FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Self-Stabilizing Byzantine Fault-Tolerant Clock Distribution in Grids

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Martin Perner
Matrikelnummer 0725782

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.-Prof. Dr. Ulrich Schmid

Wien, 28.08.2013

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Self-Stabilizing Byzantine Fault-Tolerant Clock Distribution in Grids

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

by

Martin Perner
Registration Number 0725782

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.-Prof. Dr. Ulrich Schmid

Vienna, 28.08.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Martin Perner
Oldtimerweg 1, 2353 Guntramsdorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

Acknowledgements

I would like to thank my family and friends for their support over the years that made the thesis possible. Special thanks go to Christoph Lenzen for his guidance with the proofs. Finally, I would like to thank my advisor for his feedback, suggestions and effort put into my thesis.

The work on this Master thesis was supported by the Austrian Science Foundation (FWF) under the project FATAL (P21694).

Abstract

This thesis presents design and analysis of a self-stabilizing Byzantine fault-tolerant clock distribution scheme HEX, which allows to distribute a clock signal in a hexagonal grid topology. Typical application domains are VLSI circuits, multi-core processors and other parallel/networked system architectures that require accurately synchronized clocks at neighbor nodes, e.g., for synchronous communication.

In sharp contrast to clock trees, which are commonly used for this purpose, HEX tolerates both persistent and transient faults of intermediate nodes and wires and supports multiple synchronized clock sources, as, e.g., used in the multi-synchronous GALS (globally asynchronous locally synchronous) approach. To achieve this, every node in the HEX grid is running a very simple distributed algorithm that forwards clock ticks and also provides the synchronized clock signal locally.

A VHDL implementation of the entire HEX algorithm is presented, which also incorporates a digitally controlled clock multiplier. By means of a comprehensive custom testbed, which also includes fault injection, HEX grids of variable size and with different delay parameters could be instantiated, simulated and post-processed. The entire design has been synthesized with the UMC 90 nm ASIC standard cell library, thereby generating a model that could be simulated using Mentor Graphics'® ModelSim.

Comprehensive experiments have been conducted to verify and complement the results of the theoretical worst-case analysis of the achievable synchronization accuracy (clock skew) and the stabilization time, which are also documented in this thesis. In particular, a suite of experiments revealed that the quite exotic worst-case scenarios are extremely unlikely to occur in practice, such that the typical average clock skew is much better than the worst-case. Experiments involving faulty nodes allowed us to also shed light on the excellent properties of HEX in the presence of a substantial number of failures, where analytic results are not available.

The results of this work, which has been supported by the Austrian Science Fund (FWF) project FATAL (P21694), have also been published at the 6th International Conference on Dependability (DEPEND'13) and the 25th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'13); a comprehensive journal version is currently under review.

Kurzfassung

Diese Diplomarbeit präsentiert Design und Analyse eines selbststabilisierenden, Byzantinisch fehlertoleranten Verfahrens (HEX) zur Taktverteilung in einer hexagonalen Grid-Topologie. Typische Anwendungsgebiete sind VLSI-Schaltungen, multi-core Prozessoren und andere parallele/netzwerkgekoppelte Systemarchitekturen, die, z.B. zu Kommunikationszwecken, genau synchronisierte Taktsignale in benachbarten Knoten benötigen.

Im Gegensatz zur Taktverteilung mittels einer Baumtopologie, die normalerweise hierfür verwendet wird, toleriert HEX sowohl persistente als auch transiente Fehler von Zwischenknoten und Verbindungsleitungen und unterstützt mehrfache synchronisierte Taktquellen, wie sie etwa in multisynchronen GALS (global asynchronen lokal synchronen) Architekturen verwendet werden. Um das zu bewerkstelligen, läuft auf jedem Knoten im HEX-Grid ein sehr einfacher verteilter Algorithmus, der Takte weiterleitet und auch lokal zur Verfügung stellt.

Zentraler Gegenstand der Arbeit ist eine VHDL-Implementierung des HEX-Algorithmus, die auch einen digital kontrollierten Taktmultiplizierer beinhaltet. Ein speziell entwickeltes Testbed, das auch Mechanismen zur Fehlerinjektion bereitstellt, erlaubt die Instantiierung, Simulation und das Post-Processing von HEX-Grids mit unterschiedlicher Größe und Zeitparametern. Das gesamte Design wurde mittels der UMC 90 nm ASIC-Standardzellen-Bibliothek synthetisiert, um ein für die Simulation mittels Mentor Graphics's® ModelSim geeignetes Modell zu generieren.

Umfassende Experimente wurden durchgeführt, um die Resultate der ebenfalls in dieser Arbeit dokumentierten theoretischen Worst-Case-Analyse der Synchronisationsgenauigkeit (Skew) und der Stabilisierungszeit zu verifizieren und, insbesondere, zu ergänzen. Diese bestätigten, dass die ziemlich exotischen Worst-Case Szenarien in der Praxis extrem unwahrscheinlich sind, sodass der typische mittlere Skew viel geringer als der Worst-Case ist. Experimente mit fehlerhaften Knoten, wo analytische Resultate nicht verfügbar sind, zeigten, dass HEX auch mit einer großen Anzahl fehlerhafter Knoten im Grid hervorragende Eigenschaften aufweist.

Die Resultate dieser Arbeit, die vom Österreichischen Fonds zur Förderung der wissenschaftlichen Forschung (FWF) im Rahmen des Projekts FATAL (P21694) unterstützt wurde, konnten auch in den Proceedings der 6th International Conference on Dependability (DEPEND'13) und des 25th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'13) publiziert werden; eine umfassende Journal-Version ist mittlerweile in Begutachtung.

Contents

CHAPTER 1

Introduction

Every digital system, except purely asynchronous ones, needs a clock signal at some level to operate correctly. In the extreme case of synchronous circuits, all components (flip-flops) in the circuit are driven by the same perfectly synchronous clock signal. The *globally asynchronous locally synchronous* (GALS) [9] approach opened the door to a new way of circuit design, which allows a variety of different clock domains on the same chip. The clock sources of the clock domains may be completely independent (standard GALS) or keep some synchrony relation with each other (multi-synchronous or mesochronous GALS [47, 60, 70]).

The GALS approach basically allows different parts of a chip to run independently of each other. Since a complete independence is a rare scenario in real applications, however, some sort of communication between these independent parts is needed. In a standard GALS system, this can only be achieved by means of handshake-based approaches. Unfortunately, besides performance issues and metastability concerns, such solutions easily suffer from deadlocks in the presence of faults. To avoid this, some sort of synchronous communication must be resorted to. The effort to accomplish this depend primarily on the properties of the sender and receiver clocks.

If, as in mesochronous clocking, in particular, those clocks are guaranteed to be at most a bounded number of clock ticks apart, at any time, metastability-free high-speed communication can be designed with little effort using FIFO buffers [55]. Since the ability to communicate correctly, and hence the proper operation of any higher-level application, depends crucially on the bounded synchrony of the clocks, however, critical multi-synchronous GALS systems require a highly fault-tolerant clocking system: Besides non-fault-tolerant distributed clock generation approaches like [32, 36, 42], there are also Byzantine fault-tolerant solutions like DARTS [29] that guarantee bounded synchronized clocks at all correct source nodes, despite of up to f source nodes that can behave arbitrarily faulty. However, as an instance of Byzantine agreement [38], it requires a total number of source nodes $n \geq 3f + 1$.

Since transient errors, e.g., due to ionizing particle hits, are the dominant cause of failures in modern *very-large-scale integration* (VLSI) circuits, however, exceeding $f \leq \frac{n-1}{3}$ faults is not an unlikely event. Unfortunately, classic Byzantine fault-tolerant solutions like DARTS cannot be

guaranteed to resume correct operation after such an event, even when all nodes operate correctly subsequently.

Self-stabilizing algorithms have been invented to cope with this problem [13]: They guarantee that the system will resume, within some finite stabilization time, correct operation even when started from an arbitrarily erroneous initial state. Byzantine fault-tolerant self-stabilizing clock generation methods like FATAL$^+$ [17, 18] achieve this even in the presence of up to $f \leq \frac{n-1}{3}$ permanent Byzantine faulty nodes. However, clock generation approaches like FATAL$^+$ are expensive in terms of required hardware and interconnections between the nodes, and hence do not scale well with the number of clock sources.

A simple way to mitigate this problem is to use a clock generation approach like FATAL$^+$ only to synchronize a small number of "primary" clock sources, and to distribute their clock signals to the remaining "secondary" clock sources, by means of a suitable clock distribution approach. Obviously, unlike ordinary clock-trees, the latter shall be Byzantine fault-tolerant and also self-stabilizing, but still use a sparsely connected topology.

This thesis presents design and analysis of a self-stabilizing Byzantine fault-tolerant clock distribution scheme HEX, which allows to distribute a clock signal in a hexagonal grid topology. Typical application domains are VLSI circuits, multi-core processors and other parallel/networked system architectures that require accurately synchronized clocks at neighbor nodes. In sharp contrast to clock trees, HEX tolerates both persistent and transient faults of intermediate nodes and wires and supports multiple synchronized clock sources, as, e.g., provided by primary nodes executing the FATAL$^+$ clock generation algorithm [17].

To achieve this, every non-primary node in the HEX grid is running a very simple distributed algorithm that forwards clock ticks and also provides the synchronized clock signal locally. A VHDL implementation of the entire HEX algorithm will be presented, which also incorporates a digitally controlled clock multiplier. By means of a comprehensive custom testbed, which also includes fault injection, HEX grids of variable size and with different delay parameters will be instantiated, simulated and post-processed. The entire design is synthesized with the UMC 90 nm ASIC standard cell library, thereby generating a model that can be simulated using Mentor Graphics'® ModelSim.

Comprehensive experiments will be conducted to verify and complement the results of the theoretical worst-case analysis of the achievable synchronization accuracy (clock skew) and the stabilization time, which are also documented[1] in this thesis. In particular, a suite of experiments will reveal that the quite exotic worst-case scenarios are extremely unlikely to occur in practice, such that the typical average clock skew is much better than the worst-case. Experiments involving faulty nodes will also shed light on the excellent properties of HEX in the presence of a substantial number of failures, where analytic results are not available.

The results of this work, which has been supported by the Austrian Science Fund (FWF) project FATAL (P21694), have also been published at the 6$^{\text{th}}$ International Conference on Dependability (DEPEND'13) [54] and the 25$^{\text{th}}$ ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'13) [16]; a comprehensive journal version [15] is currently under review.

[1] I acknowledge, though, that the core analytic results have primarily been obtained by my collaborators, in particular, by Christoph Lenzen (MIT).

This thesis is structured into 5 chapters: (The remainder of) Chapter 1 provides a short introduction to the basics of both fault-tolerant distributed computing and digital design. It also contains the definitions of some key terms, and briefly summarizes the (very few) existing related research. Chapter 2 is devoted to the theoretical analysis of HEX in the fault-free case taken from [16]. Chapter 3 describes the VHDL implementation of the HEX algorithm and the testbed used for our simulation evaluation, the results of which are presented subsequently. Chapter 4 is devoted to the design and analysis of the digitally controlled clock multiplier used to implement the synchronized fast clocks. Some conclusions and directions of future work in Chapter 5 round-off the thesis.

1.1 Computational Models in Distributed Computing

In this section, we will provide a short introduction to distributed computing based on the terms and notions used in [3].

Distributed computing is a field that focuses on the interaction between a set of independent *computing devices*, called *processes* or *nodes*, which try to achieve a common goal. This definition is widespread and covers many application domains, ranging from a VLSI chip to a cluster of computers to the Internet. Due to the spatial distribution of such systems, the processes need to communicate with each other, which is usually done via message-passing over dedicated links (on which we will focus here) or via shared memory.

Every process experiences *computation events* and *communication events*, where each such event is atomic and happens in zero time. A computation event causes the process to execute a computing step, which changes the state of the process performing it, based on received messages and the current internal state. Depending on the algorithm executed by the process, such a step can also include the sending of (multiple) messages. A communication event causes the delivery of a message, which was sent by another process, to the process performing the event. An *adversary* selects which process performs an event and thus determines the *execution* performed by the system. Usually, there are many possible executions that differ in the ordering of events, as the only dependency between two applicable events is that a message cannot be received before it is sent, i.e., the communication between two non-faulty processes is always causal.

Since no non-trivial distributed computing problem can be solved if the adversary never performs events on some processes, it is necessary to restrict the power of the adversary by a model of computation. These restrictions (called admissibility conditions) can include, e.g., the type and number of faults in the system, the general properties of the communication between the processes, or the maximal time between two events at the same process. There are many ways to define different models, yet there are two "extreme" models, that delimit the possible spectrum.

1.1.1 Asynchronous Model

This is the weakest and most general model. There are neither constraints on the time between two computation events at the same process nor on the time between the sending and receiving of a message, except that they must be finite and ≥ 0. Note that there are also no constraints on the time between two events on different processes. Since this puts almost no constraints

on the adversary (except that it must not starve a process forever and may not drop messages), many problems cannot be solved within the asynchronous model. For example, since a slow process cannot be distinguished from a crashed one, it is not possible to implement reliable failure detection.

1.1.2 Lock-Step Synchronous Model

This is the simplest model to analyze, albeit it is only applicable if all processes are equipped with synchronized clocks. The executions in this model are partitioned into a sequence of rounds. Every round consists of the sending of messages, the reception of all sent messages, and a computation event on every process. All processes execute their computation event simultaneously, in zero time, and all messages are sent immediately after the respective computation event has been executed. Hence, every round is perfectly aligned at all processes, which makes it easy to solve distributed computing problems. In particular, since every process must participate in every round, it is easy to reliably detect process crashes.

Assuming lock-step rounds is of course unrealistic in real systems, but models can be constructed that allow to simulate lock-step rounds. For this to achieve, the communication delays need to be bounded, the maximal time between consecutive computation event must be known, and every process must have a clock with bounded drift. Then, synchronized local clocks with some specific synchronization accuracy can be implemented, which in turn allow to build rounds that are in lock-step w.r.t. clock time.

1.1.3 Fault Models

As mentioned above, computational models must also define the types and maximum number of faults that can occur in the system. Doing this properly is not only of theoretical importance, but is also very relevant in practice. E.g., modern VLSI circuits are vulnerable to manufacturing defects and effects of radiation due to the miniaturization of the structures [5, 10]. There are certain applications, however, where a crash or a malfunction of the system is not tolerable, e.g., medical devices, power grids or aircraft. Properly designed fault-tolerant solutions, i.e., systems that do not fail despite a certain number of faults, are mandatory here.

Up to now, we only talked about faults, but never how they affect a process or the system at a conceptual level. Conceptually, there is a distinction between a fault, an error and a failure [4]. They interact in a chain, starting with the initial fault, as depicted in Figure 1.1. The initial fault can be an external fault, or an internal fault originating from a failure of a faulty process. A fault activates an error, which may causes a failure when it propagates. A failure in turn can cause a fault of another process.

Fault models only define the faults the system can experience, as the prevention of the activation of an error is in the scope of the specific algorithm. Faults are classified according to their temporal duration and their behavior toward the other processes of the system. First, we will look at the possible temporal behaviors:

Permanent Faults: These types of faults render a process erroneous from the time on when the fault occurs. This can be caused, e.g., by a non-recoverable corruption of the internal state

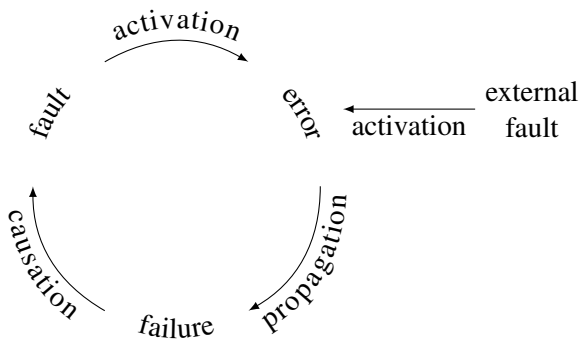


Figure 1.1: This figure shows the relationship between fault, error and failure. An initial external fault activates an error, which can propagates into a failure of a component, which in turn can cause a fault in another component, which then can activate an error.

due to an external fault or a hardware defect. Note carefully, though, that the erroneous state need not lead to a failure continuously.

Transient Faults: As the name suggests, these types of faults create erroneous states that last only for a limited timespan. Typically, transient faults are introduced by external effects, e.g., α-particles hitting a transistor or a power outage.

After the primary effect of the fault has ceased, the affected process works normally again, although its state may be contaminated and needs to be recovered so that the process can be considered non-faulty again (cf. Section 1.1.4). If the algorithm employed is not able to reintegrate such a process in the system, a transient fault has the same effect as a permanent fault.

With respect to the behavior of faulty processes towards other processes, we will briefly introduce the two most commonly assumed types of faults:

Fail-Silent Faults: Often referred to as crash fault, this type of fault is the simplest possible: A faulty process just stops sending messages. Difficulties may arise, though, when a process crashes in the middle of a broadcast.

Byzantine Faults: This type of fault is the least restricted, as a process can perform arbitrary actions. A Byzantine faulty process can behave differently to each of its neighbors, can cause timing violations of its messages, and even send incorrect messages.

Fault-tolerant computing has developed solutions that transparently mask failures caused by such faults. Conceptually, this is done by employing fault containment regions, which prohibit the activation of an error or at least its propagation across their boundaries. Some types of faults can be handled, to some degree, at the hardware level and are thus transparently masked w.r.t. the algorithm. For example, a corrupted memory bit can be handled by the hardware when the data is accessed by using an error-correcting code. This is an instance of information

redundancy (and also resource redundancy). A very popular form of fault masking by resource redundancy is replication in conjunction with some form of voting, as, e.g., used in *triple modular redundancy* (TMR). Replicated processes are the dominant approach in fault-tolerant distributed computing and can even deal with Byzantine faults [38]. Finally, time redundancy techniques, i.e., repeating a computation that has failed, are often used as a less costly alternative in cases where short response times are less important. For example, retransmitting a lost message is heavily used in network protocols.

1.1.4 Self-Stabilization

As fault masking techniques completely hide faults from becoming visible, they provide the best fault-tolerance one can hope for. Unfortunately, however, they can usually handle a limited number of faults only. For example, Byzantine distributed agreement [38] can be solved (in synchronous systems) only if at most f out of $n \geq 3f + 1$ processes can fail. If it ever happens that more than f processes behave faulty, the state of correct processes may be contaminated and hence erroneous. The system will not be able to recover from such a state, even when all n processes would work correctly afterwards, as it is the case for transient faults. Thus, a transient fault has the same effect on the system as a permanent fault, and a process struck by such a fault has to be considered faulty from that time on. Due to the increasing miniaturization in VLSI design, the rate of transient faults has increased [10], which negatively affects the overall fault-tolerance of such systems.

Another popular fault-tolerance technique is *failure detection, isolation and recovery* (FDIR), which cannot transparently mask failures, but guarantees that correct operation will eventually be resumed. The most powerful FDIR approach is a self-stabilizing algorithm. The term self-stabilization was introduced in [13] and specifies a system that, when started from any initial state, will reach a valid state in finite time, provided there is no fault during this stabilization time. The concept of self-stabilization has been extended later to also allow a limited number of permanent (Byzantine) faults [19]. To visualize the concept, consider the complete state space of a system as shown in Figure 1.2. Note that the states used here cover all processes in the system, not just the states of a single process, i.e., a state in the state space is the union of all states of the processes in the system here. This space can be subdivided into several state sets: Moving from the innermost set outwards, we first have the set of legal states, i.e., states in which the system operates correctly. This state set is enclosed by the set of safe states. These states have the property that every transition originating in them leads directly into the set of legal states. The last set is the set of pseudo-legal states, from which an execution exists that reaches a legal state. All states not covered by these sets are the remaining (erroneous) arbitrary states.

If the system is in a legal state and a "massive" fault occurs, then the next state can be outside of the legal state set. When the algorithm employed is self-stabilizing (and the allowed number of permanent faults is no longer exceeded) then the system will eventually reach a legal state again. This is called *convergence* and will happen within the stabilization time. When the system has converged, it will stay within the set of legal states (until another "massive" fault occurs), which is called *closure*.

As there are no restrictions on the states that can be reached due to a fault, this implies that the system can be started in an arbitrary state and will eventually converge to a legal state.

