SPF}}(\Delta_1, \Delta_2)$ for arbitrary $\Delta_1 < \Delta_2$ is depicted in Figure 6.5. Its Boolean function modules are defined by the functions $f_B(a) = \mathrm{AND}$ and $f_B(o) = \mathrm{OR}$. The delay Δ_- is set to $\Delta_2 - \Delta_1$. $M_{\mathbf{SPF}}$'s correctness follows from the following lemma:

Lemma 6.11. *$M_{\boldsymbol{SPF}}$ solves the **SPF** problem in environment $Env_{SPF}(\Delta_1, \Delta_2)$ for arbitrary $\Delta_1 < \Delta_2$.*

Proof. We distinguish between two cases for the input behavior:

- In case the input has a constant-0 status, or is a pulse with $\Delta < \Delta_1$, $\widetilde{a}(t) = 0$, for all $t \geqslant 0$. Thus $\widetilde{o}(t) = 0$, for all $t \geqslant 0$, implying, that $M_{\mathbf{SPF}}$ fulfills *(No shortpulse)*, *(No generation)*.

- In case the input is a pulse with $\Delta \geqslant \Delta_2$, $\widetilde{a}$ is a pulse of length

  $$\Delta - \Delta_1 \geqslant \Delta_2 - \Delta_1 = \Delta_-,$$

  which is at least of the length of the feedback delay of o. Thus, the output o will once make a single transition from 0 to 1 and then steadily stay at 1. Clearly, therefore, *(No shortpulse)*, *(No generation)* and *(No muteness)* are fulfilled.

The lemma follows. ■

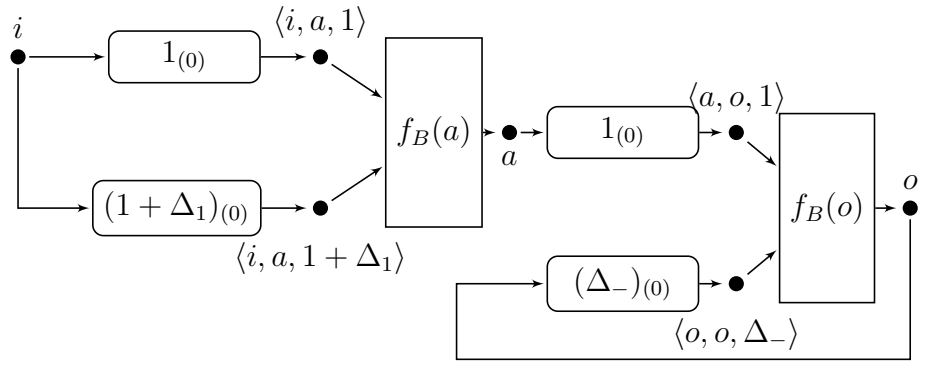

Figure 6.5: Circuit implementation module $M_{\mathbf{SPF}}$ solving **SPF** in a restricted environment.

6.1.6 Reducing circuit implementability

The result of Theorem 6.1 has some impact on the question of whether certain well-known modules have a circuit implementation. By means of reduction one can show that this question has to be negatively answered for many modules introduced so far in this thesis. For example the **CELEMENT**(v_0, δ) module, with initial value $v_0 \in \mathbb{B}$ and delay $\delta > 0$, which was introduced in Example 6.3, can be shown to have no circuit module implementation in environment Env_{SPF}: assume for the sake of a contradiction that it has a circuit module implementation M_I. Then we can construct a circuit module implementation for **SPF** by connecting (i) input port i over an arbitrary delay channel initialized with 0 to port a of M_I, and (ii) connecting input b of M_I with a subcircuit that steadily produces 1 (simply obtained by an OR gate and a feedback channel initialized with 1), and (iii) connecting the output port of M_I with o. Since this gives a circuit implementation module that solves **SPF** in environment Env_{SPF}, M_I cannot exist, which completes the proof.

Likewise, a $+/-$ counter which is among the key components of a TG-Alg module can be shown to have no circuit implementation if its remote input behavior is arbitrary within environment Env_{SPF}.

Relation to Fault-containment. What effects do these results have on the fault-containment of, e.g., a TG-Alg module? Assuming that the concept of circuit implementability of module M indeed captures physical implementability of M, they state that there are modules M whose input environments have to be further restricted in order for M to work correctly. Typically, if M is not stateless but has some memory, one such input constraint is a minimum duration of input pulses. Restricting M's input signals appropriately is not a problem at all, if the module driving M's input is correct and designed in a proper way to fulfill the additional requirements. However, assume that the predecessor module, call it B, that drives M's input is in a different fault-containment region than M. In case B is Byzantine faulty, it may provide an input signal that does not fulfill any restrictions. Since M is driven by an unrestricted input, it may behave in an arbitrary way itself, thereby leading to a violation of the fault-containment region. An

obvious workaround would have been to insert a short-pulse-filter in between B and M. Unfortunately, however, we have shown that there does not exist a circuit implementation of such a short-pulse-filter.