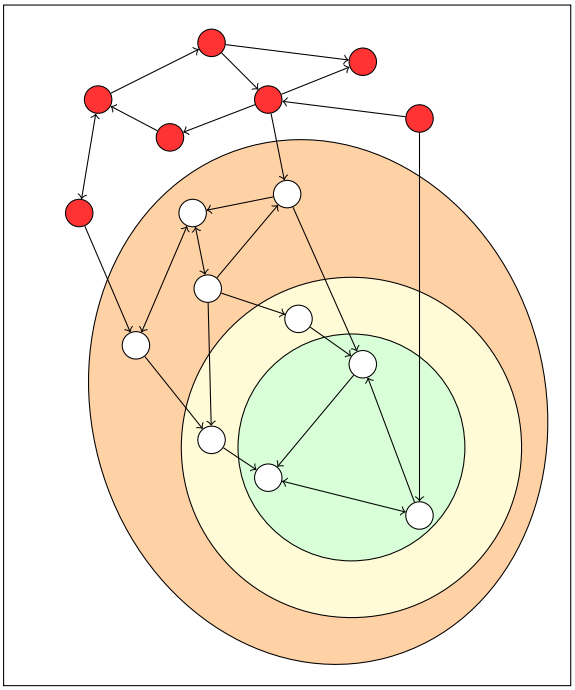


Figure 1.2: This figure shows the complete state space, which is a superset of multiple partial sets. The red nodes symbolize states which are arbitrary. The nodes in the orange area are pseudo-legal states, the nodes in the yellow area are safe states and the green area is the set of legal states.

1.2 Design Methodologies in VLSI

The design of hardware is nowadays primarily done using *very-large-scale integration* (VLSI), which is the integration of millions of transistors on a single die. In this section we will introduce relevant basics of digital circuit design and the design process. Further details may be found in [6, 65, 72], for example.

Digital circuits are designed at a reasonably high-level through *hardware description languages* (HDLs), where the two most common ones are Verilog and VHDL. They allow, at different abstraction levels, to combine different types of boolean logic gates to create combinatorial logic. This combinatorial logic is then processed by a design tool, where also a technology mapping for the intended target device is done. The technology mapping converts generic constructs of the design into specific elements available in the target device, which can be an FPGA or an ASIC. After the technology mapping is finished, a netlist is produced, which describes the final layout of the circuit on the target chip.

Yet, boolean logic gates can only implement stateless operations. As soon as some internal state must be maintained as well, which inevitably involves feedback loops, methods must be employed to ensure the correctness of the computational results. There are two fundamentally different approaches to achieve this:

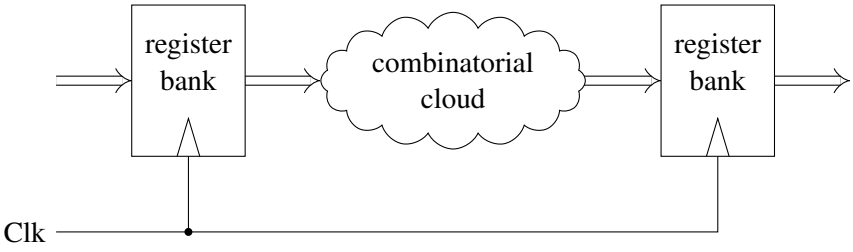


Figure 1.3: A section of a synchronous design. In the center there is the combinatorial cloud that consists of purely combinatorial logic. At every cycle of the clock, the results of the computation of the cloud is stored in the register bank on the right. The inputs for the computation stem from the left register bank. A possible feedback to the register bank on the left is not shown.

1.2.1 Synchronous Model

In the synchronous model, the components of the design can be separated into two groups, namely, (i) the register banks and (ii) the combinatorial clouds connected by parallel data paths, as shown in Figure 1.3. The combinatorial cloud is purely combinatorial logic, which is located between two register banks. Each register bank consists of at least one register, which is typically made up of flip flops. A register has a parallel data and a clock input, as well as a parallel data output. Typically, on the rising edge of the clock signal, the register samples the data on the input and forwards it to the data output, which is then held until the next clock cycle occurs. The required time between two clock edges is defined by the time needed for a signal to propagate through the *critical path*, i.e., the path through the combinatorial cloud with the longest signal delay between any two registers on the chip. The critical path is also influenced by the operating temperature and supply voltage of the chip, which affect the processing delays of the gates, and the delays of the wires connecting the components.

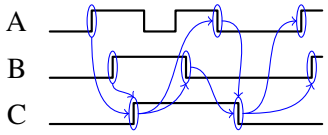
1.2.2 Asynchronous Model

Compared to the synchronous model, the asynchronous model is free of clocked components, i.e., registers. Results of the combinatorial cloud must hence be stored actively when they are available. Some form of handshaking is usually used for this purpose, but due to possibly different processing times of the different parallel paths, completion detection for a data path is a complex task. Completion detection can be done in the value domain, e.g., with encoding techniques like [12, 23, 46]. With these techniques, the data is encoded in a specific way to detect the completion of a computation via valid code words. Such techniques require 2 bits to encode 1 data bit, however, and thus suffer from increased hardware complexity.

Alternatively, completion detection can also be done in the time domain. Simple delayed handshake solutions can be employed in this case, if timing information for the critical path is available. Contrary to the synchronous model, in this scenario, one needs the individual critical path delays (per combinatorial cloud) here, and not the worst-case for the whole circuit. A very elegant approach introduced in [68] utilizes Muller C-Gates (see below) to capture the results of the computations. Nonetheless, delayed handshake solutions suffer from the same problems as

A	B	C
0	0	0
0	1	C_{-1}
1	0	C_{-1}
1	1	1

(a) Truth table of a Muller C-Gate. The inputs A and B define the output C. C_{-1} denotes the previous value of the output C.



(b) Timing diagram of a Muller C-Gate. As can be seen in the truth table on the left, the transitions of the output signal C are synchronized with the transitions of both inputs A and B. The blue markings show the dependence of the signal transitions on each other.

Figure 1.4: Truth table and timing diagram illustrating the behavior of a Muller C-Gate with the inputs A and B, and the output C.

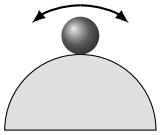


Figure 1.5: Basic principle of metastability. The ball can either go to the stable state on the left or right side of the saddle. When this happens is unknown and cannot even be bounded.

synchronous ones, as the critical path delays also vary with the temperature and supply voltage.

1.2.3 Muller C-Gate

A Muller C-Gate is a basic gate introduced in [49], which can be used for state-holding purposes in asynchronous circuits, like in [68]. In a nutshell, a Muller C-Gate can be seen as an AND-Gate for signal transitions.

The gate has two input ports and one output port. If the state of the input ports match, then their state is take as the new output state, cf. Figure 1.4a. I.e., a common state on the inputs is forwarded and held at the output. Therefore, the transitions at the output port are synchronized with the transitions at both input ports, as can be seen in Figure 1.4b.

1.2.4 What is Metastability?

Metastability [44] describes an unstable state of a stateful discrete system, from which the system will reach, in finite yet unknown time, a stable state.

This can be visualized with the simple physical model shown in Figure 1.5. The ball on top of the saddle will roll down on one of the sides, but to which side, and when this is going to happen, is unknown. In digital logic, the problem is similar. The output voltage of a transistor is an analog value, which is separated into three regions: Two regions describing the discrete values '0' and '1', and the forbidden region separating them. This forbidden region has to be crossed while switching from one state to the other. Although this is a completely normal operation, it

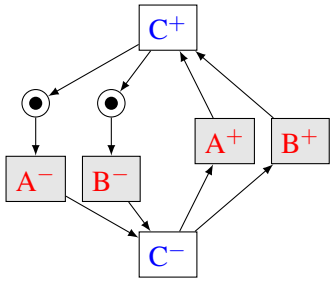


Figure 1.6: An example for a STG, which could be synthesized to a Muller C-Gate, with the inputs A and B and the output C. The nodes with $signal^+$ mark a rising transition of the signal, whereas $signal^-$ marks a falling edge of the respective signal. This is a correct STG, but does not describe the general behavior of a Muller C-Gate, as there are restrictions on the transitions of the input signals as well.

can cause an error if the signal is sampled by a memory element like a flip-flop whilst being in the forbidden region: The flip-flop can become metastable here and output an intermediate voltage for some unknown time (which, in turn, can cause the next downstream memory element to become metastable as well, i.e., metastability could spread in a system).

The same can happen within a flip-flop, due to the internal feedback loop, in case of a setup-/hold-violation. Flip-Flops require their input signal to be stable for a defined time before and after sampling, i.e., the clock transition, as the feedback loop needs some time to settle its state. In the synchronous model, metastability should not happen, as the time accounted for the critical path must incorporate these required time margins. However, input signals from outside the chip, or from other clock domains, can change their value at any time and thus induce metastability. Therefore, dedicated synchronizers, a couple of flip-flops connected in series, are employed for reducing the probability of metastability spreading to the real input of the circuit. Note that synchronizers increase the input signal propagation delay significantly (latency).

Metastability, in principle, can be avoided in asynchronous designs, as there are no clocked elements that have to make a decision in bounded time. Still, problems with external inputs remain, as they can behave arbitrarily anyway.

1.2.5 Signal Transition Graphs

A *signal transition graph* (STG) is basically a labeled Petri Net, which is often used for an event-based description of an asynchronous circuit and its environment. More specifically, STGs define the possible transitions of the signals and can be used to model a speed-independent circuit [11,49,61,64]. A speed-independent circuits functionality is independent of the processing delays of the gates used for implementing the circuit. Note that the wire delays are assumed to be zero here, but as these delays can be abstracted into the processing delays of the gates, this is not a problematic restriction.

A STG like the one shown in Figure 1.6 is basically a Petri Net with implicit transitions, i.e., there are only places (nodes) and arcs (edges) but no transitions in a STG:

Nodes: The nodes represent a single transition of a specific signal, used as the label of the node,

which can be rising (+) or falling (−). Nodes with the same label and polarity in the STG must have a node with the same label and alternative polarity between them.

Edges: The edges of a STG basically represent (some part of) the current state of the circuit. Due to this, and the alternation in polarity of the nodes, it is possible to infer the state of the system based on the currently active edges (see next item).

Token: A token marks a currently *active edge* in the graph. The tokens drawn in a STG define the initial state of the circuit. Unlike Petri Nets, just having tokens on all the incoming edges of a node alone, i.e., all incoming edges active, cannot trigger a transition: the signal transition represented by the node itself is also needed. I.e., the incoming edges of a node are pre-conditions for the execution for the respective transition. During the transition, the tokens on the incoming edges are removed and all outgoing edges are provided with tokens, i.e., the outgoing edges represent the post-condition of a transition.

The possibility to use a STG for the specification of an asynchronous system is convenient, as it allows to (almost) automatically generate a design directly from the specification by means of tools like Petrify [11] and SIS [61]. Yet there are limits: the functionality of the design has to be cyclic, in the sense that all involved signals must make both rising and falling transitions in a strictly alternating fashion. Therefore, there is no way to directly embed a reset signal into a design specified by a STG, for example.

Moreover, tokens cannot appear or disappear on edges in the STG. This is the reason why a STG does not only model a circuit but also its environment, i.e., its input signals. Take for example the STG shown in Figure 1.6. This STG could be synthesized to a single Muller C-Gate, which would indeed implement the defined transition relation. However, the STG does not describe a general Muller C-Gate, with arbitrary inputs, as the STG specifies that the inputs A and B change their respective state exactly once after the output C changed its state.

1.2.6 Functional and Timing Verification

Verifying the correctness of a circuit design is a very important part in the design process for VLSI chips, and usually requires high efforts and costs. By the rule of ten [73], there is an increase of one magnitude for the repair costs with each step in the manufacturing process, from the chip level to the board level to the system level and finally to system operation.

Compared to a software product, repairing a defect is usually not possible at a physical chip. Thus, also the verification process for a VLSI design is different from traditional software: As the production times and costs for a chip are high, it is unreasonable to depend solely on testing. The limited on-chip debugging possibilities also reduce the likelihood of reconstructing settings that caused faulty behavior, and support only limited test coverage as well. Thus, verifying the correctness of a circuit design relies heavily on simulation of the HDL code. These simulations are orders of magnitude slower than the execution of the same behavior in a physical chip, depending on the level of the simulation, but allow complete control over the *unit under test* (UUT). Note that this also allows to simulate the effects of faults, e.g., a broken wire, which are hard or impossible to analyze on a physical chip.

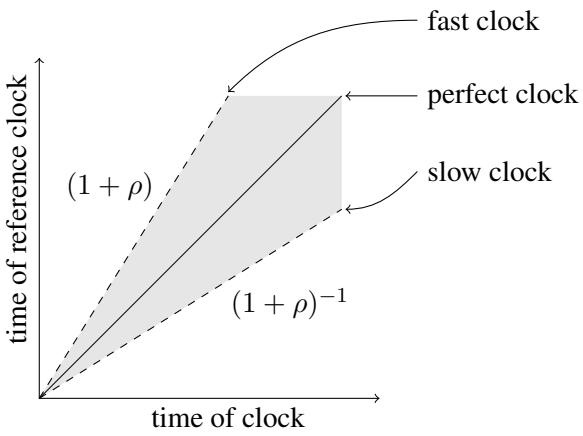


Figure 1.7: Visualization of the effect of the clock drift. The center line represents the perfect clock with no drift, the two outer line represent the minimal resp. maximal drift boundaries. These drift boundaries are reset after a resynchronization of the clock, as can be seen in the point of origin. A typical clock will be inside the gray envelope spanned by the slow and the fast clock.

A simulation environment simulates not only the UUT, but also the testbench that envelops the UUT. The testbench applies stimuli, corresponding to test cases, to the UUT and monitors the responses. Based on these responses, the testbench decides whether the results, provided by the design, are the correct results for the respective test cases. A testbench is called a testbed, when the simulation environment does not run the entire testbench. This includes cases where the testbench only provides the stimuli for the circuit and monitors the results, but applies these to a physical circuit (hardware-in-the-loop).

The abstraction level of simulations can range from a high-level structural simulation, which is rather fast, to the pre-layout simulation, where the design has already been mapped to specific hardware components of the target technology, and finally to the post-layout simulation, which is basically a simulation of the complete physical chip.

1.3 The Clock Distribution Problem

A clock is a device for measuring time and therefore produces ticks with a specific, fixed periodicity, the so called *granularity*. Yet, no clock is perfect, which means that the time between ticks may vary (although the granularity stays constant). These variations depend on many factors, from technological limitations to environmental influences, and can change over time. To specify a real clock, the derivation from an idealized, perfect, clock that displays real-time are used. We will now define the properties of the clocks used in this thesis.

Drift: The drift rate ρ_i is a multiplicative factor, which defines the minimal respectively maximal rate by which clock i (of process i) runs slower respectively faster than the perfect clock, cf. Figure 1.7. We denote with $t_i^{(k)}$ the real-time when clock i generates its k^{th} tick; note that the perfect clock reads $t_i^{(k)}$ at this time. The (short-term) drift rate is defined as the maximum

difference in time between two ticks of clock i measured with the perfect clock, divided by the granule n_i of the clock i.

$$\rho_i = \sup_{k \geq 1} \left| \frac{t_i^{(k+1)} - t_i^{(k)}}{n_i} - 1 \right|$$

Tick separation time: The tick separation time is the time between two ticks of a clock. $\Gamma_i^{\min}$ resp. $\Gamma_i^{\max}$ define the minimal resp. maximal tick separation time of clock i:

$$\Gamma_i^{\min} = \inf_{k \geq 1} \left(t_i^{(k+1)} - t_i^{(k)} \right)$$
$$\Gamma_i^{\max} = \sup_{k \geq 1} \left(t_i^{(k+1)} - t_i^{(k)} \right).$$

Note that $1/\Gamma_i^{\min}$ resp. $1/\Gamma_i^{\max}$ gives the maximal resp. minimal instantaneous clock frequency of clock i.

1.3.1 Synchronized Clocks

Due to the drift ϱ_i, clock i deviates from the perfect clock, or another clock j, after some time. To keep clocks close to each other, it is necessary to resynchronize those. The largest possible deviation between two clocks is called *precision* if measured in clock time, and *skew* if measured according to the perfect clock.

Precision: The precision π is an upper bound on the difference in ticks between two correct clocks read at the same time. Let $b_i(t)$ be defined as $b_i(t) = \sup \left\{ k | t_i^{(k)} \leq t \right\}$, then

$$\forall t, i_{\text{correct}}, j_{\text{correct}} \colon |b_i(t) - b_j(t)| \leq \pi$$

Skew: The skew is the difference in time between the same tick of two different clocks, i.e., $\left| t_i^{(k)} - t_j^{(k)} \right|$. Therefore, like the precision, the skew gives a measure for the tightness of the synchronization between different clocks. The skew manifests itself as a phase shift between the generated clock ticks.

1.3.2 Clock Generation

Instead of resynchronizing free-running clocks, a set of processes may also generate clock ticks in a synchronized fashion.

Definition 1.3.1 (Clock Generation): For a set of processes (called primary clock sources), a correct clock generation algorithm ensures that all correct clock sources generate clocks with a given maximal skew σ_0 and given minimal and maximal tick separation time in $[\Gamma^{\min}, \Gamma^{\max}]$.

There are several algorithmic solutions for both problems (synchronization and generation), which differ in the assumptions made on the system and their fault-tolerance properties [2, 17, 18, 20, 24, 59, 76] (see Section 1.4). However, all solutions with reasonable fault-tolerance properties require fully-connected communication topologies.

1.3.3 Clock Distribution

Clock distribution is a technique related to clock generation. As the name suggests, clock ticks, which are generated by one or more synchronized *clock sources*, are being distributed through a suitable *distribution network*. The clock source depends on the specific application domain but can, e.g., be a crystal oscillator or a clock generation algorithm running on a set of processes. We will refer to a clock tick as *pulse*, once it has entered the distribution network.

Definition 1.3.2 (Clock Distribution): Given a set of primary clock sources with skew σ_0 and pulse separation time in $[\Gamma^{\min}, \Gamma^{\max}]$, a correct clock distribution algorithm for a set of nodes ensures that all correct nodes eventually generate pulses with a given maximal skew σ_{ij} between all correct nodes i and j.

The distribution network can range from a single wire, which transports a signal wave corresponding to the pulse, to highly complex distribution networks consisting of nodes which forward *trigger messages*. Byzantine fault-tolerant clock distribution networks require the reception of multiple trigger messages from different nodes before the local clock tick can be triggered at a node. Thus, every node must be able to distinguish between trigger messages corresponding to different pulses: If there would be no way of detecting the corresponding clock tick of a trigger message, old trigger messages could circulate in the network and spuriously trigger nodes and generate more such trigger messages.

Basically, there are two approaches for solving this problem: (i) using some kind of high-level message to label the trigger messages in some way, or (ii) ensuring that the pulse separation time is large enough so that it is not possible for two trigger messages, corresponding to different pulses, to be mixed up. The first approach is used in theoretical and practical work [27, 66], and is required if trigger messages can circulate in the network. The second approach is the simplest approach possible, as it works with anonymous messages / clock signal transitions. It may decrease the maximum clock frequency, however, as, due to multiple paths, trigger messages can be in transit for a long time.

1.4 Related Work

We are not aware of any related work on Byzantine fault-tolerant clock distribution. We can hence survey some less-related research only.

The capability to distributed a clock to a large number of nodes in a synchronized fashion is essential in many areas, though without considerations for fault-tolerance. Examples are VLSI circuits and other hardware devices, which are the primary focus of this thesis, or a set of distributed nodes in a master-slave topology either on a wired [33,48] or wireless network [30,67]. In VLSI circuits, clock trees are used to provide a high-frequency clock to parts of a chip with synchrony requirements in the range of a fraction of the cycle time [69]. Especially in synchronous designs, skew and power consumption are major problems nowadays due to the rising clock frequencies, so much effort is put into the construction of networks which minimize the skew [25, 39, 40, 56, 57, 62, 77]. In the area of 3D clock distribution for chips fault-tolerance has become an issue, but up to now only benign (fail-silent) faults have been considered [41, 43].

Distributed clock synchronization is an old, well-known problem. It was an active research topic in the 1980s [37, 66, 76], primarily as a real-world application of the consensus problem. More recent research in this area focused primarily on wireless sensor networks [67], self-stabilization [20] and VLSI implementations [27, 28]. The need for distributed clock generation stems from the fact that traditional clock sources, e.g., quartz oscillators, are difficult to control and also highly susceptible to faults. This susceptibility is also a problem for the existing non-fault-tolerant approaches for distributed clock generation [21, 22, 31, 32, 35, 36, 42, 45, 58], which use distributed ring oscillators, *phase locked loops* (PLLs) or similar constructs. Besides Byzantine fault-tolerant solutions like DARTS [28, 29], there are also clock generation algorithms like FATAL$^+$ [17, 18], which are Byzantine fault-tolerant and self-stabilizing. Since they require a fully-connected network, however, they do not scale to the large number of nodes targeted by HEX.