6.2 Probabilistic Fault Containment

We have shown that there does not exist a circuit implementation solving the **SPF** problem. For a circuit implementation, we can hence only hope that no input signal ever experiences a forbidden pulse that produces an output signal violating the **SPF** specification. It could be argued that the unsolvability result of the **SPF** problem with circuit implementation modules is overly pessimistic and suffers from over-abstraction: more accurate predictions of the behavior of physical components are obtained by modeling their signals' behavior with continuous-valued and differentiable status, leading to finite rise and fall times of transitions. Clearly an arbitrarily short pulse with zero rise and fall times, as allowed by our purely digital framework cannot be produced by a physical circuit. However, analog representations of digital circuits face another problem: metastability. Metastability is the problem[3] of a physical module that is bistable, i.e., has two stable internal states, to remain in an unstable state different from the two stable states for an unbounded time. An example of such a bistable module is the C-Element. Here metastability occurs if the C-Element's input signal changes such that the C-Element reacts with an output transition from, say, 0 to 1. In case the input changes back before the transition has completed (note that it has non-zero duration in an analog representation), the C-Element might end up in a metastable state, where it remains for an unbounded time. The adverse power of metastability is that (i) the unstable state may switch to any of the stable states at any time and (ii) it may lead to an output state that lies in between the voltage range of logical 0 or 1 for an unbounded time, thereby possibly being perceived differently by different successor modules: one may take the value as a 0 and one as a 1. Several realistic system models for physical modules [7, 53, 61, 68] have been utilized to prove the impossibility of implementing bistable modules that are required to be in a stable state within bounded time. By analogous means, a natural adaptation of the **SPF** specification in terms of continuous signals can be shown not to have a physical implementation [61]. Consider, for example, the physical implementation of a short-pulse-filter with a C-Element as presented in Section 6.1.6. Figure 6.6 shows SPICE simulation results of a C-Element based on detailed Bsim3v3 [9] transistor models. A set of input pulses together with their corresponding output pulses is shown: it can be observed that the shortest pulse (100ps length) leads to a spike at the output that returns to 0V before it reached the final 1.8V representing a logical 1. If we assume that it did not reach the threshold voltage of any of its successor modules, these consistently interpret the pulse as a steady logical 0 signal. The longest pulse (150ps length) already produces a 0-to-1 transition with normal slope. Pulses with $\Delta \in \mathcal{I} = [100, 150]$ps either produce spikes that possibly reach the successor modules'

[3]Not in the sense as **SPF** is a problem.

threshold voltages or may take arbitrarily long to reach the final steady state (either 1.8V or 0V). $\mathcal{I}$ is called *interval of forbidden pulse lengths*. Note that the metastable state is with high probability in the logical 1 region of all the successor modules here.[4] While this prevents inconsistent recognition at different successor modules, the output signal of the short-pulse-filter may drop to logical 0 at an arbitrary time after the transition to 1 — thereby possibly leading to a short pulse at the output port.

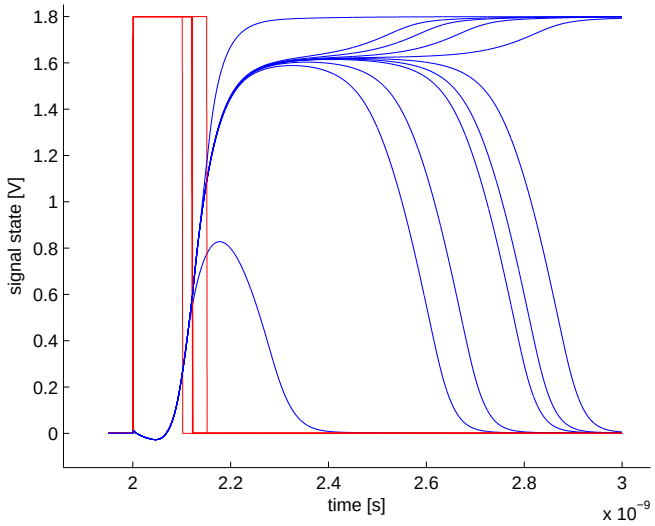

Figure 6.6: Simulation results of a C-Element short-pulse-filter implementation.

How do metastability and short-pulses relate? Despite the quite abstract, since purely digital, means used to describe circuits in this thesis, it is conjectured by the author that the presented digital framework reflects the metastability problem (continuous in value) in terms of arbitrary short pulses (discrete in value). The question is whether the concept of circuit implementation modules alone already captures all important aspects of metastability of analogue physical implementations. Unfortunately, this is not the case. Although a detailed treatment is outside the scope of this thesis and subject to future work, we will give ideas why this holds and how the formalism of circuit implementation modules could be extended. We start with a discussion of metastability upsets in physical implementations using the example of DARTS.

Consider the distributed system of TG-Algs presented in Chapter 5. Clearly, the $+/-$ counter modules in the TG-Algs comprise bistable modules and are thus susceptible to metastability. A faulty TG-Alg may thus provide an input signal that propagates through the $+/-$ counter and leads to a metastable upset of the PCSG module outputs, further leading to a metastable upset of the Threshold module and finally driving the Req generation module into an unstable state — a violation of fault-containment.

[4] Actually the voltage of the metastable state changes with the output load of the C-Element.

Although this scenario seems unlikely, an investigation of its likelihood is inevitable for a system intended for high-reliability applications like DARTS. In Fuchs *et al.* [34] it has been shown that a faulty TG-Alg that may produce pulses of arbitrary length is very unlikely to lead to a metastable upset of a correct TG-Alg's PCSG module's output: the remote pipeline, part of the $+/-$ counter, acts like a chain of synchronizers that decrease the possibility of a metastable upset of a PCSG output with each additional pipeline stage. Simulation results carried out in Matlab with simplified C-Element models (first order approximations of forward and backward path in the storage loop, calibrated by matching with simulations from detailed Bsim3v3 transistor models) have shown that the intervals $\Delta \in \mathcal{I}$ (where a pulse with length Δ at the remote pipeline's input leads to a metastability upset of the remote pipeline's output) decrease exponentially in length with the number of the remote pipeline's stages. This provides us with an effective method to make the probability of a metastable-upset across fault-containment regions arbitrarily small: concatenating an appropriate number of equivalent pipeline stages.

Now, consider the related **SPF** problem in our purely digital modeling framework. It was shown that module $M_{\mathbf{SPF}}$ from Figure 6.5 solves the **SPF** problem if the input environment is forbidden to contain pulses of length Δ in interval $\mathcal{I} = [\Delta_1, \Delta_2]$, with arbitrarily small $|\mathcal{I}|$. A concatenation of $M_{\mathbf{SPF}}$s forming a module chain, however, does not further reduce the interval of forbidden pulse lengths but rather enlarges it to $\mathcal{I} = [0, \Delta_2]$. Alternative short-pulse-filter designs seem to behave similarly: in contrast to their analogue counterparts, concatenation does not result in shrinking $|\mathcal{I}|$. It turns out that this is not an ultimate restriction of a purely digital framework. One way to capture the synchronizing behavior of concatenated circuit modules is to allow their channels to have more complicated delivery functions: for example, instead of a constant value one could allow delivery times that depend on the pulse length Δ. This would allow mapping of short pulses to even shorter pulses at the output. Finding appropriate delay functions that are in accordance with analogue models is subject to future work.

6.3 Related Work

Further reading: All the details, including the analytic model and simulation results, on the synchronizing behavior of the control structure of micropipelines can be found in [34].

Existing work: Besides the work by Lamport [53, 55], Marino [61] and Branicky [7], who all studied metastability in system models with continuous signal behavior, Anderson *et al.* [2] presented a proof for the impossibility of implementing an arbiter (a bistable module) in a purely discrete system model: the model is discrete in the signals' state, like the model presented in this thesis, with the major difference, however, that modules perform only a countable number (rather than continuous many) of discrete steps, like in the classical system models described in Section 2.1. Furthermore [2] does not allow to constrain channel delays.

Chapter 7

Conclusions

FAULT-TOLERANCE OF VLSI systems plays an important role, and is even gaining in relevance, not only for highly-reliable but also for everyday applications, because of decreasing transistor sizes. Appropriate failure mitigation techniques allow to increase both the yield and a circuit's probability to fulfill its specification during mission-time. To rigorously analyze such techniques, and to answer the question whether some circuit indeed solves a problem (respectively implements its specification), an investigation in terms of a formal framework is inevitable. Formalization approaches are typically of one of the following two kinds: (i) The circuit under investigation is specified in a framework that allows to assign probabilities to system executions. From these the probability that a system fulfills its specification can be calculated. (ii) A deterministic fault-hypothesis is formulated (e.g., up to f Byzantine faults may occur in any execution) and the circuit is proven to fulfill its specification in *each* of the executions that adhere to the fault-hypothesis. The latter can solely be carried out in a non-probabilistic framework. Finally, the probability that the circuit fulfills its specification at least is equal to the probability that the fault-hypothesis holds during mission time (the so called coverage). Although (ii) may lead to conservative bounds, its decoupling into a probabilistic and non-probabilistic analysis is usually the only feasible way to rigorously analyze real-world applications.

In this thesis, method (ii) was hence chosen and a framework for purely digital VLSI circuits following those of classic distributed systems was presented. We have seen that despite the convincing commonalities between classic distributed systems and modern VLSI systems, a number of key differences exist, most notably the absence of discrete computing steps in circuits, which prevents a trivial take-over of results from one community to the other. The formalism introduced in this thesis incorporates the peculiarities of chip-level computations, but in contrast to detailed electrical engineering system models, abstracts from analogue signals: a port's status can be 0, 1 or undefined at some time $t \in \mathcal{T}$. In general a port's behavior over time can be specified by three different means in our framework: most fundamentally by signals, by its status and finally by its counting function. This three-fold representations turned out to be flexible for specifying and analyzing

non-trivial circuits like the system of TG-Algs presented in Chapter 5.