For didactic reasons, related work on clock multiplication is presented in Section 4.3.

CHAPTER 2

HEX

"Kein Hirt und eine *Herde! Jeder will das Gleiche, jeder ist gleich: wer anders fühlt, geht freiwillig ins Irrenhaus."*
— Friedrich Nietzsche*, Also sprach Zarathustra*

This chapter presents HEX, which is an abbreviation for *hexagonal grid*. HEX comprises an algorithm and a corresponding network structure, which together provide a fault-tolerant, self-stabilizing clock distribution scheme. The name arises from the structure of the neighbors of a node in a HEX grid, which resembles a hexagon, cf. Figure 2.1. This specific network structure was chosen as it is planar and regular. Furthermore, as the average node degree in a planar graph must be < 6,[1] the HEX grid has also the largest possible node degree for a planar graph while still being regular.

This chapter will first provide HEX-related definitions and the analysis of the achievable clock skew in the fault-free case. Subsequently it provides a discussion of possible fault models and their consequences for the skew. Finally, the pulse separation time, which is crucial for the self-stabilizing property, will be determined.

2.1 Topology

HEX is based on a cylindric, directed graph $G = (V, E)$, which is parameterized by $L \in \mathbb{N}$, the *layers* of the grid, and $W \in \mathbb{N}$, the *columns* of the grid. The node set V defines the vertices of the grid and is the set of tuples $(\ell, i) \in [L + 1] \times [W]$. The edge set E, for every node $(\ell, i) \in V$, consists of the following links, over which zero-length messages can be sent:[2]

- $((\ell, i), (\ell, i + 1 \bmod W))$ and vice versa to/from the right neighbor for $\ell \neq 0$.

[1] This follows from Euler's formula. With $3f \leq 2e$, which is a basic property of any graph, and $\sum_{v \in V}(deg(v)) = kv$, with k being the average node degree, this can be shown easily.

[2] Note that, in an actual implementation, the timeouts calculated in Section 2.5 must be increased by the duration needed to transmit a message.

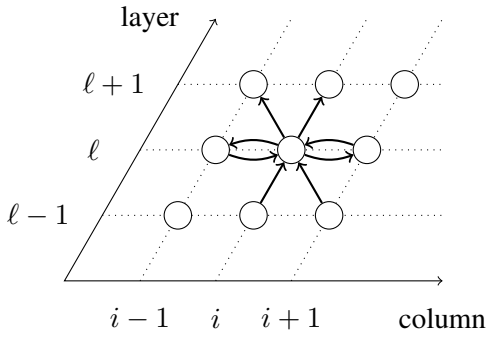


Figure 2.1: The node (ℓ, i) and its neighboring nodes. Column coordinates are modulo W, layer coordinates are between 0 and L. The structure of the connected neighbors form the name-giving hexagon.

- $((\ell, i), (\ell, i-1 \bmod W))$ and vice versa to/from the left neighbor for $\ell \neq 0$.
- $((\ell-1, i), (\ell, i))$ from the lower-left neighbor iff $\ell \geq 1$, and $((\ell, i), (\ell+1, i))$ to the upper-right neighbor iff $\ell \leq L$.
- $((\ell-1, i+1 \bmod W), (\ell, i))$ from the lower-right neighbor iff $\ell \geq 1$, and $((\ell, i), (\ell+1, i-1 \bmod W))$ to the upper-left neighbor iff $\ell \leq L$.

As it can be seen by the definition of the edges, HEX has the structure of a open cylinder. Thus all calculations on the grids columns are modulo W, without being mentioned explicitly.

Assumption 2.1.1: Each edge has a delay associated with it, which can vary between $[d^-, d^+] \subset (0, \infty)$. We assume that

$$\varepsilon = d^+ - d^- \leq \frac{d^+}{2}.$$

2.2 Algorithm

Algorithm 1: Pulse forwarding algorithm for nodes in layer $\ell > 0$.

```
1 forever
2   on received trigger message from (left and lower-left neighbor) or
      from (lower-left and lower-right neighbor) or
      from (lower-right and right neighbor)
      do
3     broadcast trigger message;                    // local clock pulse
4     sleep for some time within [T^-, T^+];
5     forget previously received trigger messages;
```

Every node in the HEX grid executes the algorithm shown in Algorithm 1. The node is constantly checking if a trigger condition becomes true, i.e., if the node received trigger messages from one of the three defined node pairs. If the node has been triggered, it broadcasts a trigger message to its outgoing neighbors and then goes to sleep. After the node wakes up, it deletes all trigger messages which were received since the last clearance and repeats checking the trigger conditions.

2.3 Skew Analysis

Definition 2.3.1 (causal links, neighbors and paths): A link is a *causal link* for some node if the trigger message received over this link contributed to its triggering, i.e., enabled the guard in Line 2 of Algorithm 1. Note that in Algorithm 1 there are always (at least) two causal links. Furthermore, we say that a node is *left triggered* if the link to the left neighbor is causal, *right triggered* is defined analogously. If a nodes is neither left nor right triggered, it is called *bottom triggered*.
A neighbor is a *causal neighbor* if the link from this neighbor is a causal link. According to Assumption 2.1.1, if a causal neighbor (ℓ', i') of a node (ℓ, i) is triggered at time $t_{\ell',i'}$, then the node cannot be triggered before time $t_{\ell',i'} + d^-$, i.e., $t_{\ell,i} \geq t_{\ell',i'} + d^-$.
A *causal path* consists of causal links.

Note that in the theoretical analysis we assume that the processing time of a HEX node is zero, i.e., a link delay must incorporate the edge delay and the processing delay of the node.

2.3.1 The Fault-Free Case

In the fault-free case, we can restrict our attention to the propagation of a single pulse through the network, i.e., one tick of the underlying clock generation system.

Definition 2.3.2 (Left Zig-Zag Path): Informally, a left zig-zag path is a causal path consisting only of links from left and lower-right neighbors.

Given $\ell \in [L+1]$, $\ell > 0$ and $i, i' \in [W]$, $i < i'$, the causal path $p_{\text{left}}^{i' \to (\ell,i)}$ is constructed starting from node (ℓ, i) as follows: Assume the current starting node of $p_{\text{left}}^{i' \to (\ell,i)}$ is (ℓ'', i''). If the left neighbor is causal, then its rightward link $((\ell'', i''-1), (\ell'', i''))$ is prepended to $p_{\text{left}}^{i' \to (\ell,i)}$ and thus $(\ell'', i''-1)$ is the new starting node of the left zig-zag path. Otherwise, the lower-right neighbor must be causal, so its upward-left link $((\ell''-1, i''+1), (\ell'', i''))$ is prepended to $p_{\text{left}}^{i' \to (\ell,i)}$. If $i''+1 = i'$ and $p_{\text{left}}^{i' \to (\ell,i)}$ contains more upward-left links than rightward links, the construction is terminated. If $\ell''-1 = 0$, with arbitrary $i''+1$, the construction is also terminated.

Note that the left zig-zag path depends on the current execution of the grid, as the causal links are determined by the link delays. Furthermore, $p_{\text{left}}^{i' \to (\ell,i)}$ does not necessarily start in column i', i.e., when the construction has been terminated due to $\ell''-1 = 0$, the column may not be i'.

Due to the fact that causal paths are acyclic, $p_{\text{left}}^{i' \to (\ell,i)}$ will descend to a lower layer after at most $|W|-1$ added rightward links. This also shows that causal paths are finite and thus their construction terminates in finite time.

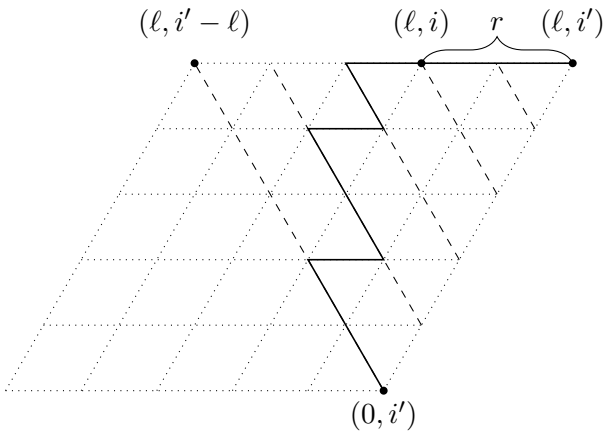


Figure 2.2: Visualization of the "spanning" triangle containing all left zig-zag paths in the proof of Lemma 2.3.5. The dashed lines are the diagonals.

Definition 2.3.3 (Relative Width of a Left Zig-Zag Path): The relative width r of a left zig-zag path is the number of upward-left links in $p_{\text{left}}^{i' \to (\ell,i)}$ minus the number of rightward links.

Lemma 2.3.4 *If a left zig-zag path $p_{\text{left}}^{i' \to (\ell,i)}$ has $r > 0$ and starts in (ℓ', i') then the same holds for every prefix.*

PROOF Suppose that the prefix π violates this, then it must end with a node (ℓ'', i'') with $i'' \geq i'$. But then the suffix must have more up-left than rightward edges and thus must have crossed i'. As a crossing of i' terminates the construction of a left zig-zag path, such a left zig-zag path cannot exist. □

Lemma 2.3.5 *Consider a prefix π of $p_{\text{left}}^{i' \to (\ell'', i'')}$, starting at node (ℓ', i') and ending at (ℓ, i), for which $\ell > 0$ and $r > 0$ holds. Then, the triggering time of node (ℓ, i') satisfies*

$$t_{\ell,i'} \leq t_{\ell,i} + rd^- + (\ell - \ell')\varepsilon.$$

PROOF W.l.o.g. we shift ℓ' to 0 and update ℓ accordingly to $\ell = \ell - \ell'$.

Thus $p_{\text{left}}^{i' \to (\ell,i)}$ is limited by the triangle spanned by the nodes $(0, i')$, $(\ell, i' - \ell)$ and (ℓ, i'), cf. Figure 2.2. Note that (ℓ, i) is between $(\ell, i' - \ell)$ and (ℓ, i') as $i' - \ell \leq i < i'$. Furthermore note that $p_{\text{left}}^{i' \to (\ell,i)}$ will, as shown in Lemma 2.3.4, not cross column i' and thus $r > 0$ also holds for every prefix of π. Also note that in cases where $\ell > W$ the triangle would contain some nodes more than once; nonetheless the left zig-zag paths in such cases are still bounded by this lemma, only the timebound given is an overapproximation.

We define diagonal k as the set of nodes $0 \leq k \leq \ell$, on the line between the nodes (k, i') and $(\ell, i' - \ell + k)$.

To prove the lemma, we will first show that every node p on diagonal k, which is on or to the right of π, will be triggered by time $t_p \leq t_{\ell,i} - (\ell - r)d^- + kd^+$. Note that (ℓ, i) resp. (ℓ, i') are on diagonal $\ell - r$ resp. ℓ.

First we show that the statement holds for every node p on π. The predecessor q of p on π was triggered at least d^- before p, i.e., $t_q \leq t_p - d^-$, as π is a causal path. This argument can be extended to every predecessor of p. So u, the h^{th} predecessor of p, which is h hops away from p, was triggered at least at $t_u \leq t_p - hd^-$. We now relate the diagonal to the hop count: The minimal diagonal reachable is the number of rightward links minus the hop count: $k \geq (\ell - r) - h$. Thus, we can conclude that $t_u \leq t_p - hd^- \leq t_p - (\ell - r)d^- + kd^- \leq t_p - (\ell - r)d^- + kd^+$, and have thereby shown that the statement holds for all nodes on π.

We will now prove by induction that our statement holds on every diagonal. Observe that every node on diagonal 0 is either on π or left of π. Thus, the base case $h = 0$ case has already been shown.

Assume now that statement holds for diagonal k and observe diagonal $k+1$. We only need to consider nodes which are to the right of π. Observe that the left and lower-left neighbor of every node on diagonal $k+1$ are on diagonal k. So every node p on diagonal $k+1$ will be triggered at most d^+ after its neighbors on diagonal k were triggered: $t_p \leq t_{\ell,i} - (\ell - r)d^- + kd^+ + d^+ = t_{\ell,i} - (\ell - r)d^- + (k+1)d^+$, which proves our statement.

To prove our lemma, we just have to plug-in: Since (ℓ, i') is on diagonal ℓ, which is actually $(\ell - \ell')$ due to our initial shifting, we obtain $t_{\ell,i'} \leq t_{\ell,i} - ((\ell - \ell') - r)d^- + (\ell - \ell')d^+ = t_{\ell,i} + rd^- + (\ell - \ell')\varepsilon$ as asserted. □

Definition 2.3.6 (Distance and Skew Potential): For two nodes $i, j \in [W]$, we define their distance as $|i - j|_W = \min(d, W - d)$, where $d = i - j \bmod W$. For $\ell \in [L + 1]$, the skew potential on layer ℓ is $\Delta_\ell = \max\left(\max_{i,j\in[W]}(t_{\ell,i} - t_{\ell,j} - |i - j|_W\, d^-), 0\right)$.

The skew potential is thus the difference between the trigger time of nodes i and j, at the same layer compensated by the time a trigger message would need, in the best case, to propagate between them.

The following Lemma 2.3.7 reveals an important property of HEX, namely, that the skew at layer 0 does not affect the skew at layer $W - 2$ and above.

Lemma 2.3.7 *For all $\ell \in \{W - 2, \ldots, L\}$ we have that $\Delta_\ell \leq 2(W - 2)\varepsilon$.*

PROOF We will derive a bound on Δ_ℓ by subtraction the earliest from the latest triggering time of any node on layer ℓ.

W.l.o.g. we assume that $\ell = W - 2$. Further, we pick two nodes $i, i' \in [W]$, $i < i'$, for which $i' - i = |i - i'|_W$ holds. Cases where the wrap-around distance is shorter can be handled by swapping the nodes, which is symmetrical anyway. We distinguish two cases.

Case 1: $p_{\text{left}}^{i' \to (\ell,i)}$ starts at node (ℓ', i') for some $\ell' \in \{1, \ldots, \ell - 1\}$. Then, by Lemma 2.3.5,

$$t_{\ell,i'} \leq t_{\ell,i} + (i' - i)d^- + (\ell - \ell')\varepsilon \leq t_{\ell,i} + (i' - i)d^- + \ell\varepsilon \leq t_{\ell,i} + (i' - i)d^- + (W - 2)\varepsilon.$$

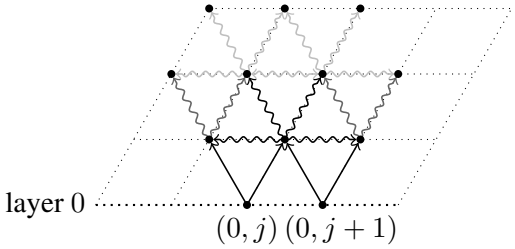


Figure 2.3: Illustration of the induction proof Lemma 2.3.7 at time d^+, considering all link delays are d^+. The solid lines mark the trigger messages which where received at d^+, the wiggly black lines symbolize the trigger messages which will be sent at d^+. The dark gray wiggly lines are the trigger messages which will be sent at $2d^+$, and the light gray are those sent at $3d^+$.

Case 2: $p_{\text{left}}^{i' \to (\ell,i)}$ starts at node $(0, j)$, $j \in [W]$. The path from $(0, j)$ to (ℓ, i) contains ℓ upward-left links and at least $(\ell - (i' - i))$ rightward links.

Denote by t_0 the first time two adjacent nodes on layer 0 trigger. Then the earliest time node (ℓ, i) will be triggered is:

$$t_{\ell,i} \geq t_0 + \ell d^- + (\ell - (i' - i))d^- = t_0 + (2(W-2) - (i' - i))d^- \tag{2.1}$$

We will now show that every node in layer $\ell = W - 2$ will be triggered at latest by $t_0 + 2(W-2)d^+$. Therefore, assume that $(0, j)$ is the node such that $\max(t_{0,j}, t_{0,j+1}) = t_0$. We will prove by induction on λ that, for every layer λ, the nodes $(\lambda, j-\lambda), (\lambda, j-\lambda+1), \ldots, (\lambda, j+1)$ will be triggered no later than $t_0 + 2\lambda d^+$. An illustration of the first few layers is shown in Figure 2.3.

For the base case $\lambda = 0$ consider layer 0: As the nodes $(0, j)$ and $(0, j+1)$ were used to define t_0, the hypothesis holds.

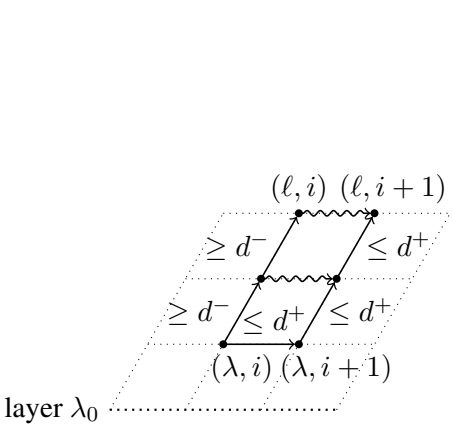
Assume that the hypothesis held in layer λ, then the nodes $(\lambda, j-\lambda), (\lambda, j-\lambda+1), \ldots, (\lambda, j+1)$ triggered at latest by $t_0 + 2\lambda d^+$. The nodes $(\lambda+1, j-\lambda), (\lambda, j-\lambda+1), \ldots, (\lambda, j)$ all have two neighbors in layer λ which triggered no later than $t_0 + 2\lambda d^+$, thus those nodes will trigger no later than $t_0 + (2\lambda+1)d^+$. Hence, no later than $t_0 + (2\lambda+2)d^+ = t_0 + 2(\lambda+1)d^+$ the nodes $(\lambda+1, j-\lambda+1)$ and $(\lambda+1, j+1)$ will also trigger and thereby establish the hypothesis for $\lambda + 1$.

We can now use Equation (2.1) to calculate the worst-case skew between the nodes i and i':

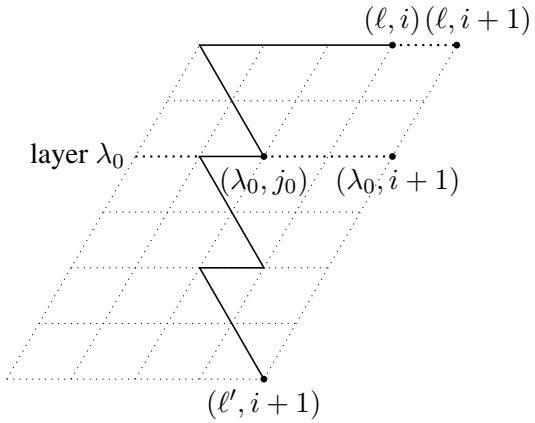
$$\begin{aligned} t_{\ell,i} &\geq t_0 + (2(W-2) - (i' - i))d^- \\ t_{\ell,i'} &\leq t_0 + 2(W-2)d^+ \\ t_{\ell,i'} - t_{\ell,i} &\leq t_0 + 2(W-2)d^+ - t_0 - (2(W-2) - (i' - i))d^- \\ &\leq 2(W-2)\varepsilon + (i' - i)d^- \end{aligned}$$

As all cases of $(i' - i)$ can be covered by symmetry, as stated in the beginning, we can replace the term with the node distance, and conclude that

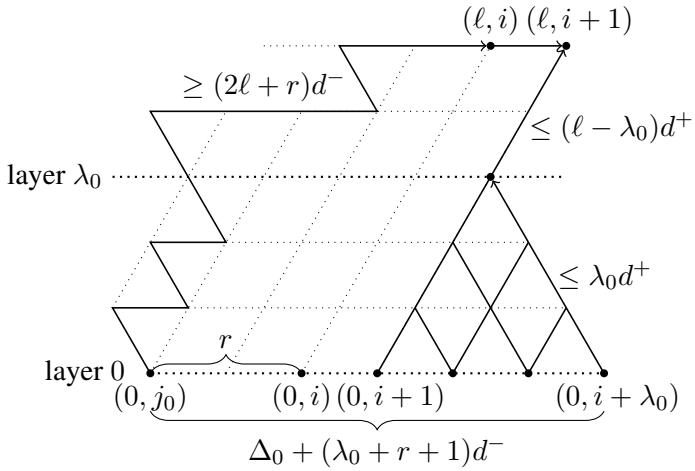
$$\Delta_\ell = \max_{i,i' \in [W]} (t_{\ell,i'} - t_{\ell,i} - |i' - i|_W \, d^-) \leq 2(W-2)\varepsilon$$



(a) Illustration of Case 1 of Lemma 2.3.8.



(b) Illustration of Case 3 of Lemma 2.3.8.



(c) Illustration of Case 2 of Lemma 2.3.8.

Figure 2.4: Illustration of the three cases used to proof Lemma 2.3.8.

holds as claimed in both cases. □

Lemma 2.3.8 *For all layers $\ell_0 \in [L]$ and $\ell \in \{\ell_0 + 1, \ldots, L\}$, it holds for each $i \in [W]$ that*

$$|t_{\ell,i} - t_{\ell,i+1}| \leq d^+ + \left\lfloor \frac{(\ell - \ell_0)\varepsilon}{d^+} \right\rfloor \varepsilon + \Delta_{\ell_0}.$$

PROOF We define $\lambda_0 = \left\lfloor \frac{\ell d^-}{d^+} \right\rfloor$, which marks the last layer where the longest possible $p_{\text{left}}^{i' \to (\ell,i)}$, under Assumption 2.1.1 with delay d^-, from a node $(0, i)$ is slower than the direct path from

$(0,i)$ to (λ_0,i) with delay d^+. Furthermore it holds that

$$\ell - \left\lfloor \frac{\ell d^-}{d^+} \right\rfloor = \left\lceil \frac{\ell \varepsilon}{d^+} \right\rceil . \tag{2.2}$$

Case 1: $t_{\lambda,i+1} \leq t_{\lambda,i} + d^+$ for the smallest $\lambda > \lambda_0$ such that for all $\lambda' \in \{\lambda+1,\dots,\ell\}$: $t_{\lambda',i+1} > t_{\lambda',i} + d^+$ holds true, cf. Figure 2.4a. This implies that the nodes (λ',i) cannot be triggered by their right neighbor, and therefore the links from $(\lambda'-1,i)$ must be causal. From this it follows that $t_{\lambda'-1,i} + d^- \leq t_{\lambda',i}$. As this property holds for all nodes, by induction, we get $t_{\ell,i} \geq t_{\lambda,i} + (\ell-\lambda)d^-$.

By the required property on λ we can also be sure that the trigger message from (λ',i) will arrive at node $(\lambda',i+1)$ before $t_{\lambda',i+1}$. Hence, the node will trigger when the message from its lower-left neighbor $(\lambda'-1,i+1)$ arrives, if the node has not already done so. Consequently, d^+ is the upper bound for the skew between two nodes in column $i+1$ starting from layer λ, and we can, by induction, conclude that

$$t_{\ell,i+1} \leq t_{\lambda,i+1} + (\ell-\lambda)d^+. \tag{2.3}$$

As we know, by our assumption, the difference between the two nodes (λ,i) and $(\lambda,i+1)$, we can extend the former inequality to $t_{\ell,i+1} \leq t_{\lambda,i} + (\ell-\lambda)d^+ + d^+$. The skew between the two neighbors (ℓ,i) and $(\ell,i+1)$ on layer ℓ hence follows from

$$\begin{aligned}
-t_{\ell,i} &\leq -t_{\lambda,i} - (\ell-\lambda)d^- \\
t_{\ell,i+1} - t_{\ell,i} &\leq t_{\lambda,i} + (\ell-\lambda)d^+ + d^+ - t_{\lambda,i} - (\ell-\lambda)d^- = (\ell-\lambda)\varepsilon + d^+ \\
t_{\lambda',i+1} > t_{\lambda',i} + d^+ &\Rightarrow t_{\ell,i+1} - t_{\ell,i} > 0 \\
t_{\ell,i+1} - t_{\ell,i} &= |t_{\ell,i} - t_{\ell,i+1}| \leq (\ell-\lambda)\varepsilon + d^+
\end{aligned} \tag{2.4}$$

Case 2: Case 1 does not apply and $p_{\text{left}}^{i'\to(\ell,i)}$ starts at some node $(0,j_0)$, for $j_0 \neq i+1$. We can also conclude that $r \geq 0$ (recall Definition 2.3.3), as otherwise the end would be in node $(0,i+1)$, which is excluded, or the left zig-zag path would have crossed column $i+1$, in which case $r > 0$ would have been true and thus the left zig-zag path would have terminated there. Hence, $p_{\text{left}}^{i'\to(\ell,i)}$ has length $2\ell + r$ and $j_0 = i - r \bmod W$ and thereby we can infer that $t_{\ell,i} \geq t_{0,j_0} + (2\ell+r)d^-$.

Observe that a d^+ path from node $(0,i+1)$ to $(\lambda_0,i+1)$ spans a triangle with node $(0,i+\lambda_0+1)$ as the third corner, as can be seen in Figure 2.4c. For each $j \in \{i+1,i+2,\dots,i+\lambda_0+1\}$ it holds that $|j-j_0|_W \leq j - j_0 = j - i + r$. From that, and using the definition of the skew potential, we obtain

$$\begin{aligned}
t_{\ell,i} &\geq t_{0,j_0} + (2\ell+r)d^- \\
&\geq t_{0,j} - |j-j_0|_W d^- - \Delta_0 + (2\ell+r)d^- \\
&\geq t_{0,j} - (j-i)d^- - \Delta_0 + 2\ell d^- \\
&\geq t_{0,j} - \Delta_0 + (2\ell - \lambda_0 - 1)d^-.
\end{aligned}$$

By induction, we can show that for each $\lambda \in \{0, \dots, \lambda_0\}$ every node $(\lambda, i+1), (\lambda, i+2), \dots, (\lambda, i+\lambda_0+1-\lambda)$ will trigger no later than

$$\max_{j \in \{i+1, \dots, i+\lambda_0+1-\lambda\}} (t_{0,j}) + \lambda d^+ \leq t_{\ell,i} + \Delta_0 - (2\ell - \lambda_0 - 1)d^- + \lambda d^+.$$

By using the definition of λ_0, as the maximal value for λ, we get

$$\begin{aligned} t_{\lambda_0,i+1} &\leq t_{\ell,i} + \Delta_0 - (2\ell - \lambda_0 - 1)d^- + \lambda d^+ \\ &\leq t_{\ell,i} + \Delta_0 - (2\ell - \lambda_0 - 1)d^- + \frac{\ell d^-}{d^+} d^+ = t_{\ell,i} + \Delta_0 - (\ell - \lambda_0 - 1)d^-. \end{aligned}$$

Case 1 does not apply, thus $t_{\lambda,i+1} > t_{\lambda,i} + d^+$ holds for all $\lambda \geq \lambda_0$, which allows us to use similar arguments as we used to derive Equation (2.3): Every node $(\lambda, i+1)$ is left triggered and thus its triggering time can be obtained, by induction, starting from node $(\lambda_0, i+1)$, which leads to

$$t_{\ell,i+1} \leq t_{\lambda_0,i+1} + (\ell - \lambda_0)d^+. \tag{2.5}$$

Therefore, it follows that

$$\begin{aligned} t_{\ell,i+1} - (\ell - \lambda_0)d^+ \leq t_{\lambda_0,i+1} &\leq t_{\ell,i} + \Delta_0 - (\ell - \lambda_0 - 1)d^- \\ t_{\ell,i+1} &\leq t_{\ell,i} + \Delta_0 - (\ell - \lambda_0 - 1)d^- + (\ell - \lambda_0)d^+ \\ &= t_{\ell,i} + \Delta_0 + (\ell - \lambda_0)\varepsilon + d^- = t_{\ell,i} + \Delta_0 + \left(\ell - \left\lfloor \frac{\ell d^-}{d^+} \right\rfloor\right)\varepsilon + d^- \\ &= t_{\ell,i} + \Delta_0 + \left\lceil \frac{\ell \varepsilon}{d^+} \right\rceil \varepsilon + d^-, \end{aligned}$$

from which we arrive at

$$t_{\ell,i+1} - t_{\ell,i} = |t_{\ell,i} - t_{\ell,i+1}| \leq \Delta_0 + \left\lceil \frac{\ell \varepsilon}{d^+} \right\rceil \varepsilon + d^-.$$

Case 3: Neither Case 1 nor Case 2 apply. In this case, $p_{\text{left}}^{i' \to (\ell,i)}$ starts at some node $(\ell', i+1)$ for some $\ell' < \ell$, and $t_{\lambda,i+1} > t_{\lambda,i} + d^+$ holds for all $\lambda \geq \lambda_0$. Therefore, the first link of $p_{\text{left}}^{i' \to (\ell,i)}$ is $((\ell', i+1), (\ell'+1, i))$, as shown in Figure 2.4b. As Case 1 does not apply, $p_{\text{left}}^{i' \to (\ell,i)}$ must start on a layer $\ell' < \lambda_0 - 1$: Otherwise, e.g., for $\ell' = \lambda_0 - 1$, $(\lambda_0, i+1)$ would be triggered no later than $\max(t_{\ell',i+1} + d^+, t_{\lambda_0,i} + d^+)$. Hence $t_{\lambda_0,i+1} \leq t_{\lambda_0,i} + d^+$ would hold, which contradicts the assumption that Case 1 does not hold.

Let (λ_0, j_0) be the last node of $p_{\text{left}}^{i' \to (\ell,i)}$ on layer λ_0. Observe that $j_0 = i + 1 - r$ holds and thus Lemma 2.3.5 can be applied on a prefix π of $p_{\text{left}}^{i' \to (\ell,i)}$ ending at node (λ_0, j_0), thus yielding $t_{\lambda_0,i+1} \leq t_{\lambda_0,j_0} + rd^- + (\lambda_0 - \ell')\varepsilon = t_{\lambda_0,j_0} + (i+1-j_0)d^- + (\lambda_0 - \ell')\varepsilon$. As Case 1 does not hold, we can use Equation (2.5) and obtain

$$t_{\ell,i+1} \leq t_{\lambda_0,j_0} + (i+1-j_0)d^- + (\lambda_0 - \ell')\varepsilon + (\ell - \lambda_0)d^+.$$

By construction, $p_{\text{left}}^{i' \to (\ell,i)}$ has length $2(\ell - \ell') - 1$ and the prefix π has length $2(\lambda_0 - \ell') - (i+1-j_0)$, thus the suffix starting at node (λ_0, j_0) has length $2(\ell - \lambda_0) + (i - j_0)$. As the suffix is also a causal path, we can infer that

$$t_{\ell,i} \geq t_{\lambda_0,j_0} + (2(\ell - \lambda_0) + (i - j_0))\, d^-.$$

Thus, we arrive at

$$\begin{aligned}
t_{\ell,i+1} - t_{\ell,i} &\leq (1 + i - j_0)d^- + (\lambda_0 - \ell')\varepsilon + (\ell - \lambda_0)d^+ - (2(\ell - \lambda_0) + (i - j_0))\, d^- \\
&= d^- + (\lambda_0 - \ell')\varepsilon + (\ell - \lambda_0)d^+ - 2(\ell - \lambda_0)d^- \\
&= d^- + (\ell - \ell')\varepsilon - (\ell - \lambda_0)d^- = (\ell - \ell')\varepsilon - (\ell - \lambda_0 - 1)d^- \\
&\underbrace{\leq}_{\ell'=0} \ell\varepsilon - \left(\ell - \frac{\ell d^-}{d^+} - 1\right) d^- = \ell\varepsilon - \left(\frac{\ell\varepsilon}{d^+} - 1\right) d^- = \ell\varepsilon - \frac{\ell\varepsilon d^-}{d^+} + d^- \\
&\leq \left\lceil \frac{\ell\varepsilon}{d^+} \right\rceil \varepsilon + d^-.
\end{aligned}$$

Since these three cases where exhaustive and for each the claimed bound holds, this concludes the proof. □

Corollary 2.3.9 *Set $\delta = d^-/2 - \varepsilon$. For each layer $\ell \in \{W, \dots, L\}$ and $i \in [W]$ it holds that*

$$|t_{\ell,i} - t_{\ell,i+1}| \leq \max\left\{ d^+ + \left\lceil \frac{W\varepsilon}{d^+} \right\rceil \varepsilon,\ \Delta_{\ell - W} + d^+ - W\delta \right\}. \tag{2.6}$$

PROOF The proof is, for great parts, analogous to Lemma 2.3.8 and thus will follow the same patterns. W.l.o.g. we assume that $\ell_0 = \ell - W$.

Case 1: $t_{\lambda,i+1} \leq t_{\lambda,i} + d^+$ for the smallest $\lambda > \lambda_0$ such that for all $\lambda' \in \{\lambda + 1, \dots, \ell\}$: $t_{\lambda',i+1} > t_{\lambda',i} + d^+$ holds true. As there are no assumptions on the layer in Lemma 2.3.8, we can directly take Equation (2.4) from said lemma:

$$|t_{\ell,i} - t_{\ell,i+1}| \leq (\ell - \lambda)\varepsilon + d^+ \leq \left\lceil \frac{(\ell - \ell_0)\varepsilon}{d^+} \right\rceil \varepsilon + d^+.$$

Due to the assumption about ℓ_0, the equation simplifies to

$$|t_{\ell,i} - t_{\ell,i+1}| \leq d^+ + \left\lceil \frac{W\varepsilon}{d^+} \right\rceil \varepsilon,$$

which proves this case.

Case 2: Case 1 does not apply and $p_{\text{left}}^{i' \to (\ell,i)}$ starts at some node $(0, j_0)$, for $j_0 \neq i+1$. This case needs some additional refinements to show the bound. The beginning is equal to Case 2 of Lemma 2.3.8, but w.l.o.g. we now assume that $\ell = W$.

For each $j \in \{i+1, i+2, \dots, i+\lambda_0+1\}$ it holds that $|j - j_0|_W \leq j - j_0 \leq W/2 = \ell/2$, which is different from what has been used in Lemma 2.3.8. From that, and using the definition of the skew potential, we obtain

$$\begin{aligned} t_{\ell,i} &\geq t_{0,j_0} + (2\ell + r)d^- \\ &\geq t_{0,j} - |j - j_0|_W d^- - \Delta_0 + (2\ell + \underbrace{r}_{\geq 0})d^- \\ &\geq t_{0,j} - \frac{\ell d^-}{2} - \Delta_0 + 2\ell d^- = t_{0,j} - \Delta_0 + \frac{3\ell d^-}{2}. \end{aligned}$$

The proof now proceeds as in Case 2 of Lemma 2.3.8. By induction, we can show that

$$t_{\lambda_0, i+1} \leq t_{\ell,i} + \Delta_0 - \frac{\ell d^-}{2}.$$

Plugging this into Equation (2.5) finally leads to

$$\begin{aligned} t_{\ell,i+1} &\leq t_{\ell,i} + \Delta_0 - \frac{\ell d^-}{2} + (\ell - \lambda_0)d^+ = t_{\ell,i} + \Delta_0 - \frac{\ell d^-}{2} + \left\lceil \frac{\ell\varepsilon}{d^+} \right\rceil d^+ \\ &\leq t_{\ell,i} + \Delta_0 - \frac{\ell d^-}{2} + \ell\varepsilon + d^+ = t_{\ell,i} + \Delta_0 - \ell\left(\frac{d^-}{2} - \varepsilon\right) + d^+, \end{aligned}$$

from which we can conclude that

$$|t_{\ell,i+1} - t_{\ell,i}| \leq \Delta_0 - \ell\left(\frac{d^-}{2} - \varepsilon\right) + d^+.$$

Using the assumption that $\ell = W$, and generalizing for arbitrary ℓ_0, we arrive at

$$|t_{\ell,i+1} - t_{\ell,i}| \leq \Delta_{\ell - W} + d^+ - W\delta.$$

Case 3: Neither Case 1 nor Case 2 apply. Similar to Case 1, we only have to consider the changed bound for $\ell' = \ell - W$

$$\begin{aligned} t_{\ell,i+1} - t_{\ell,i} &\leq (\ell - \ell')\varepsilon - (\ell - \lambda_0 - 1)d^- \\ &\underbrace{\leq}_{\ell' = \ell - W} W\varepsilon - \left(\frac{W\varepsilon}{d^+} - 1\right)d^- = \frac{W\varepsilon^2}{d^+} + d^- \\ &\leq d^+ + \left\lceil \frac{W\varepsilon}{d^+} \right\rceil \varepsilon. \end{aligned}$$

As for all three cases the asserted bound holds, this concludes the proof. □

Theorem 2.3.10 (Skew Bounds — Fault-Free Case) *Suppose that $\varepsilon \leq d^+/7$. Then the following upper bounds hold on the intra-layer skew $\sigma_\ell = \max_{i\in[W]}\{|t_{\ell,i} - t_{\ell,i+1}|\}$ in layer ℓ: If $\Delta_0 = 0$, then σ_ℓ is uniformly bounded by $d^+ + \lceil W\varepsilon/d^+\rceil\,\varepsilon$ for any $\ell \in [L+1]$. In the general case,*

$$\forall \ell \in \{1,\dots,2W-3\}\colon \sigma_\ell \leq d^+ + 2W\varepsilon^2/d^+ + \Delta_0,$$
$$\forall \ell \in \{2W-2,\dots,L\}\colon \sigma_\ell \leq d^+ + \lceil W\varepsilon/d^+\rceil\varepsilon.$$

Moreover, regarding the inter-layer skew of layer $\ell \in [L]$ to its neighboring layer $\ell+1$, it holds for all $i \in [W]$ that

$$t_{\ell,i} - \sigma_\ell + d^- \leq t_{\ell+1,i} \leq t_{\ell,i} + \sigma_\ell + d^+ \text{ and}$$
$$t_{\ell,i+1} - \sigma_\ell + d^- \leq t_{\ell+1,i} \leq t_{\ell,i+1} + \sigma_\ell + d^+.$$

PROOF For sake of the argument, assume a HEX grid with $\Delta_0 = 0$ and link delays, up to layer $W-1$, of d^+, which results in $\Delta_\ell = 0$ for all $\ell \in \{0,1,\dots,W-1\}$.

By applying Lemma 2.3.7, we get that $\Delta_\ell \leq 2(W-2)\varepsilon$ holds for all $\ell \in \{W-2,\dots,L\}$. We now apply Corollary 2.3.9 to derive an intra-layer bound for the layers from W on. For all layers, including $2W-1$, the second term of the max term in Equation (2.6) reduces to $d^+ - W(d^-/2 - \varepsilon)$, as $\Delta_\ell = 0$ up to layer $W-1$. Thus, for those layers, the first term will determine the maximum. For the layers from $2W-1$ onwards, we also consider the following part of the second term in Equation (2.6):

$$\begin{aligned}\Delta_{\ell-W} - W\delta &\leq 2(W-2)\varepsilon - W\delta = W(2\varepsilon-\delta) - 4\varepsilon = W(3\varepsilon - d^-/2) - 4\varepsilon \\ &\leq W(3d^+/7 - \underbrace{d^-/2}_{\frac{6}{7}d^+ \leq d^-}) - 4\varepsilon\end{aligned}$$

As we assumed that $\varepsilon \leq d^+/7$ holds, this term is also negative. Thus, for all layers greater than W, the first term also dominates the max term.

Now consider a grid with $\Delta_0 = 0$ and $L - W$ layers. This grid can be seen as the part of the above grid from layer W onwards. Thus we can conclude that, for $\Delta_0 = 0$, the intra-layer skew is bounded by $d^+ + \lceil W\varepsilon/d^+\rceil\,\varepsilon$ for any $\ell \in [L+1]$.

We will now consider skew bounds for arbitrary Δ_0. For the layers up to $2W-2$, we can directly apply Lemma 2.3.8, which leads to an upper bound of $d^+ + \left\lfloor \frac{W\varepsilon}{d^+}\right\rfloor \varepsilon + \Delta_0$. For the remaining layers of the grid, we can use Corollary 2.3.9 in combination with Lemma 2.3.7. The lemma provides us with a skew potential of $\Delta_\ell \leq 2(W-2)\varepsilon$ for $\ell \in \{W-2,\dots,L\}$ that, in combination with the corollary, results in the same bound as established above for $\Delta_0 = 0$.

The inequalities regarding the inter-layer skew are proven by using the inter-layer skew bound and the link delay bounds:

$$t_{\ell,i} - \sigma_\ell + d^- \leq \min(t_{\ell,i}, t_{\ell,i+1}) + d^- \leq t_{\ell+1,i} \leq \max(t_{\ell,i}, t_{\ell,i+1}) + d^+ \leq t_{\ell,i} + \sigma_\ell + d^+.$$

The second inequality is proven analogously, which concludes the proof. □

2.4 Fault Models

As usual network properties are not provided by HEX, like (almost) fully connectivity or bidirectional communication, we cannot hope to match classic resilience results, like tolerance to f Byzantine node failures provided $n \geq 3f + 1$ [14, 52].