The introduction of Petri-net-like languages in Chapter 4 allows writing specifications and correctness proofs of circuit modules at a more convenient level of abstraction. Nevertheless, it can be translated back into a set of relations on signal level at any point during the analysis if needed. Petri-net-like approaches are well known to be a useful way to model control structures of clockless circuits. To adequately express fault-tolerant control structures we restricted ourselves to the analysis of event graphs and, in a second step, introduced threshold graphs by adding so called threshold transitions, naturally capturing OR causality. The latter becomes inevitable when describing the behavior of clockless circuits in the presence of faults. General Join modules were found to be central circuit components in this context, as they generalize the well known Join modules with "wait-for-all" semantics to modules with "wait-for-k-out-of-n" semantics with arbitrary $0 \leq k \leq n$. Interestingly, General Joins have short representations in terms of threshold graphs. In a detailed treatment of General Joins, their timing properties have been analyzed and an implementation in terms of low-level building blocks has been presented and proven correct. These results not only showed the adequacy of the formal framework for non-trivial modules, but also allow a hardware designer to systematically incorporate fault-tolerance at control flow level into clockless circuits.

The importance of General Join modules was underlined in Chapter 5, where it was shown that a system of General Joins interconnected by channels, called DARTS, provides us with a solution of the tick generation problem in spite of a fraction of Byzantine faulty modules. This is of considerable practical importance, as a circuit fulfilling the tick generation specification can be used to provide synchronized clock signals to a set of replicated synchronously clocked functional units in a fault-tolerant system-on-chip. In [32], it was shown that the attainable synchronization of DARTS in realistic scenarios is in the order of a few clock ticks only.

Throughout the thesis faulty modules and thereby faulty General Joins were allowed to behave arbitrarily. Although a low-level circuit was shown to correctly implement a General Join in spite of possibly faulty predecessor modules in Chapter 4, the question arises whether an implementation in terms of simple Boolean gates and fixed delay channels (called circuit implementation) exists that implements a General Join even if some input is driven by a faulty module. Clearly, an affirmative answer is necessary if fault-containment of modules like the General Join shall be maintained. In order not to distract form the essence of the problem, the simpler question of whether a module with a single input and output port can filter out arbitrarily short pulses at the input in a non-trivial way (filtering all pulses as well as always producing a proper pulse at the output is both not allowed) was investigated. Unfortunately, it turned out in Chapter 6 that such a short-pulse-filter module does not have a circuit implementation and thus cannot be built by purely digital components. The chapter ended with a brief investigation of whether switching from digital to analogue system models helps to circumvent this impossibility result. However, this is not the case: in the latter, the problem of metastability is inevitable [61]. Fault-containment thus can only be ensured if faulty modules are disallowed to produce pulses

of lengths within an arbitrarily small, but non empty, interval.

7.1 Future Work

The author's aim was to provide a representative overview on a formal framework intended to argue about fault-tolerant on-chip algorithms and to show its usability by deriving non-trivial propositions. Along the main track, a number of interesting future research questions have been identified. We will briefly summarize some here:

- In Chapter 4 the General Join module was introduced. Since it is intended to be used in on-chip algorithms where modules may deadlock, General Joins comprise queues of unbounded size at their input ports, which allow only limited backpressure on the predecessor modules. For this reason the acknowledge output ports to the predecessor modules have been entirely removed when implementing a General Join J with a GJ_{Imp} module. However, in cases where no permanent but only transient faults are assumed to occur, General Joins with bounded queue size become interesting. Clearly, their treatment becomes more involved since backpressure can be passed from successor modules to predecessor modules, resulting in non-negligible queueing effects.

  Related to this topic is a refined analysis of queuing effects inside a GJ_{Imp} module: In this thesis queueing of transitions occurs because of the Diff-Gate only. All propagation delays of transitions through the local and remote pipes are conservatively mapped to the $\tau^{\pm}_{loc}$ and $\tau^{\pm}_{rem}$ delay bounds while the queues themselves are modeled by zero-delay queues of unbounded size. If this leads to unacceptable results, the queue models could be replaced by bounded size queues, built of separate stages that explicitly take queueing effects into account.

- The current DARTS solution of the tick generation problem turns transient faults into permanent faults: since only event messages (in contrast to status messages) are exchanged between the TG-Algs, spurious or missing transitions are memorized in the local and remote pipes at the respective TG-Algs. This is especially problematic if the probability of link failures is significant. Ideas on how to restore the pipe's status were presented in Section 5.3. A formal specification, correctness proof as well as a prototype implementation are planned future work.

- In Chapter 6 limitations of fault-containment regions in the presence of Byzantine faulty modules have been discussed. The main theorem of this chapter is an impossibility result of the implementability of short-pulse-filters with purely digital components. Many results on the non-existence of circuit implementation modules for other important problems, like the arbiter problem, follow by reduction. A detailed investigation of which problems can be reduced to which is a topic of future research.

Recently there has been made some progress[1]: It has been shown that there exists a circuit implementations of an *eventual short-pulse-filters*, i.e., a module that fulfills properties (no generation), (no muteness) and is allowed to glitch only finitely often.

It was conjectured that the impossibility to build short-pulse-filters with digital circuits relates to the impossibility to build metastability free circuits with continuous, analogue components. An in depth treatment of this research topic is of great importance as it would allow to abstract from analogue system models to purely digital models while still not loosing the essential characteristics.

[1] Unpublished note by Matthias Függer and Thomas Nowak

Bibliography

[1] M. Ampalam and M. Singh. Counterflow pipelining: architectural support for preemption in asynchronous systems using anti-tokens. In *ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 611–618, New York, NY, USA, 2006. ACM.

[2] J. H. Anderson and M. G. Gouda. A new explanation of the glitch phenomenon. *Acta Informatica*, 28(4):297–309, 1991.

[3] V. C. Barbosa and E. Gafni. Concurrency in heavily loaded neighborhood-constrained systems. *ACM Trans. Program. Lang. Syst.*, 11(4):562–584, 1989.

[4] R. Baumann. Soft errors in advanced computer systems. *IEEE Design & Test of Computers*, 22(3):258–266, May-June 2005.

[5] R. C. Baumann and E. B. Smith. Neutron-induced 10b fission as a major source of soft errors in high density srams. *Microelectronics Reliability*, 41(2):211 – 218, 2001.

[6] R. Bhamidipati, A. Zaidi, S. Makineni, K. Low, R. Chen, K.-Y. Liu, and J. Dalgrehn. Challenges and methodologies for implementing high-performance network processors. *Intel Technology Journal*, 6(3):83–92, Aug. 2002.

[7] M. S. Branicky. Why you can't build an arbiter. Technical report, Massachusetts Institute of Technology. Laboratory for Information and Decision Systems., 1993.

[8] C. Brej. *Early Output and Anti-Tokens.* PhD thesis, Department of Computer Science, University of Manchester, 2005.

[9] Bsim3v3 Standard. www-device.eecs.berkeley.edu/~bsim3/get.html.

[10] A. Bystrov, D. Sokolov, and A. Yakovlev. Low-latency contro structures with slack. *Asynchronous Circuits and Systems, International Symposium on*, 0:164, 2003.

[11] T. Calin, M. Nicolaidis, and R. Velazco. Upset hardened memory design for submicron CMOS technology. *IEEE Transactions on Nuclear Science*, 43(6):2874–2878, Dec 1996.

[12] K. M. Chandy and J. Misra. The drinking philosophers problem. *ACM Trans. Program. Lang. Syst.*, 6(4):632–646, 1984.

[13] D. M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems*. PhD thesis, Stanford University, Oct. 1984.

[14] B. Charron-Bost, S. Dolev, J. Ebergen, and U. Schmid, editors. *Fault-Tolerant Distributed Algorithms on VLSI Chips*, Schloss Dagstuhl, Germany, September 2008. http://drops.dagstuhl.de/opus/frontdoor.php?source_opus=1927.

[15] Y. Cheng. A survey of the theory of min-max systems. In D.-S. Huang, X.-P. Zhang, and G.-B. Huang, editors, *ICIC (2)*, volume 3645 of *Lecture Notes in Computer Science*, pages 616–625. Springer, 2005.

[16] W. A. Clark. Macromodular computer systems. In *AFIPS '67 (Spring): Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 335–336, New York, NY, USA, 1967. ACM.

[17] E. M. Clarke. Distributed computing issues in hardware design. *Distributed Computing*, 1(4):185–186, 1986.

[18] C. Constantinescu. Impact of deep submicon technology on dependability of VLSI circuits. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'02)*, pages 205–209, June 2002.

[19] C. Constantinescu. Trends and challenges in VLSI circuit reliability. *IEEE Micro*, 23(4):14–19, July 2003.

[20] C. Constantinescu. Neutron SER characterization of microprocessors. In *International Conference on Dependable Systems and Networks (DSN)*, 2005.

[21] J. Cortadella and M. Kishinevsky. Synchronous elastic circuits with early evaluation and token counterflow. In *DAC '07: Proceedings of the 44th annual Design Automation Conference*, pages 416–419, New York, NY, USA, 2007. ACM.

[22] J. Cortadella, M. Kishinevsky, and B. Grundmann. Synthesis of synchronous elastic architectures. In *DAC '06: Proceedings of the 43rd annual Design Automation Conference*, pages 657–662, New York, NY, USA, 2006. ACM.

[23] A. Dielacher, M. Függer, and U. Schmid. Brief announcement: How to speed-up fault-tolerant clock generation in VLSI systems-on-chip via pipelining. In *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC'08)*, page 423. ACM Press, Aug. 2008. An extended version is available as RR 15/2009, Institut für Technische Informatik, TU-Wien.