A promising alternative are f–local failure models [7, 34, 53], which assume that every node has at most f faulty nodes among its n direct neighbors. It is particularly suitable for regular topologies like HEX, and has the advantage that the network-wide number of faulty nodes scales with the network size.

To define suitable failure semantics, we first have to consider the possible effect of faults on the HEX grid. As elaborated in Section 1.1.3, we have two basic fault types:

Fail-Silent A fail-silent or crash faulty node can only stop sending trigger messages forever. It thus can only slow-down the firing of a neighbor, if the link to that respective neighbor would have been causal.

Byzantine As a Byzantine faulty node actually has no limitations on its behavior, the current implementation, presented in Chapter 3, would have a problem: The receiver component (cf. Section 3.1.1) is prone to metastability if the trigger message is too short. Thus, we must restrict Byzantine faults from inducing metastability.

It follows that a Byzantine fault is limited to

- moderately speed-up its neighboring nodes, as two causal links are necessary for a node to trigger.
- arbitrarily slow-down its neighboring nodes. However, compared to the fail-silent fault, the trigger message could be arbitrarily delayed for pulse k, which may in turn result in an arbitrary speed-up for pulse $k + 1$!

As Algorithm 1 is designed to withstand one Byzantine faulty neighbor, this property can be used as a first base of our fault model:

Assumption 2.4.1: Every node in the HEX grid has at most one faulty neighbor.

Under this restriction, a faulty nodes requires that the 18 nodes with a distance of ≤ 2 hops are non-faulty, cf. Figure 2.5. This number may be smaller for narrow grids or nodes near the top or the bottom. Requiring such a large neighborhood of correct nodes is rather restrictive, as the birthday paradox[3] [1] causes the probability of a "correct" random placement of faulty nodes to drop fast with the number of faulty nodes.

It is easy to obtain a lower bound for the probability that a correct placement can be achieved: The following formula provides a lower bound for the probability of a correct placement of

[3] In [1] a solution is presented for the question who likely it is that f people have their birthdays at least $k/2$ days apart. As these $k/2$ days span in both directions, they represent the size of the neighborhood of the fault in which no other fault must be placed. Unfortunately, this 1D problem does not fully match the 2D situation in our grid: While the mapping, by itself, from these $k/2$ day interval from the 1D timeline to the 2D grid would not produce problems, the 1D solution allows no overlapping of the neighborhoods of two persons which is possible in our case.

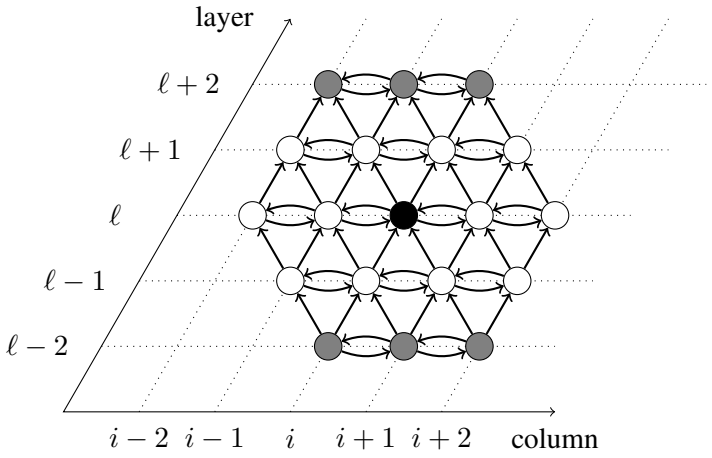
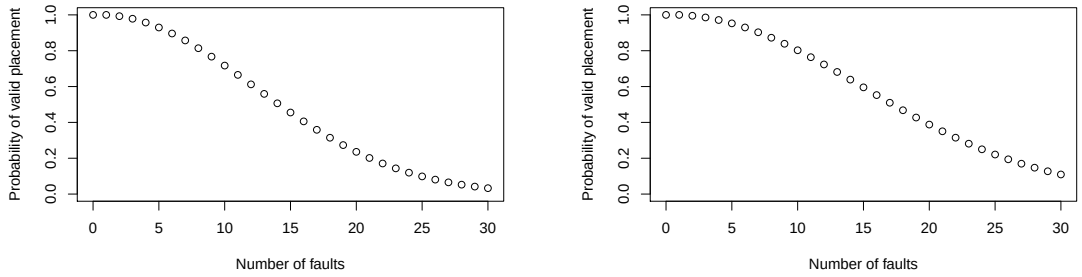


Figure 2.5: When the node (ℓ, i) is faulty, then if one of the white nodes would also be faulty, one node would have two faulty nodes on incoming edges. If one of the gray nodes would be faulty, then some nodes would have two faulty neighboring nodes.



(a) 18 correct neighbors per fault required

(b) 12 correct neighbors per fault required

Figure 2.6: Probability of correct placement of faults under the 1–local fault model according to Equation (2.7). The functions were plotted with 2500 nodes and such that 12 resp. 18 direct neighbors of a faulty node must be correct.

f faulty nodes, which require k surrounding neighboring nodes to be non-faulty, in a grid with n nodes:

$$p(k, f, n) \geq \frac{n \times (n-(k+1)) \times \ldots \times (n-(f-1)(k+1))}{f!} \binom{n}{f}^{-1} \tag{2.7}$$
$$\geq \prod_{i=0}^{f-1} \frac{1-(k+1)i}{n} \geq \left(\frac{1-(k+1)f}{n}\right)^{f}.$$

Equation (2.7) is similar to a (n, f)-combination, but instead of removing only one element per selection from the set of possible elements, $k+1$ elements, the entire neighborhood of the faulty node and the faulty node itself, are removed. To get the probability, the term is divided by the $\binom{n}{k}$ possible combinations of placing f faulty nodes in n nodes.

Figure 2.6a shows a plot of Equation (2.7) with 2500 nodes, which is the grid size used in our simulations in Chapter 3. The probability for a successful placement of 15 faults is at least $\approx 46\%$, which is pretty low given that only 0.6% of the nodes are faulty.

Fortunately, Assumption 2.4.1 can be weakened. As the HEX grid is directed, the fault model only has to ensure that no node has two faulty nodes on its incoming links.

Definition 2.4.2 (1–local fault model): Every node in the HEX grid has at most one faulty neighbor on its incoming links.

This weakened model reduces the size of the fault-free neighborhood to 12 nodes: Every in-neighbor of the 4 nodes who have a faulty node as in-neighbor must be non-faulty. With this model, the probability of a correct placement of 15 faults has increased to at least $\approx 60\%$, for $f = 20$, this probability has increased from at least $\approx 24\%$ to at least $\approx 39\%$.

Clustered Faults In general, a HEX node can only tolerate one faulty in-neighbor. Nonetheless, HEX allows some cases where one node has multiple faulty neighbors, at the cost of an increase of the skew. Consider the case where the two intra-layer neighbors of (ℓ, i) are faulty. In this case, node (ℓ, i) would not be affected by the faults, due to the firing rules. Unfortunately, however, if both faulty nodes are fail-silent, then the nodes $(\ell + 1, i - 1)$ and $(\ell + 1, i)$ could not fire, i.e., become mute.

A more extreme scenario would assume that the two adjacent in-neighbors of (ℓ, i) in the layer below, i.e., $(\ell - 1, i)$ and $(\ell - 1, i + 1)$, are faulty. In the case of Byzantine faults, they could force (ℓ, i) to produce spurious pulses, resulting in a spurious wave similar to the scenario shown in Figure 2.3. In the case of fail-silent nodes, the skew of $(\ell, i - 1)$ and $(\ell, i + 1)$ would possibly increase and (ℓ, i) would become mute.

In the case of three neighboring fail-silent nodes, e.g. $(\ell-1, i-1)$, $(\ell-1, i)$ and $(\ell-1, i+1)$, the nodes $(\ell, i - 1)$ and (ℓ, i) would also become mute. Yet, due to the structure of the HEX grid, this "muteness wave" will not expand, but rather vanish in the form of a triangle with increasing layers. Note that this even generalizes to more than three fail-silent neighbors in the same layer.

If the fail-silent nodes would be in adjacent layers and in the same column, then there is no way to cause other nodes to become mute. But a chain of fail-silent faults along a column would slice part of the cylindrical HEX grid open, which would, again, have a bad effect on the skews.

2.5 Pulse Separation

Pulse separation is needed to restrain a node from firing pulse k (partially) based on stale trigger messages for pulse $k-1$, which may have arrived late. There are generally two methods to prevent such behavior, either (i) counting or labeling the messages carrying the pulses or (ii) temporal separation of their occurrence such that it is not possible for the nodes to mix them up. HEX uses the latter approach, which leads us to the question of how to determine the timeout interval $[T^-, T^+]$ (cf. Algorithm 1), given some pulse separation time S such that $S + t_+^{(k)} \leq t_-^{(k+1)}$. Herein $t_-^{(k)}$ denotes the time when the first node in layer 0 generates pulse k, and $t_+^{(k)}$ gives the time when the last node in layer 0 does so. The basic idea is to sleep so long that every

outstanding pulse k trigger message from the neighboring nodes has been received, but not so long that a message corresponding to pulse $k+1$ is missed.

First, consider the following general upper bound on the pulse separation time at layer ℓ, which must hold in any case to ensure this:

$$t_{\ell,i}^{(k)} + T^+ < \min\left\{t_{\ell,i-1}^{(k+1)}, t_{\ell,i+1}^{(k+1)}, t_{\ell-1,i}^{(k+1)}, t_{\ell-1,i+1}^{(k+1)}\right\} + d^-$$

Consider now the lower timeout interval bound T^-. As established before, this value must be large enough to prevent the node from firing due to old pulses arriving late.

Lemma 2.5.1 *Suppose that the skew potential for any pulse k satisfies $\max_{k\in\mathbb{N}}\left\{\Delta_0^{(k)}\right\} \leq \Delta$ and denote $\sigma_0 = \Delta + d^-$. Then, a feasible lower bound for the timeout interval, which ensures that every node fires at most once per pulse k, is*

$$T^- > \sigma_\ell + d^+ + \varepsilon, \tag{2.8}$$

for all $\ell \in [L+1]$, where σ_ℓ is the intra-layer skew as in Theorem 2.3.10 for $\Delta_0 = \Delta$.

PROOF W.l.o.g. consider the node (ℓ, i). T^- must be chosen so that, after triggering pulse k, no other triggering message from a correct node based on pulse k can trigger (ℓ, i) again. We will thus consider the three firing rules defined in Algorithm 1.

Rule 1: first trigger by the lower neighbors. Assume that at time t_0 the last pulse from the lower neighbors of node (ℓ, i) was sent. This pulse will be received by (ℓ, i) not before $t_0 + d^-$, i.e., by $t_0 + d^-$ or later node (ℓ, i) triggers for the first time.

At latest by $t_0 + d^+$ the left and right neighbor of (ℓ, i) will also receive the pulse, and trigger no later than $t_0 + d^+ + \sigma_{\ell-1}$ due to the trigger message from their other lower neighbor. The pulses from the left and right neighbor take at most d^+ until they are received by (ℓ, i). Since (ℓ, i) has triggered for the first time not before $t_0 + d^-$, it would be triggered for a second time no later than $d^+ + \sigma_{\ell-1} + \varepsilon$ after the first triggering of node (ℓ, i). As the proposed bound is required to hold for all $\ell \in [L+1]$, the bound holds in this case.

Rule 2: first trigger by the left and lower-left neighbor. In this rule, we have to consider the time required until rule 1 or rule 3 would trigger node (ℓ, i) again. Rule 3 would require a pulse from the right neighbor, which must occur not later than σ_ℓ after the firing of node (ℓ, i) and will thus, with the proposed bound, be ignored. Rule 1 would require an additional message from the lower-left neighbor which could happen if this neighbor is Byzantine faulty, which is inside our fault hypothesis. Nonetheless, no node can be triggered by one faulty neighbor alone so that an additional triggering by rule 1 cannot happen in the fault-free case.

Rule 3: first trigger by the right and lower-right neighbor. This rule is analog to rule 2, which concludes the proof. □

Due to the nature of the timeout-interval, the upper bound T^+ must just be larger than T^-, or to be more precise, larger than the actual timeout instant of the timer. As we will see, when T^+ is known, the pulse separation time can be calculated.

Lemma 2.5.2 *Assume that the conditions of Lemma 2.5.1 hold and that the pulse separation time satisfies*

$$S > L\varepsilon + Wd^+ + T^+.$$

If the clock generation algorithm employed in layer 0 is self-stabilizing and no more faults are present in the grid, then HEX will self-stabilize within L pulses after layer 0 stabilized. The stabilization is meant in the sense that each node triggers exactly once per pulse and the skew bounds from Theorem 2.3.10 hold.

PROOF Assume for simplicity that pulse 0 is the first stable pulse, after all faults disappeared and a valid pulse separation time has passed, generated by the clock generation system at layer 0. The proof is by induction, using the induction hypothesis that every layer ℓ fires correctly at pulse $k \geq \ell$. Thus by the above assumption the hypothesis holds for pulse 0 and layer 0. For the induction step from ℓ to $\ell + 1$ assume that layer ℓ has stabilized, i.e., pulse $k = \ell$ has fired. The time in which the first and the last nodes in layer ℓ fired pulse k is bounded by the interval $[t_-^{(k)} + \ell d^-, t_+^{(k)} + \ell d^+]$. Hence, no node in layer $\ell + 1$ can be triggered anew, for pulse $k + 1$, by its lower neighbors alone within $[t_+^{(k)} + (\ell + 1)d^+, t_-^{(k+1)} + (\ell + 1)d^-]$.

However, layer $\ell + 1$ has not stabilized yet, so nodes in that layer may have missed trigger messages for pulse k from their lower neighbors. In the worst-case, one node was trigger by its lower neighbors, not later than $t_+^{(k)} + (\ell + 1)d^+$, and the resulting trigger messages propagate through the W columns of layer $\ell+1$ in a chain, taking at most Wd^+ time. Therefore, no node in layer $\ell+1$ will trigger anew within the interval $[t_+^{(k)} + (\ell+1)d^+ + Wd^+, t_-^{(k+1)} + (\ell+1)d^-]$, and hence no node in layer $\ell+1$ will be sleeping in $[t_+^{(k)} + (\ell+1)d^+ + Wd^+ + T^+, t_-^{(k+1)} + (\ell+1)d^-]$.

With L being the last layer which will be triggered, we can conclude that $\ell+1 = L$ yields the smallest interval and hence gives the pulse separation time as stated in our lemma. In combination with Lemma 2.5.1, which guarantees that every node triggers at most once, this shows that each node will trigger exactly once per pulse.

It remains to show that the stabilization time and the skew bounds hold as proposed. We will use induction on the layers $\ell \in [L + 1]$ to show that (i) for pulses $k > \ell$, the nodes in layers $\ell \leq k$ will not trigger due to spurious trigger messages and (ii) that for all nodes in layers $\ell \leq k$ the skew bounds from Theorem 2.3.10 apply. Note that (i), in combination with the fact that no node is sleeping when pulse k arrives, as shown above, guarantees correct behavior of the nodes in layers $\ell \leq k$ for which Theorem 2.3.10 was shown; thus (ii) will hold.

The anchor of the induction is at $\ell = 0$. As we require layer 0 to be stable to begin with, no spurious pulses will occur, and as the intra-layer skew is $\sigma_0 \leq \Delta + d^-$ the skew bound obviously holds.

For the induction step from ℓ to $\ell + 1$, we use the fact that every node in layer ℓ will exactly fire once. As shown above in the derivation of the pulse separation time, every node in layer $\ell + 1$ will be awake to receive a trigger message. Furthermore, as shown in the proof of Lemma 2.5.1, every node in layer $\ell+1$ will sleep long enough so that no trigger message from a correct neighbor

will be received after waking up. Hence every node in layer $\ell + 1$ will fire exactly once. As state above, (ii) directly follows from this fact, which concludes the proof. □

CHAPTER 3

HEX – Implementation and Results

This chapter explores, complementary to the theoretical approach in the previous chapter, the practical implementation of a HEX node and the results gained by simulating a grid of HEX nodes in different scenarios.

The simulations were conducted to (i) complement the analytic worst-case skew and stabilization results of Chapter 2, and to (ii) explore the behavior of HEX under conditions which are beyond a reasonable analytical treatment. As for (i), although the worst-case skews for HEX are rather large, the scenarios required for them to manifest themselves are quite rare. We hence conjectured that the skew in typical "average" scenarios would be much better. Simulation experiments are the method of choice for verifying this conjecture. With respect to (ii), an analytic treatment even of the worst-case skew in the presence of just a single fault suffers from combinatorial explosion [15]. Again, simulation is a convenient way to study the behavior of HEX even in the presence of multiple faults.

As the HEX algorithm is designed to be self-stabilizing, it is also of interest how long this process takes under typical circumstances. As we will show, the effect of faults on the grid is mostly local, so it is also of interest if and how well self-stabilization works in a grid with permanent faulty nodes.

We will start with a description of the VHDL implementation of a HEX node and then proceed with the MATLAB and ModelSim simulation environments.

3.1 VHDL Implementation

To be able to perform ModelSim simulations, and to provide a reference implementation, the HEX algorithm was implemented in VHDL. This implementation is a composition of multiple components, as shown in Figure 3.1. The main component is the *asynchronous state machine* (ASM), the primary input (trigger) of which is provided by a threshold gate. The threshold gate takes the outputs of the receivers of the incoming links of a HEX node and checks whether one of the firing conditions, defined by Algorithm 1, is satisfied.

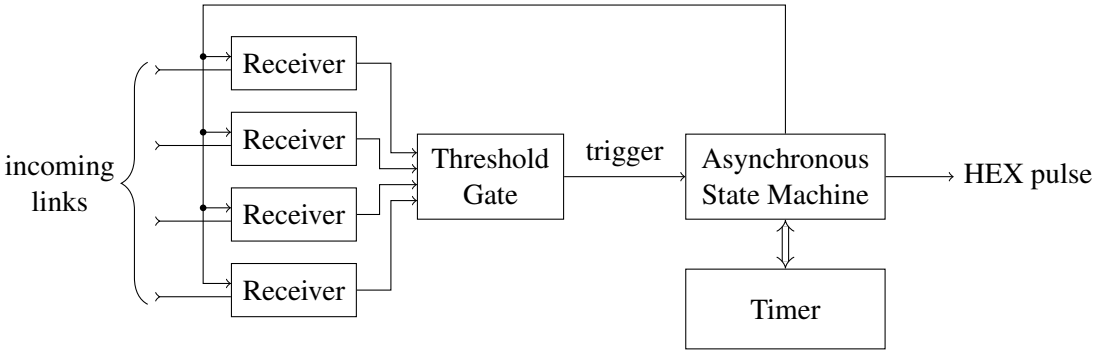


Figure 3.1: Schematic overview of the implementation of a HEX node. On the left are the receivers. Those are connected to the threshold gate, which provides the ASM with a trigger signal when one of the firing condition is satisfied. The ASM performs the state transitions, interacts with the timer, and outputs a HEX pulse when necessary. The reset signal is not drawn in this figure to avoid clutter.

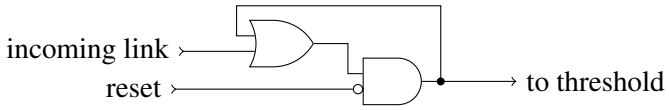


Figure 3.2: The receiver component of a HEX node. A simple memory flag, which stores, through a loopback, any trigger message (= rising signal transition) received from a neighbor.

The implemented design was synthesized using the UMC 90 nm standard cell library [71], using Synopsis® Design Compiler version C-2009.06-SP4. Note that we used an augmented version of the UMC 90 nm library, which includes a custom Muller C-Gate [68] added in the context of the DARTS project [27, 28]. To provide timing characteristics for this gate, the timing information of the AND-Gate of said library was used; the resulting inaccuracy is negligible w.r.t. our purposes.

3.1.1 Receiver

The receivers are implemented as simple memory flags. Their sole purpose is to store an incoming trigger message, which is just a rising signal transition on the single wire of the link, from a neighbor until it is reset. The implementation is shown in Figure 3.2. In the current implementation, the input of a link is just tied to the HEX pulse output of the sender node, i.e., a trigger message is just the HEX pulse output of the ASM.

3.1.2 Asynchronous State Machine

The state machine has basically three states, and is similar to the quick cycle state machine of FATAL$^+$ [17]. The states are the following:

wait The state machine waits for one of the firing conditions to become true.

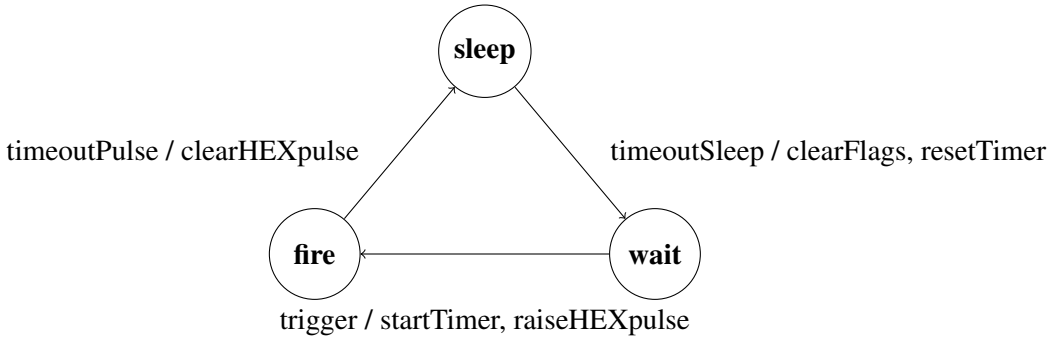


Figure 3.3: The state transition diagram of the ASM. Circular nodes represent states, edges represent the transitions. The transition label includes the guarding signal of the transition separated by a slash from the outputs which are enabled on taking the transition.

fire At least one of the firing conditions has become true and a HEX pulse is generated.

sleep The state machine sleeps for some time within the interval $[T^-, T^+]$.

The transitions between those states are cyclic, i.e. *wait* to *fire*, *fire* to *sleep* and *sleep* to *wait*, as shown in Figure 3.3. As the quick cycle state machine in FATAL$^+$ was implemented as a *hybrid state machine* (HSM), the first implementation also used a HSM.

A HSM is a combination of an ASM with a synchronous *transition state machine* (TSM). The asynchronous part decided which transition should be taken and starts a local oscillator. This oscillator acts as the clock for the TSM, which then executes the actual state transition. This is done to ensure that only one transition is done at a time, as the asynchronous part could enable multiple transition at the same time. Although the HSM is a valid solution for this problem, it led to a large delay between the acceptance of a firing condition and the output of a HEX pulse: In the pre-layout simulation this delay was about 3 ns.[1]

Fortunately, since the state transition diagram in Figure 3.3 is deterministic, i.e., the transition to be taken is predefined in every state and not conditional on certain signals, a pure ASM turned out to be sufficient. Compared to the HSM approach, an additional timeout, *timeoutPulse*, was required to ensure the proper length of the HEX pulse, which was inherently guaranteed by the transition sequence *wait* $\rightarrow$ *fire* $\rightarrow$ *sleep* of the TSM. The design of this ASM was done with Petrify [11].

As described in more detail in Section 1.2.5, Petrify allows the generation of speed-independent asynchronous circuits. The relationship between the signal transitions were given to Petrify in form of the *signal transition graph* (STG) of Figure 3.4. From this input, Petrify generated a net-list, on which a technology mapping to Verilog can be performed. This feature was used, with a generic technology library augmented by Muller C-Gates, to generate a Verilog net-list, which was then manually converted to VHDL. During the manual conversion, a *reset* signal was incorporated into the design, which ensures the required initial states for the ASM. In Figure 3.5, the timing diagram of an execution of the ASM is shown, while Figure 3.6 shows the circuit

[1] This value depends also on the frequency of the oscillator and thus should be taken with caution.

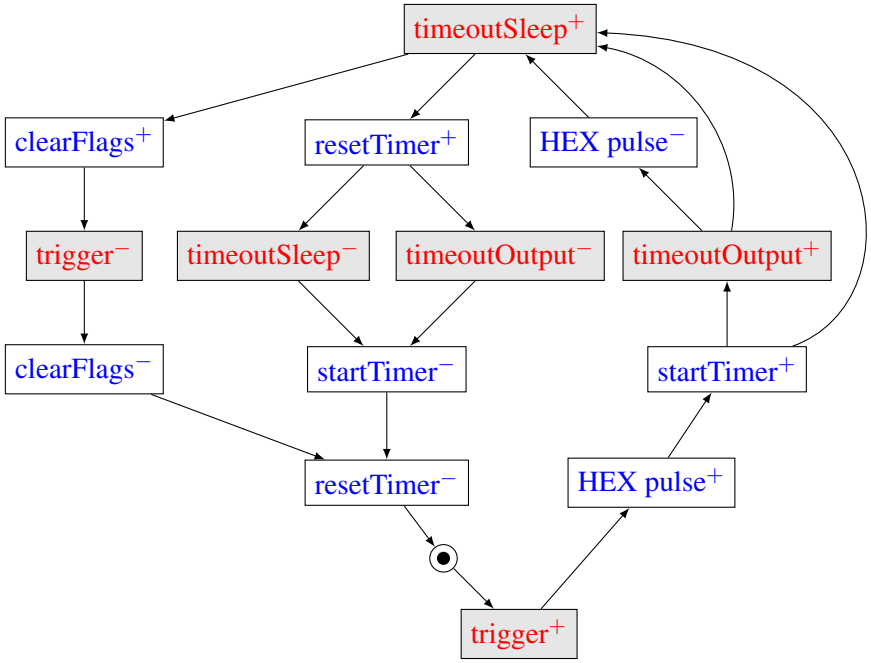


Figure 3.4: The *signal transition graph* (STG) that describes the ASM implementing the HEX algorithm. Nodes with red text, and filled gray represent events on input signals of the state machine, whereas white nodes with blue text represent output signals. A $signal^+$ as a node label represents a rising transition of the signal, $signal^-$ a falling transition. More detail on the interpretation of a STG can be found in Section 1.2.5.

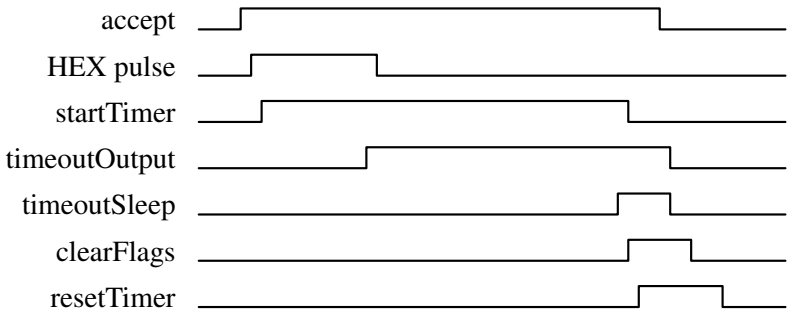


Figure 3.5: Timing diagram of a complete cycle of the ASM

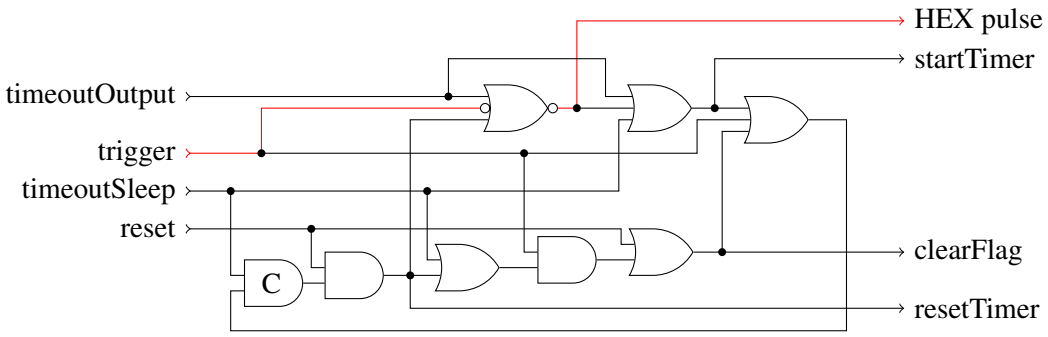


Figure 3.6: The logic elements building the ASM of a HEX node. Input signals are on the left, output signals are on the right. The path from acceptance of a firing condition to generation of a HEX pulse is marked red.

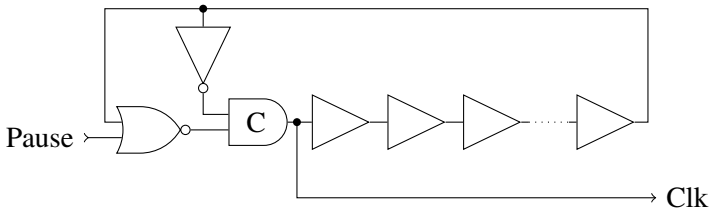


Figure 3.7: Start/stoppable ring oscillator used for generating the clock of the timer. The clock frequency is determined by a series of buffer elements, which determine the delay of a feedback loop that also incorporates a Muller C-Gate, i.e., allows to open/close the feedback loop and hence to start/stop the generation of the clock signal.

diagram of the generated HDL code: The ASM has as its main input the *trigger* signal, which comes directly from the threshold gate. This signal is responsible for triggering a HEX pulse, represented by the signal *HEX pulse*, if the node is not sleeping. The timer is controlled with *startTimer* and *resetTimer*, while the timeouts are introduced into the ASM with *timeoutOutput*, which defines the length of the HEX pulse signal, and *timeoutSleep*, which wakes up the node after sleeping for some time $T \in [T^-, T^+]$. Finally, the signal *clearFlags* is used to reset the receivers' memory flags.

Note that it is possible to find multiple correct STGs for the given ASM. They differ by the detailed dependencies of the transitions, e.g., the *clearFlags* could be enabled after the timeout signal was reset. The version given in Figure 3.4 was chosen, as it does not contain any RS-latches in the output generated by Petrify, which could introduce metastability into the system.

3.1.3 Timer and Ring Oscillator

As the ASM needs some kind of time reference, to guarantee the minimal/maximal duration of the HEX pulse and the sleeping time, a timer is needed. The timer was implemented by a

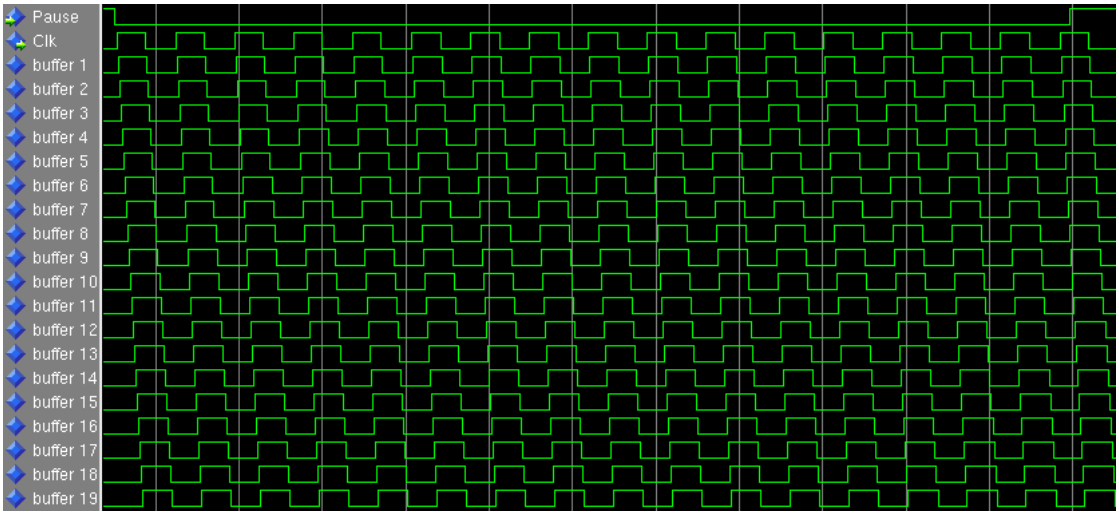


Figure 3.8: A screenshot of a simulation of the ring oscillator. The oscillator has 19 buffer gates, which form the delay chain. On the left, it is also visible how the active high *pause* signal controls the generation of the clock signal.

synchronous counter driven by a start/stoppable ring oscillator, similar to the implementation in FATAL$^+$ [17].

A single timer consists of a register, which stores a value that is incremented with every clock cycle of the ring oscillator, until the oscillator is stopped. When a predefined value is reached, the timer asserts an overflow signal, which represents the timeout. Our implementation of a HEX node uses two timers, which share the same register. The first, lower-range timer is called *timeoutPulse* and specifies the length of the HEX pulse that will be generated by the ASM. The second, higher-range timer is called *timeoutSleep*: it enables the transition from *sleep* to *wait* in the ASM, and stops the oscillator upon its timeout.

The ring oscillator is implemented by an inverter, a chain of buffer gates, which delays the signal and thus determines the clock frequency, and a Muller C-Gate in a feedback loop, cf. Figure 3.7. The Muller C-Gate allows the generation of the clock signal to be paused. This is necessary to avoid metastability in the timer: The signal, which starts the timer, could violate the setup/hold-requirements of the timer register when the clock would always be running. An exemplary execution of the ring oscillator is shown in Figure 3.8. The oscillator has a chain of 19 buffer gates, the delays of which result in the generation of 17 clock cycles while the *pause* signal is disabled.

3.1.4 Implementation Characteristics

In the pre-layout simulations, the response time of the HEX node circuit, from applying the last signal of a firing condition at a receiver to the generation of the HEX pulse, was between 125 ps and 165 ps, depending on which firing condition enabled the triggering of the node. The difference is caused by the difference in length of the signal paths through the threshold gate.

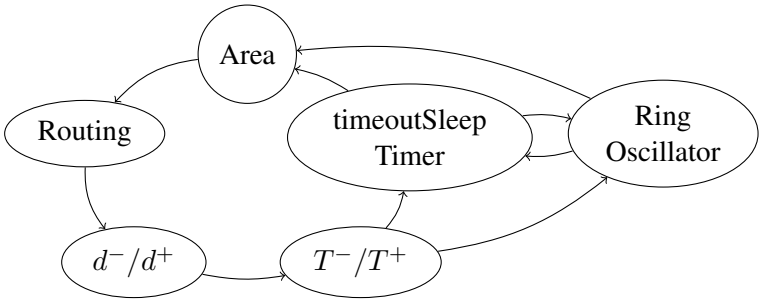


Figure 3.9: Cyclic dependencies of the different factors that influence the required area of the timer and the ring oscillator.

We can expect this response time to increase in the post-layout design, where the wire delays are incorporated, although this should not have an effect on the magnitude of the difference between the former HSM and the ASM implementation. Nonetheless, the post-layout response time will most likely be affected by the actual end-to-end link delays d^- and d^+, which in turn incorporate the response time: As the timeout interval $[T^-, T^+]$ depends on those values, recall Equation (2.8), this directly affects the area required for the timer and the ring oscillator, which in turn can influence these response times, due to routing decisions. Due to this cyclic dependency, a comprehensive verification of the timing parameters is imperative to ensure the correct operation of a HEX grid. Cf. Figure 3.9 for a graphical representation of the cyclic dependencies of the parameters just discussed.

In terms of area, the current HEX node design has size of about $110\,\mu m^2$, without the timer and with the approximated area for the Muller C-Gate. The timer will, very likely, be the dominant factor in terms of area requirements of the design. As the area of the timer depends on the frequency of the ring oscillator and the bits required to represent the timer value, however, it is not possible to give an exact value for the area of the timer. In the designs used for the simulations later on, the area of the timer was 2 to 3 times larger than the area of the other components of the HEX node together. With respect to the post-layout design, we note that the missing routing delays will significantly increase the delay of the feedback loop of the ring oscillator, which allows to reduce the number of buffer gates. Alternatively, the *timeoutSleep* timeout value and thus possibly the number of bits required for the representation of the timer value could be reduced. Both changes have a positive effect on the area required.

3.2 Simulation Environments

3.2.1 MATLAB

The first simulations where conducted with MATLAB, which offers a controllable environment where implementation and other errors could be determined and eliminated easily. Also, the strive for an interactive simulation environment was a primary driver for this approach.

The HEX grid was implemented directly, using a 2D array of nodes. Using object-oriented programming, subclasses for different types of nodes where derived from a superclass and placed

in that array. The node types included layer 0 generators, different kind of faulty nodes, and the regular HEX node.

Every node has a FIFO-queue for every incoming link, which also stores the time when a messages was inserted to ensure correct handling of the link delays. A time instant was simulated by iterating over every node of the grid and calling a *process* function, which executes the nodes' actions for that instant.

As expected, the MATLAB simulations offered flexibility and a high level of control over the simulations. It was possible to define the link delay for every link, and to configure the behavior of every faulty node. Thus, this simulation environment has been used for confirming the worst-case scenario, as well as for the analysis of the effects of faults on a HEX grid. The results have already been published in [16].

Nevertheless, it turned out that the MATLAB simulation environment caused long simulation times, along with other limitations. E.g., the link delays can only be integer values, uniformly distributed within a given range. Therefore, this environment has not been used for the simulation of realistic scenarios.

3.2.2 ModelSim

ModelSim is a well-known environment for simulation and verification of hardware designs developed by Mentor Graphics®.

Our simulation environment was build upon a testbed which was written in the Haskell programming language.[2] Its top-level features are to setup a testbench for a test configuration, start ModelSim for performing the simulations, and process the results. Over the duration of this thesis, the last part was separated into a standalone application, as ModelSim was later used to also generate preprocessed data, which was then used for the statistical evaluation. Figure 3.10 shows a overview of the major components used for the simulation of a specific scenario.

A scenario describes the parameters of the HEX grid that shall be simulated. As some parameters are randomly distributed, e.g., the link wire delays or the placement of the faults, multiple executions of the same scenario were used to acquire meaningful data. We call every such execution a *test set*. For every test set, the process shown in Figure 3.10 is executed, i.e., a new testbench is generated and then simulated.

We also distinguish between two types of scenarios: Scenarios which observe only one pulse in the HEX grid, and ones that observe multiple pulses. We denote with *runs* the number of pulses simulated in a test set. For stabilization experiments, multiple runs are obviously required: For the skew evaluation, a single run is sufficient.

Every scenario is generated from a configuration file provided by the user. From this configuration file, the testbed generates files that build the testbench for a test set of that scenario. The testbench is then used as an input by ModelSim for the simulation of the HEX grid.

We will now explain each component of Figure 3.10 in more detail.

Configuration File: The configuration file describes a specific scenario by defining the HEX grid, which includes the following properties:

[2] www.haskell.org

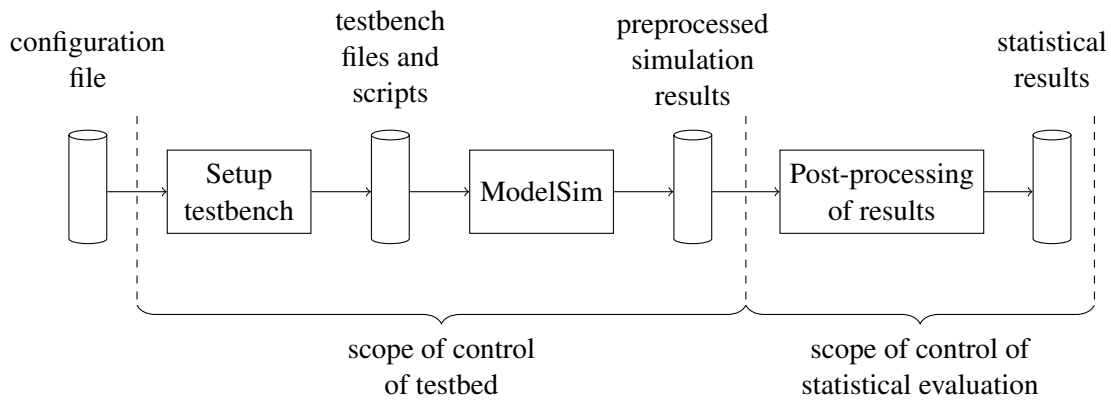


Figure 3.10: This figure visualizes, from left to right, the process from a configuration file to statistical data. Starting with the configuration file, the testbed generates testbench files, which are then simulated with ModelSim. The (preprocessed) simulation results are then further evaluated by a statistical evaluation software, which provides the final results.

- the size of the HEX grid.
- the interval in which the link wire delays shall be distributed.
- the number of test sets that shall be simulated.
- the number of runs per test set.
- the properties of the layer 0 clock source.
- whether the link wire delays shall change on a per-run basis.
- whether an arbitrary initial state of the HEX nodes shall be enforced before the first run. This was used for the stabilization experiments.
- the number and types of the faults which should be placed in the HEX grid, without violating the 1–local fault hypothesis. The types of faults include
  - fail-silent faulty nodes.
  - Byzantine faulty nodes, which behave, on a per-link basis, either fail-silent or send constantly a (fast) trigger message.
  - link-faulty nodes, i.e., correct nodes where a few outgoing links are faulty.