[24] A. Dielacher, M. Függer, and U. Schmid. How to speed-up fault-tolerant clock generation in VLSI systems-on-chip via pipelining. Research Report 15/2009, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2009.

[25] D. Dolev, J. Y. Halpern, and H. R. Strong. On the possibility and impossibility of achieving clock synchronization. *Journal of Computer and System Sciences*, 32:230–250, 1986.

[26] J. C. Ebergen. A formal approach to designing delay-insensitive circuits. *Distributed Computing*, 5:107–119, 1991.

[27] S. Fairbanks. Method and apparatus for a distributed clock generator, 2004. US patent no. US2004108876.

[28] S. Fairbanks and S. Moore. Self-timed circuitry for global clocking. In *Proceedings of the Eleventh International IEEE Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 86–96, Mar. 2005.

[29] M. Ferringer, G. Fuchs, A. Steininger, and G. Kempf. VLSI Implementation of a Fault-Tolerant Distributed Clock Generation. *IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT2006)*, pages 563–571, Oct. 2006.

[30] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, Apr. 1985.

[31] E. G. Friedman. Clock distribution networks in synchronous digital integrated circuits. *Proceedings of the IEEE*, 89(5):665–692, May 2001.

[32] G. Fuchs. *Fault-Tolerant Distributed Algorithms for On-Chip Tick Generation: Concepts, Implementations and Evaluations.* PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2009.

[33] G. Fuchs, M. Függer, U. Schmid, and A. Steininger. Mapping a fault-tolerant distributed algorithm to systems on chip. In *11th Euromicro conference on Digital System Design Architectures, Methods and Tools (DSD'08)*, pages 242–249, Parma, Italy, September 2008.

[34] G. Fuchs, M. Függer, and A. Steininger. On the threat of metastability in an asynchronous fault-tolerant clock generation scheme. In *15th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'09)*, Chapel Hill, N. Carolina, USA, May 2009.

[35] G. Fuchs, M. Függer, A. Steininger, and F. Zangerl. Analysis of constraints in a fault-tolerant distributed clock generation scheme. *3rd International Workshop on Dependable Embedded Systems (WDES'06)*, Oct. 2006.

[36] M. Függer. Fault-Tolerant Distributed Clock Generation in VLSI Systems-on-Chip. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstrasse 1-3/182-1, A-1040 Vienna, Austria, Apr. 2006.

[37] M. Függer, A. Dielacher, and U. Schmid. How to speed-up fault-tolerant clock generation in VLSI systems-on-chip via pipelining. In *Proceedings of the Eighth European Dependable Computing Conference (EDCC)*, 2010.

[38] M. Függer, G. Fuchs, U. Schmid, and A. Steininger. On the stability and robustness of non-synchronous circuits with timing loops. *3rd Workshop on Dependable and Secure Nanocomputing*, Jun. 2009.

[39] M. Függer, T. Handl, A. Steininger, J. Widder, and C. Tögel. An Efficient Test for a Transistion Signalling based Up-/Down-Counter. In *The Austrian National Conference on the Design of Integrated Circuits and Systems (Austrochip 2006)*, Oct. 2006.

[40] M. Függer and U. Schmid. Reconciling fault-tolerant distributed computing and systems-on-chip. Research Report 13/2010, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2010.

[41] M. Függer, U. Schmid, G. Fuchs, and G. Kempf. Fault-Tolerant Distributed Clock Generation in VLSI Systems-on-Chip. In *Proceedings of the Sixth European Dependable Computing Conference (EDCC-6)*, pages 87–96. IEEE Computer Society Press, Oct. 2006.

[42] E. M. Gafni and D. P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, 29(1):11–18, January 1981.

[43] J. Grahsl, T. Handl, and A. Steininger. Exploring the usefulness of the gate-level stuck-at fault model for Muller C-elements. In *Proceedings 20. Workshop für Testmethoden und Zuverlässigkeit von Schaltungen und Systemen (TuZ'08)*, pages 165–169, Vienna, Austria, Feb. 2008.

[44] J. Gunawardena. Causal automata. *Theoretical Computer Science*, 101(2):265 – 288, 1992.

[45] J. Gunawardena. Cycle times and fixed points of min-max functions. In *11th International Conference on Analysis and Optimization of Systems*, pages 266–272. Springer, 1994.

[46] J. Y. Halpern, N. Megiddo, and A. A. Munshi. Optimal precision in the presence of uncertainty. In *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 346–355, New York, NY, USA, 1985. ACM.

[47] B. Heidergott, G. J. Olsder, and J. von der Woude. *Max plus at work*. Princeton Univ. Press, 2006.

[48] International Technology Roadmap for Semiconductors, 2009.

[49] W. Jang and A. J. Martin. Soft-error robustness in QDI circuits. In *First workshop on System Effects of Logic Soft Errors (SELSE 1)*, April 2005.

[50] W. Jang and A. J. Martin. A soft-error tolerant asynchronous microcontroller. In *13th NASA Symposium on VLSI Design*, June 2007.