  The behavior of the Byzantine and the link-faulty nodes are chosen randomly on a per-link basis, in every test set.
- the initial seed for the random number generator used.

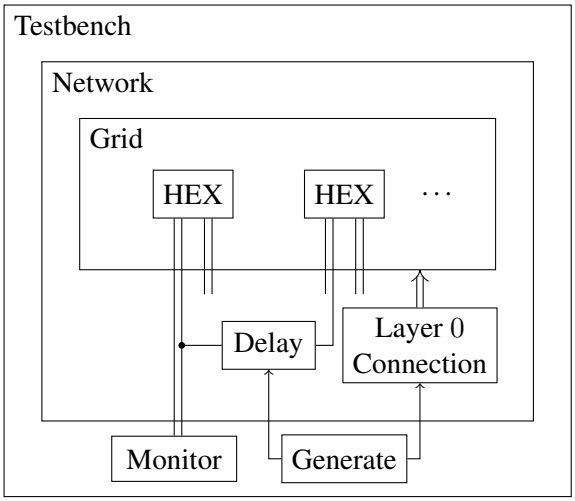


Figure 3.11: Simplified overview of the testbench generated by the testbed. The *Testbench* monitors the signals applied to and generated by the HEX nodes. It also acts as a clock source for the layer 0 pulses and selects the delays applied to the links, if they should change with every run. The *Network* generates the interconnect links between the nodes and thereby constructs the HEX grid. These interconnection links are equipped with delay elements, which simulate the link delays. The *Grid* instantiate the HEX nodes, or the selected faulty nodes, at the specified places in the HEX grid.
To avoid clutter, this figure only shows two HEX nodes, with one connected link. Also only two signals from one node are connected to the *Monitor*.

Testbench: As stated above, the testbed is provided with a configuration file for a specific test configuration, from which multiple files are generated. These files are needed to build a complete testbench and to setup ModelSim to simulate the specific test set of the given configuration, cf. Figure 3.11. The files are

- the *Testbench*. This design entity acts as a clock source for the layer 0 pulses, selects the link wire delays, monitors the HEX grid and applies the reset signal to the other component of the testbench on startup. The actual monitoring of the system is not done within the testbench, but rather uses ModelSim facilities. Nonetheless, the signals are mapped to allow later enhancements, e.g., controlling more intelligent faults.

- the *Network* configuration, which is responsible for connecting the HEX nodes, thereby forming the HEX grid.

- the *Grid*, which is not generated by the testbed, but is rather a given entity that is instantiated in the Network and configured to generate the HEX nodes needed. I.e., the Grid instantiates all HEX nodes and faulty nodes and provides a signal mapping for them.

- an optional script, which is executed by ModelSim after the reset signal was applied to force an arbitrary state of all HEX nodes. This is done by forcing certain signals in the

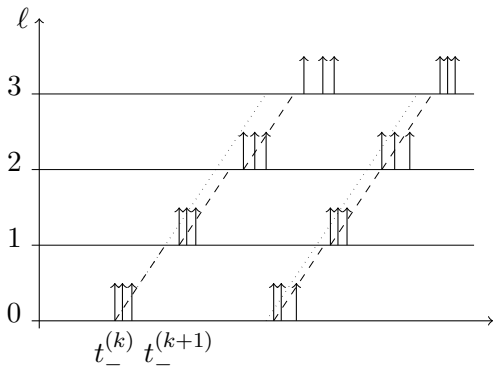


Figure 3.12: Exemplary multi-pulse execution. The arrows mark the triggering of certain nodes, the ℓd^- lines are dotted and the dashed lines mark the respective $\min_{i\in[W]}\left(t_{\ell,i}^{(k)}\right) + d^-$ line.

HEX nodes to a randomly selected value, wait for a certain time until the values have propagated through the nodes, and then releasing the signals and let the nodes progress.

- a script, which tells ModelSim the signals that shall be monitored and later dumped as preprocessed simulation results. This is done via the list feature of ModelSim, which proved to be less data and time consuming, w.r.t. post-processing, than the processing of waveform files.

ModelSim: ModelSim is an environment for simulation and verification of hardware designs developed by Mentor Graphics®. ModelSim is frequently used in industrial practice and allows simulations from structural to post-layout level. The software allows extensive control over the signals in the design, monitoring of the same and, if applicable, timing checks on the components of the design.

Statistical Evaluation: The preprocessed results, generated by ModelSim, are finally analyzed by a statistical evaluation application also written in Haskell. The operations conducted on the results are data intensive but, for most parts, trivial. One operation stands out as being more complex, though: In the test sets with multiple runs, it must be determined whether some generated HEX pulse corresponds to pulse k or $k+1$. The trivial approach of counting from the last pulse recorded backwards does not work in these experiments, as, due to faults and stabilization, nodes can fire multiple times during the period of one pulse, which must be correctly detected. In settings with sufficient pulse separation time, the simple approach of splitting the timeline for node (ℓ, i) into intervals of the form $[t_-^{(k)} + \ell d^-, t_-^{(k+1)} + \ell d^-)$, with $t_-^{(k)}$ being the time when the first node in layer 0 triggers pulse k, is sufficient.

With decreasing pulse separation time, it is possible that $t_{\ell,i}^{(k)} \geq t_-^{(k+1)} + \ell d^-$, due to the inevitable jitter $\ell\varepsilon$ of the triggering times, i.e., there is no way to unambiguously assign a HEX pulse to pulse k or pulse $k+1$. To handle this case, a more involved approach is required. Therefore, instead of the theoretical lower interval bound, the practical lower interval bound is

taken: A node in layer $\ell + 1$ cannot trigger pulse k before the first node[3] in layer ℓ has done so plus the minimal link delay d^-, i.e., $\min_{i\in[W]} \left(t^{(k)}_{\ell+1,i}\right) \geq \min_{i\in[W]} \left(t^{(k)}_{\ell,i}\right) + d^-$. A visualization of the difference between those two approaches is shown in Figure 3.12.

3.3 Simulation Results

3.3.1 MATLAB Simulations

First, we take a closer look at the propagation of a single HEX pulse through the HEX grid without detailed consideration of the skews. These simulations where conducted with MATLAB, and the results are presented in wave propagation diagrams like Figure 3.13. Those diagrams show a three-dimensional representation of the propagation of a single pulse, where the xy-plane spans the columns/layers of the grid, and the z-axis gives the triggering times $t_{\ell,i}$ of the respective node (ℓ, i). For readability, all nodes on the same layer are connected by a straight line, i.e., the triggering times $(\ell, i, t_{\ell,i})$ and $(\ell, i+1, t_{\ell,i+1})$ for all $\ell \in [L+1]$ for $L = 20$ and $i \in [W-1]$ for $W = 10$ are connected.

The results shown in this section are always a single test set with randomly chosen link delays, except for the worst-case scenario shown in Figure 3.17. The behavior of the Byzantine faulty nodes is statically defined so that the nodes send their trigger messages too early (with a link delay of 0 ns) to their right and up-right neighbor, while the link delay to the other two neighbors is too slow ($3d^+$).

Figure 3.13 provides a wave propagation with uniform link delays in the interval $[50, 55]$,[4] with initial skew potential $\Delta_0 = 0$, recall Definition 2.3.6. We see a very even propagation of the pulse throughout the grid, as it was expected in this "perfect" scenario.

With this even progress in the fault-free case in mind, the effects of faults on this propagation, especially the recovery of the grid from those faults, at higher layers, becomes interesting. Figure 3.14 shows a wave propagation under the same settings as in the fault-free case, but with one Byzantine faulty node at $(3, 8)$ displayed with a triggering time $t_{3,8} = -1$. Due to the above defined behavior of the Byzantine faulty nodes, the up-left neighbor is left triggered, which causes a slow-down effect which ripples through the grid along the diagonal until its recovery after W layers; Subsequently, the pulse again propagates evenly. Compared to Figure 3.13, this causes a later absolute triggering time for some nodes in layers above the layer where the fault occurred, however: The triggering times in the last layers of Figure 3.14 occur about d^+ later than in Figure 3.13.

The case of a single fail-silent node produces a similar result as the given Byzantine case, as this type of fault can only slow-down other nodes. Thus, we will turn our attention to entire "clusters" of fail-silent nodes, which violate the 1–local fault hypothesis of Definition 2.4.2. Figure 3.15 shows a cluster of fail-silent nodes in the lower layers of the grid, namely $(3, 7)$, $(3, 8)$, $(4, 6)$, $(4, 7)$, $(4, 8)$, $(5, 6)$, and $(5, 7)$, also displayed with triggering time -1. As expected from this kind of violation of the fault hypothesis, there is one node $(6, 6)$, which is not faulty but does not trigger. This is caused by the two faulty lower neighbors of this node. But again, it is

[3] In the fault-free case, the triggering of the second node would be sufficient.

[4] Recall that the MATLAB simulation use integer values for the link delays.

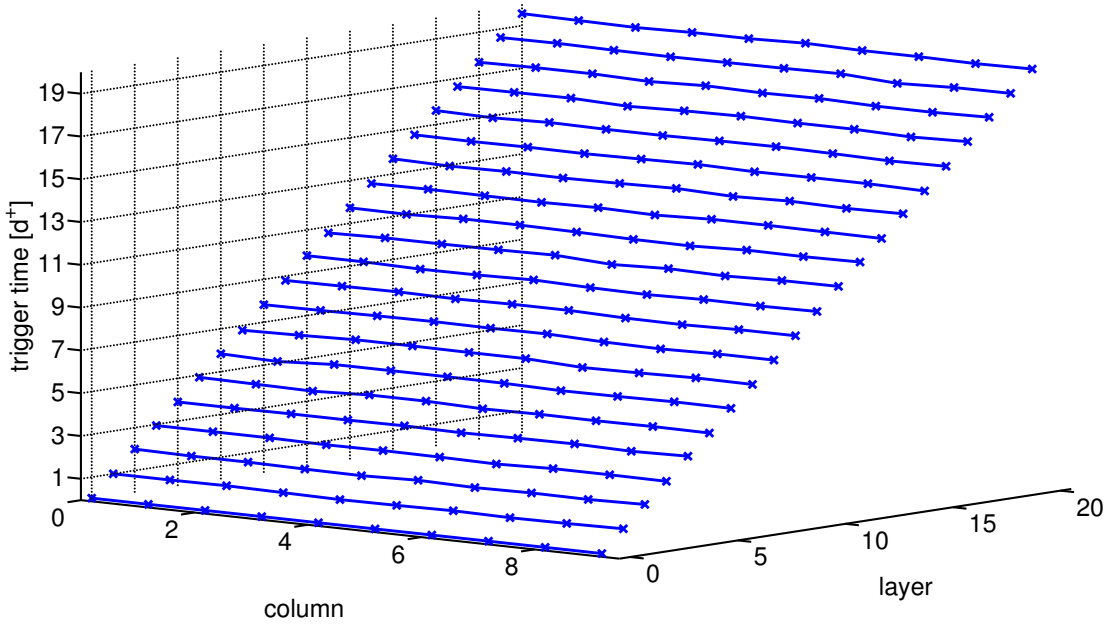


Figure 3.13: Pulse wave propagation in a grid with 21 layers ($L = 20$), 10 columns, link delays uniformly chosen in the interval $[50, 55]$, and a layer 0 skew potential of $\Delta_0 = 0$.

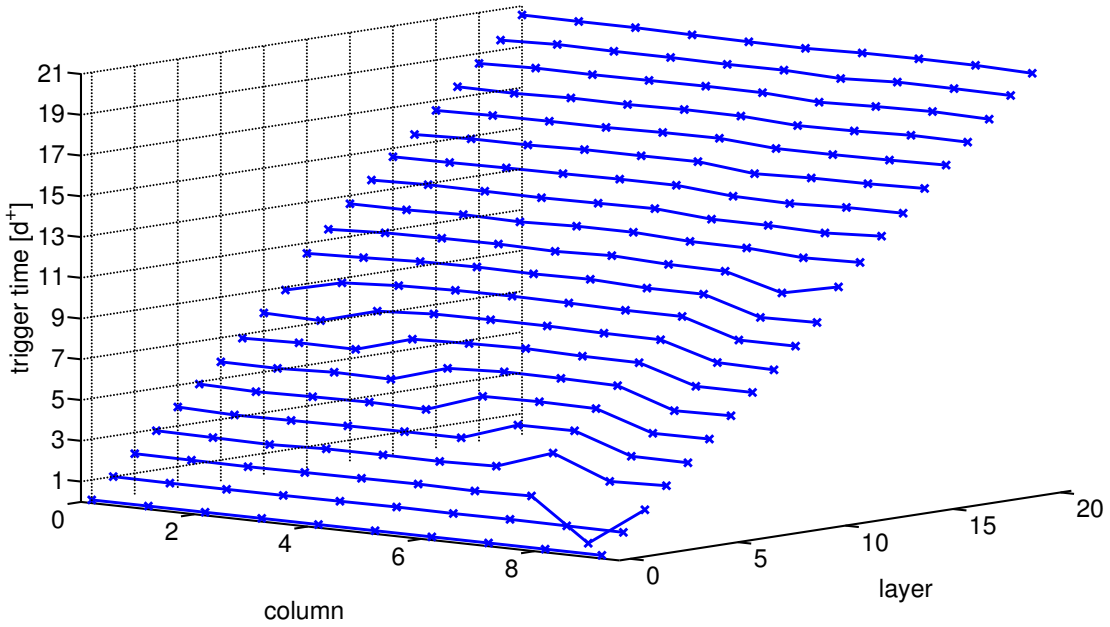


Figure 3.14: Pulse wave propagation with link delays uniformly chosen in the interval $[50, 55]$, with $\Delta_0 = 0$ and a single Byzantine node at $(3, 8)$. The faulty node is displayed with a triggering time of -1.

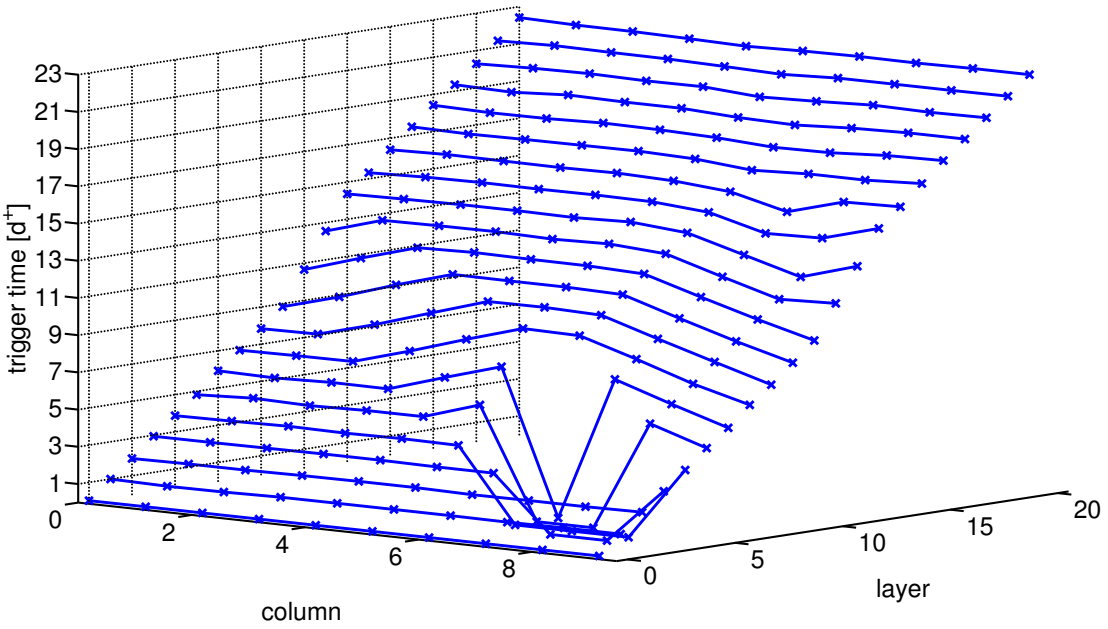


Figure 3.15: Pulse wave propagation with link delays uniformly chosen in the interval $[50, 55]$, with $\Delta_0 = 0$ and a cluster of fail-silent nodes at $(3, 7)$, $(3, 8)$, $(4, 6)$, $(4, 7)$, $(4, 8)$, $(5, 6)$, and $(5, 7)$. The faulty nodes are displayed with a triggering time of -1. Observe that the cluster "shadows" the node $(6, 6)$, which is not faulty, but cannot be triggered and is thus displayed with a triggering time of 0 in this plot.

apparent that this effect evens out after about W layers, after which the pulse propagates evenly again. We also see an increase in the triggering time, as well as similar inter- and intra-layer skews. The behavior of the wave propagation with this cluster is hence again similar to that of the single Byzantine fault.

Figure 3.16 shows a setting with multiple isolated and also clustered Byzantine faults, namely $(2, 6)$, $(2, 7)$, $(2, 8)$, $(6, 8)$, and $(13, 1)$. We observe again effects similar to those of Figure 3.15. Note that this wave propagation shows a rather benign behavior, however, caused by the "nice" behavior of the Byzantine faulty nodes. Actually, the cluster of Byzantine faults in layer 2 alone could also introduce spurious pulses into the grid, which would propagate through all layers.

Finally, Figure 3.17 shows an execution leading to the worst-case skew predicted by Theorem 2.3.10. To accomplish this, the link delays have been chosen deterministically according to the worst-case scenarios used in its proof: The basic idea is to have a fast triggering triangle from column 8 at layer 0 to the left side, which leads to a pulse reaching the top layer very fast. In the complementary slow triangle from $(8, 1)$ towards column 9 the link delays are gradually decreased on a per-layer basis. The lower layers have maximal link delays which are used to build up skew potential. The link delays then gradually decrease, as can be seen in Figure 3.17 by the decreasing slope of the triggering times, so that the nodes in the triangle are all left triggered. While the right side of the grid is generally set to maximal link delay, it is additionally stalled by a large initial skew of the clock sources. The nodes in column 16, except the clock source in

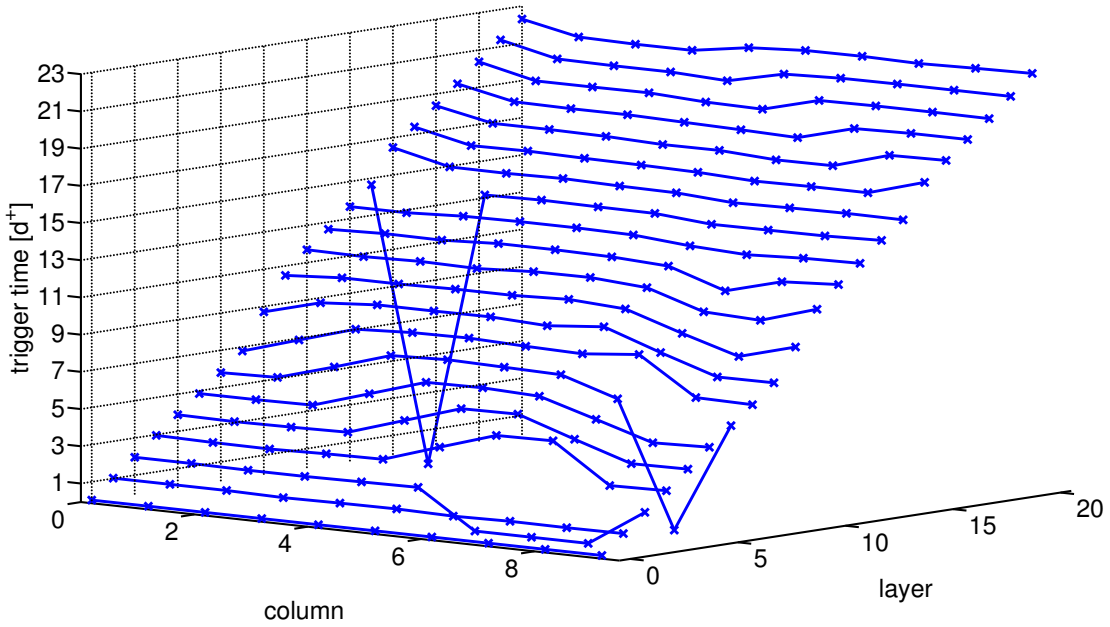


Figure 3.16: Pulse wave propagation with link delays uniformly chosen in the interval $[50, 55]$, with $\Delta_0 = 0$ and multiple Byzantine faulty nodes at $(2, 6)$, $(2, 7)$, $(2, 8)$, $(6, 8)$, and $(13, 1)$. The faulty nodes are displayed with a triggering time of -1.