[51] I. Koren and Z. Koren. Defect tolerance in VLSI circuits: Techniques and yield analysis. *Proceedings of the IEEE*, 86(9):1819–1838, Sep 1998.

[52] C. LaFrieda and R. Manohar. Fault detection and isolation techniques for quasi delay-insensitive circuits. In *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*, page 41, Washington, DC, USA, 2004. IEEE Computer Society.

[53] L. Lamport. On the glitch phenomenon. Technical report, SRI Technical Report, 1976.

[54] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[55] L. Lamport. Buridan's principle. Technical report, SRI Technical Report, 1984.

[56] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[57] G. Le Lann and U. Schmid. How to implement a timer-free perfect failure detector in partially synchronous systems. Technical Report 183/1-127, Department of Automation, Technische Universität Wien, January 2003. (Replaced by Research Report 28/2005, Institut für Technische Informatik, TU Wien, 2005.).

[58] N. Lynch. *Distributed Algorithms*. Morgan Kaufman, 1996.

[59] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 137–151, New York, NY, USA, 1987. ACM.

[60] N. A. Lynch and F. W. Vaandrager. Forward and backward simulations for timing-based systems. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors, *REX Workshop*, volume 600 of *Lecture Notes in Computer Science*, pages 397–446. Springer, 1991.

[61] L. Marino. General theory of metastable operation. *IEEE Transactions on Computers*, C-30(2):107–115, February 1981.

[62] A. J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1:226–234, 1986.

[63] A. J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In *AUSCRYPT '90: Proceedings of the sixth MIT conference on Advanced research in VLSI*, pages 263–278, Cambridge, MA, USA, 1990. MIT Press.

[64] F. Mattern. On the relativistic structure of logical time in distributed systems. In *Parallel and Distributed Algorithms*, pages 215–226. Elsevier Science Publishers B.V, 1992.

[65] D. Mavis and P. Eaton. Seu and set modeling and mitigation in deep submicron technologies. In *Reliability physics symposium, 2007. proceedings. 45th annual. ieee international*, pages 293 –305, 15-19 2007.

[66] M. S. Maza and M. L. Aranda. Analysis of clock distribution networks in the presence of crosstalk and groundbounce. In *Proceedings International IEEE Conference on Electronics, Circuits, and Systems (ICECS)*, pages 773–776, 2001.

[67] M. S. Maza and M. L. Aranda. Interconnected rings and oscillators as gigahertz clock distribution nets. In *GLSVLSI '03: Proceedings of the 13th ACM Great Lakes symposium on VLSI*, pages 41–44. ACM Press, 2003.

[68] M. Mendler and T. Stroup. Newtonian arbiters cannot be proven correct. *Formal Methods in System Design*, 3(3):233–257, 1993.

[69] M. Merritt, F. Modugno, and M. R. Tuttle. Time-constrained automata. In *Proceeding of CONCUR '91*, pages 408–423, 1991.

[70] C. Metra, S. Francescantonio, and T. Mak. Implications of clock distribution faults and issues with screening them during manufacturing testing. *IEEE Transactions on Computers*, 53(5):531–546, May 2004.

[71] Y. Monnet, M. Renaudin, and R. Leveugle. Designing resistant circuits against malicious faults injection using asynchronous logic. *IEEE Transactions on Computers*, 55:1104–1115, 2006.

[72] D. E. Muller, W. S. Bartky, and J. Gunawardena. A theory of asynchronous circuits. In *of LNCS*, pages 278–292. Draft, 1959.

[73] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4), 1989.

[74] C. J. Myers and T. H. Y. Meng. Synthesis of timed asynchronous circuits. *IEEE Trans. VLSI Syst.*, 1(2):106–119, 1993.

[75] C. J. Myers, T. G. Rokicki, and T. H. Meng. Automatic synthesis and verification of gate-level timed circuits. Technical report, Stanford, CA, USA, 1994.

[76] E. Normand. Single-event effects in avionics. *IEEE Transactions on Nuclear Science*, 43(2):461–474, Apr 1996.

[77] J. L. Peterson. Petri nets. *ACM Comput. Surv.*, 9(3):223–252, 1977.

[78] T. Polzer, T. Handl, and A. Steininger. A metastability-free multi-synchronous communication scheme for fault-tolerant SoCs. Research Report 10/2009, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2009.

[79] P. J. Restle et al. A clock distribution network for microprocessors. *IEEE Journal of Solid-State Circuits*, 36(5):792–799, May 2001.

[80] U. Schmid. How to model link failures: A perception-based fault model. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'01)*, pages 57–66, Göteborg, Sweden, July 1–4, 2001.

[81] U. Schmid, J. Klasek, T. Mandl, H. Nachtnebel, G. R. Cadek, and N. Kerö. A Network Time Interface M-Module for distributing GPS-time over LANs. *J. Real-Time Systems*, 18(1):24–57, Jan. 2000.

[82] U. Schmid and A. Steininger. Dezentrale Fehlertolerante Taktgenerierung in VLSI Chips. Research Report 69/2004, Technische Universität Wien, Institut für Technische Informatik, 2004. (Österr. Patentanmeldung A 1223/2004).

[83] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Comput. Surv.*, 22(4):299–319, 1990.

[84] N. Seifert, P. Shipley, M. Pant, V. Ambrose, and B. GiII. Radiation-induced clock jitter and race. In *Proceedings 43rd Annual IEEE International Reliability Physics Symposium*, pages 215–222, 17-21, 2005.

[85] Y. Semiat and R. Ginosar. Timing measurements of synchronization circuits. *Asynchronous Circuits and Systems, International Symposium on*, 0:68, 2003.

[86] W. Snoeys, F. Faccio, M. Burns, M. Campbell, E. Cantatore, N. Carrer, L. Casagrande, A. Cavagnoli, C. Dachs, S. D. Liberto, F. Formenti, A. Giraldo, E. H. M. Heijne, P. Jarron, M. Letheren, A. Marchioro, P. Martinengo, F. Meddi, B. Mikulec, M. Morando, M. Morel, E. Noah, A. Paccagnella, I. Ropotar, S. Saladino, W. Sansen, F. Santopietro, F. Scarlassara, G. F. Segato, P. M. Signe, F. Soramel, L. Vannucci, and K. Vleugels. Layout techniques to enhance the radiation tolerance of standard cmos technologies demonstrated on a pixel detector readout chip. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 439(2-3):349 – 360, 2000.

[87] J. Sparsø and S. Furber. *Principles of Asynchronous Circuit Design*. Dimes, 2001.

[88] T. K. Srikanth and S. Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, July 1987.

[89] A. Steininger, M. Függer, U. Schmid, and G. Fuchs. Fault-Tolerant Algorithms on SoCs - A case study. In *Supplemental Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*, pages 190–191, Philadelphia, USA, June 2006.

[90] A. Steininger, T. Handl, G. Fuchs, and F. Zangerl. Testing the hardware implementation of a distributed clock generation algorithm for SoCs. *IEEE East-West Design and Test International Workshop*, pages 59–64, Sept. 2006.

[91] M. J. Stucki, S. M. Ornstein, and W. A. Clark. Logical design of macromodules. In *AFIPS '67 (Spring): Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 357–364, New York, NY, USA, 1967. ACM.

[92] I. E. Sutherland. Micropipelines. *Communications of the ACM, Turing Award*, 32(6):720–738, June 1989. ISSN:0001-0782.

[93] S. M. Sze and K. K. Ng. *Physics of Semiconductor Devices*. Wiley-Interscience, 3rd edition, 2006.

[94] P. Teehan, M. Greenstreet, and G. Lemieux. A survey and taxonomy of gals design styles. *IEEE Design and Test of Computers*, 24(5):418–428, 2007.

[95] T. Verdel and Y. Makris. Duplication-based concurrent error detection in asynchronous circuits: shortcomings and remedies. In *Proceedings 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2002)*, pages 345–353, 2002.

[96] J. Widder, G. Le Lann, and U. Schmid. Failure detection with booting in partially synchronous systems. In *Proceedings of the 5th European Dependable Computing Conference (EDCC-5)*, volume 3463 of *LNCS*, pages 20–37, Budapest, Hungary, Apr. 2005. Springer Verlag.

[97] J. Widder and U. Schmid. The Theta-Model: Achieving synchrony without clocks. *Distributed Computing*, 2009.

[98] A. Yakovlev, M. Kishinevsky, A. Kondratyev, L. Lavagno, and M. Pietkiewicz-Koutny. On the models for asynchronous circuit behaviour with or causality. *Formal Methods in System Design*, 9:189–233, 1996.

[99] Q. Zhou and M. Kartik. Transistor sizing for radiation hardening. In *IEEE international reliability physics symposium proceedings, 42nd annual*, 2004.

[100] J. F. Ziegler and W. A. Lanford. The effect of sea level cosmic rays on electronic devices. *Journal of Applied Physics*, 52(6), 1981.

Curriculum Vitae
Matthias Függer

Date of Birth:	March, 3^{rd}, 1983
Place of Birth:	Vienna, Austria
1989–2001	4 years Volksschule St. Ursula (Vienna) 8 years Oberstufenrealgymnasium St. Ursula (nat. wiss. Zweig mit Darstellender Geometrie). Matura: June 15, 2001.
2001–2004	Bachelor Curriculum Computer Engineering at the Vienna University of Technology. Graduation with distinction.
2004–2006	Master Curriculum Computer Engineering at the Vienna University of Technology. Graduation with distinction and nomination for the best thesis award. Sponsion at June 29, 2006.
2004–2009	Curriculum Philosophy at the University of Vienna (without graduation).
2006–2010	PhD Curriculum Computer Engineering at the Vienna University of Technology.
2006–2010	Research Assistant and then Assistant Professor at the ECS Group, Department of Computer Engineering, Vienna University of Technology.