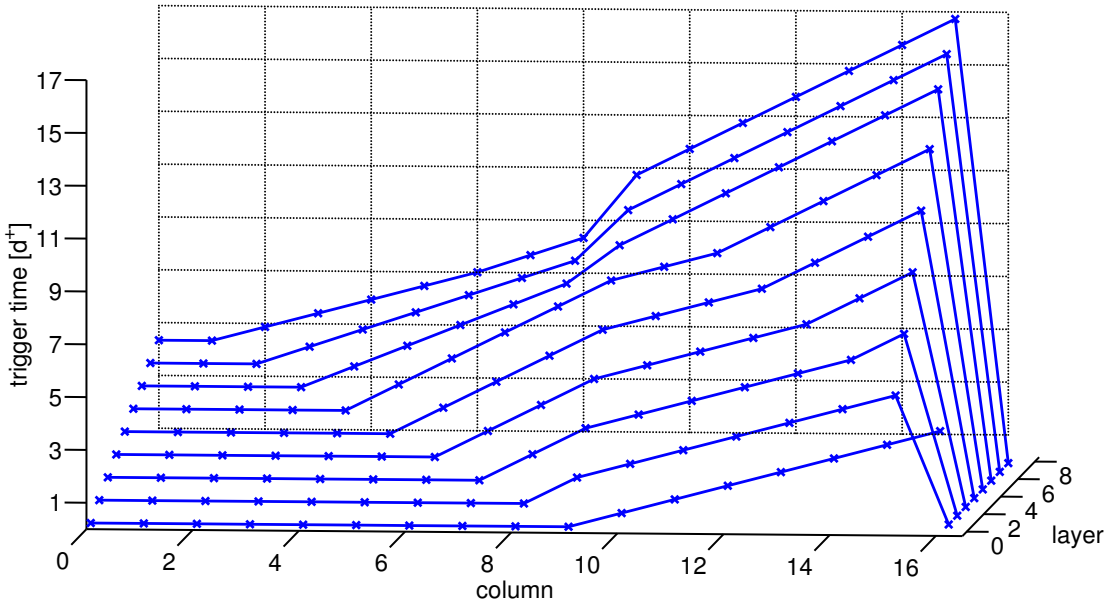


Figure 3.17: A worst-case pulse propagation wave, with the maximal intra-layer skew in layer 8 between the nodes in column 8 and 9. Note that the nodes in column 16, except the clock source in layer 0, are deliberately fail-silent and hence displayed with a triggering time of -1.

layer 0, are artificially shown as fail-silent faulty nodes (triggering time 0) to keep the width of the grid small and still separate the fast left side of the grid from the slow right side.

The fast pulse from the left side finally ripples through the top layer and meets the slow part (without being able to accelerate the latter) of the grid, and thus generates a high intra-layer skew between the nodes $(8, 8)$ and $(8, 9)$.

3.3.2 Skew Evaluations

In this section, we will focus on the detailed quantitative evaluation of the typical skew between neighboring HEX nodes in the grid. We will observe the influence of the initial skew of the clock source, the dependence on the layer, and the dependence of the skews on faults in the HEX grid.

In all our experiments, we will consider a HEX grid with 100 layers and 25 columns. The link wire delays were chosen uniformly distributed in the interval of $[7, 8]$ ns. Adding the processing delays of the HEX node implementation, mentioned in Section 3.1.4, the end-to-end link delays $[d^-, d^+]$ are in the interval $[7.125, 8.165]$ ns.

The key quantities observed in the simulations were:

- the (absolute) layer ℓ intra-layer neighbor skews $|t_{\ell,i} - t_{\ell,i-1}|$ of every node (ℓ, i) in layer $\ell > 0$. This skew is defined as an absolute value due to the symmetry in the topology (and thus the skews) within a layer.

- the layer ℓ inter-layer neighbor skews $t_{\ell,i} - t_{\ell-1,i}$ and $t_{\ell,i} - t_{\ell-1,i+1}$ of every node (ℓ, i) in layer $\ell > 0$.

Let $\sigma_\ell^{\mathrm{op}} = \mathrm{op}_{i\in[W]}(|t_{\ell,i} - t_{\ell,i+1}|)$ with $\mathrm{op} \in \{\mathrm{avg}, \max\}$ denote the average and maximum (absolute) layer ℓ intra-layer skew, respectively. Similarly, let $\hat{\sigma}_\ell^{\mathrm{op}} = \mathrm{op}_{i\in[W]}(t_{\ell,i} - t_{\ell-1,i}, t_{\ell,i} - t_{\ell-1,i+1})$, where $\mathrm{op} \in \{\min, \mathrm{avg}, \max\}$, be the (signed) inter-layer skew between layer ℓ and $\ell - 1$. The global intra-layer resp. inter-layer skews in the entire system are defined as $\sigma^{\mathrm{op}} = \mathrm{op}_{\ell\in[L+1]}(\sigma_\ell^{\mathrm{op}})$ resp. $\hat{\sigma}^{\mathrm{op}} = \mathrm{op}_{\ell\in[L+1]\setminus\{0\}}(\hat{\sigma}_\ell^{\mathrm{op}})$. Note that the minimum of the global intra-layer skew is not provided, as it is close or equal to 0. As we used single-run test sets to collect the skew data σ^{op}, etc. they also define their respective per-test set result. To get a deeper understanding of the distribution of the skews, we also determined q_x, the x^{th} quantile of the intra-layer skews of a test set (over all layers). $\hat{q}_x$ is the equivalent of q_x for the inter-layer skew.

For the following simulation results, 300 test sets, with 1 run each, were simulated. In each test case, the link delays were randomly chosen in the above mentioned interval. Furthermore, if applicable, the layer 0 clock skews were also randomly chosen for every test set. The placement of the faulty nodes, and their fault-type, was also randomly chosen for each test set. The Byzantine faulty nodes behave, on a per-link basis, either fail-silent or continuously send a (fast) trigger message.

3.3.2.1 Initial Skew-Dependence

For the evaluation of the influence of the initial skews of the clock sources on the HEX grid we employed four different layer 0 clock sources. Those sources differ in their Δ_0 skew potential

		σ^{op}		$\hat{\sigma}^{\mathrm{op}}$		
scenario	init. layer 0	avg	max	min	avg	max
(i)	0	0.40	3.63	7.13	7.91	11.58
(ii)	rand. [0,d^-]	0.46	6.97	7.13	7.94	15.07
(iii)	rand. [0,d^+]	0.46	7.86	7.13	7.94	15.88
(iv)	ramp d^+	1.41	8.16	0.96	8.36	16.28

Table 3.1: Intra- and inter-layer skew σ^{op} and $\hat{\sigma}^{\mathrm{op}}$ of all nodes in a 100×25 grid for uniformly distributed link delays in $[7.125, 8.165]$ ns. The values are determined by applying the specified op over the calculated results of 300 test sets with 1 run each.

which, as proven in Lemma 2.3.7, affects only the lower $W-3$ layers of the grid. The scenarios were chosen as follows:

(i) All layer 0 nodes fire at the same time. Thus $\sigma_0 = \Delta_0 = 0$, which is the best-case.

(ii) The nodes in layer 0 fire randomly within $t + [0, d^-]$, after a given starting time t, resulting in $\sigma_0 \approx d^-$ and $\Delta_0 = 0$.

(iii) The nodes in layer 0 fire randomly within $t + [0, d^+]$, after a given starting time t, resulting in $\sigma_0 \approx d^+$ and $\Delta_0 \approx \varepsilon$.

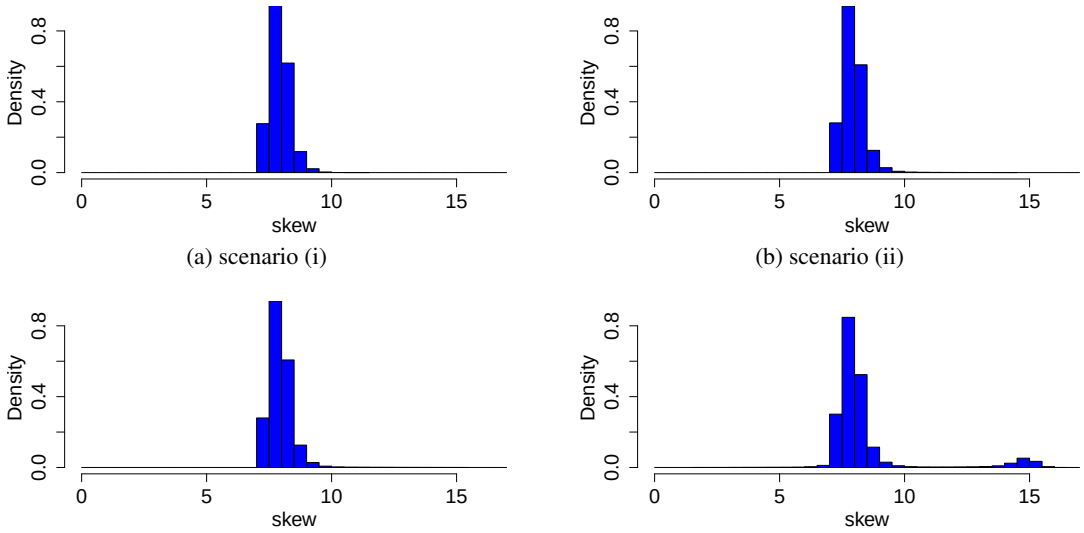
(iv) Every node in layer 0 deterministically triggers at a time that results in a difference of d^+ between two neighboring nodes in a ramping fashion, i.e.,

$$\forall i \in \{1, \dots, W-1\} \colon t_{0,i} = \begin{cases} t_{0,i-1} + d^+ & 0 < i < W/2 \\ t_{0,i-1} - d^+ & \text{otherwise} \end{cases}$$

This results in $\sigma_0 = d^+$ and $\Delta_0 = W\varepsilon/2$ for even W (for odd W, those values are close approximations).

Note that scenario (iii) corresponds to the average-case generated by a clock source with skew bound d^+, while scenario (iv) corresponds to the worst-case under the same conditions.

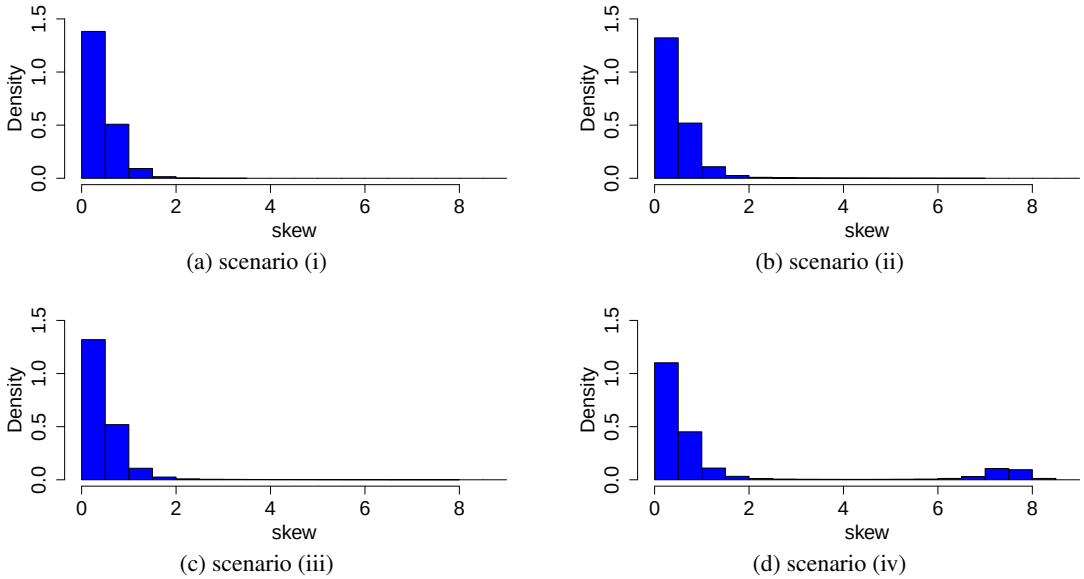
In Table 3.1 and Figures 3.18 and 3.19, the effects of the different scenarios (i) – (iv) on the skews are shown. While in Table 3.1 the statistical data provided is the specified operation on the results over the entire test sets, e.g., $\max(\sigma^{\max})$, the histograms are created over all skew data of the individual executions. From Table 3.1, it is apparent that scenario (i) leads to a low maximal intra-layer skew, as one would expect. Scenarios (ii) and (iii) have a higher maximal intra-layer skew, but still in the range of d^-; The increase of the maximal inter-layer skew can be accounted to the Δ_0 of layer 0. Scenario (iv) results, as expected, in excessive minimal and maximal skews, although Figures 3.18d and 3.19d reveal that those values are rare compared to the average. Note that the minimal inter-layer skew close to 0 even suggests the possibility of negative inter-layer skews under extreme clock source skews. Yet the maximal inter-layer skew is still below $2d^+$, which is below the worst-case skew of ≈ 27 ns predicted by Theorem 2.3.10. Note that, for scenarios (i) – (iii), $\hat{\sigma}^{\min} \approx d^-$, which reveals that without faults and excessive skews the HEX nodes will always be triggered by their lower neighbors, resulting in average skews close to the minimum.



(a) scenario (i) (b) scenario (ii)

(c) scenario (iii) (d) scenario (iv)

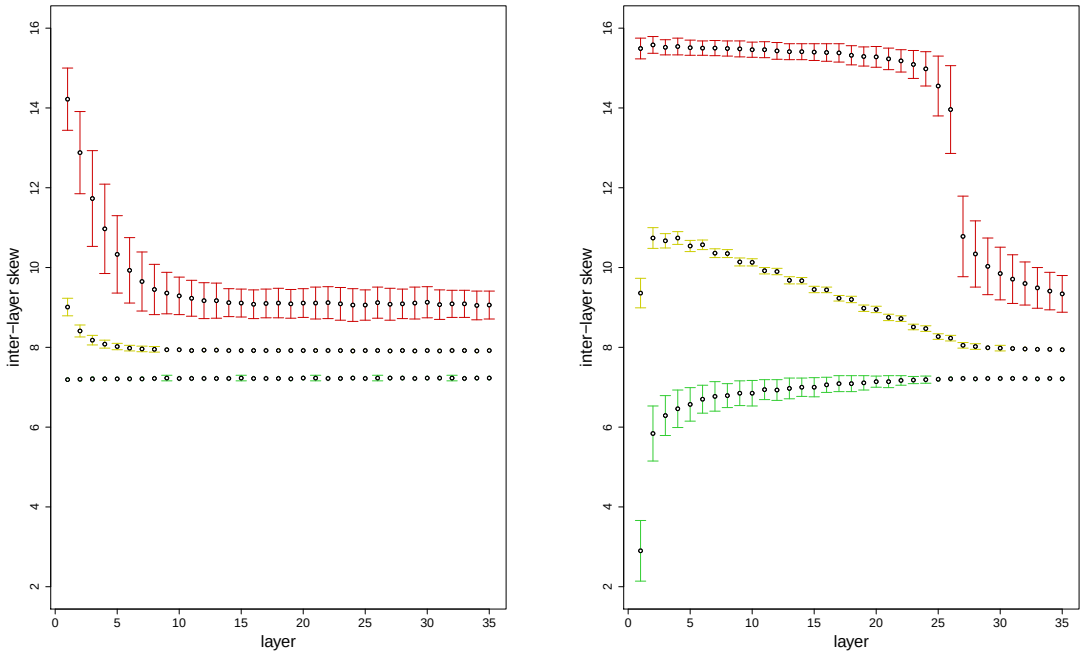
Figure 3.18: Cumulated histogram for the global inter-layer skew $\hat{\sigma}_\ell$ for all $\ell \in [L+1]$ (in ns) from 300 test sets, of 1 run each. The used scenario is given in the sub-caption of the respective figure. Note that skews < 7 ns occurred only in scenario (iv).



(a) scenario (i) (b) scenario (ii)

(c) scenario (iii) (d) scenario (iv)

Figure 3.19: Cumulated histogram for the global intra-layer skew σ_ℓ for all $\ell \in [L+1]$ (in ns) from 300 test sets, of 1 run each. The used scenario is given in the sub-caption of the respective figure.



(a) scenario (iii)

(b) scenario (iv)

Figure 3.20: Visualization of the inter-layer skews in scenario (iii) and (iv) (Tables 3.2 and 3.3). The averages of the data series are plotted with their respective standard deviation on a per layer basis. The top data series (red whiskers) is $\hat{\sigma}^{\max}$, the middle (yellow) one is $\hat{\sigma}^{\text{avg}}$, and the lower (green) series is $\hat{\sigma}^{\min}$.

3.3.2.2 Layer Dependence

Knowing the influence of the layer 0 skew in general already, we now take a closer look at the layer-dependence of this influence on the inter-layer skew. Table 3.2 and Figure 3.20a show the per-layer skews for scenario (iii). The data provided is the average and the standard deviation of the specified operation, e.g., $\text{avg}\left(\sigma^{\max}\right) \pm \text{stddev}\left(\sigma^{\max}\right)$. This data shows a fast decline of the maximal inter-layer skew, which is nearly constant from layer 12 onwards. The average skew stabilizes fast, and an influence on the minimal inter-layer skew is almost not noticeable.

Table 3.3 and Figure 3.20b show the per-layer inter-layer skew for scenario (iv). As mentioned above, this scenario can be seen as a worst-case setting of a clock source with bounded skew. Apparently, there are very large differences between the minimum and the maximum in the lower layers, but, as proven in Lemma 2.3.7, from layer 22 on, which is $W - 3$ in our experiments, the influence of Δ_0 vanishes rapidly and the values reach similar ranges as for scenario (iii) just after a few more layers.

3.3.2.3 Faults

To study the behavior of a HEX grid in the presence of faults, where analytical results are difficult to obtain, we conducted a suite of dedicated simulation experiments. As faulty nodes have a

layer ℓ	$\hat{\sigma}_\ell^{\min}$	$\hat{\sigma}_\ell^{\text{avg}}$	$\hat{\sigma}_\ell^{\max}$
1	7.19 ± 0.04	9.01 ± 0.22	14.22 ± 0.78
2	7.20 ± 0.05	8.41 ± 0.15	12.88 ± 1.03
3	7.21 ± 0.05	8.18 ± 0.12	11.73 ± 1.20
4	7.21 ± 0.06	8.08 ± 0.10	10.97 ± 1.12
5	7.21 ± 0.06	8.02 ± 0.08	10.33 ± 0.97
6	7.21 ± 0.05	7.98 ± 0.07	9.93 ± 0.82
7	7.21 ± 0.05	7.96 ± 0.07	9.65 ± 0.74
8	7.22 ± 0.06	7.95 ± 0.07	9.45 ± 0.63
9	7.23 ± 0.07	7.94 ± 0.06	9.36 ± 0.52
10	7.22 ± 0.06	7.94 ± 0.06	9.29 ± 0.47
11	7.22 ± 0.06	7.92 ± 0.06	9.23 ± 0.45
12	7.22 ± 0.06	7.93 ± 0.06	9.17 ± 0.45
13	7.22 ± 0.06	7.93 ± 0.06	9.17 ± 0.44
14	7.22 ± 0.06	7.92 ± 0.06	9.12 ± 0.35
15	7.23 ± 0.07	7.92 ± 0.06	9.11 ± 0.35
16	7.22 ± 0.06	7.92 ± 0.06	9.08 ± 0.36
17	7.22 ± 0.06	7.92 ± 0.06	9.10 ± 0.36
18	7.22 ± 0.06	7.92 ± 0.05	9.11 ± 0.37
19	7.21 ± 0.06	7.92 ± 0.06	9.09 ± 0.36
20	7.23 ± 0.06	7.92 ± 0.06	9.11 ± 0.36
21	7.23 ± 0.07	7.92 ± 0.05	9.11 ± 0.40
22	7.22 ± 0.06	7.92 ± 0.06	9.12 ± 0.40
23	7.22 ± 0.06	7.92 ± 0.06	9.09 ± 0.41
24	7.23 ± 0.06	7.91 ± 0.06	9.06 ± 0.41
25	7.22 ± 0.06	7.92 ± 0.06	9.06 ± 0.38
26	7.23 ± 0.07	7.92 ± 0.06	9.12 ± 0.39
27	7.23 ± 0.06	7.91 ± 0.06	9.08 ± 0.40
28	7.23 ± 0.06	7.92 ± 0.06	9.09 ± 0.37
29	7.22 ± 0.06	7.91 ± 0.06	9.11 ± 0.40
30	7.23 ± 0.06	7.92 ± 0.06	9.13 ± 0.39
31	7.23 ± 0.06	7.91 ± 0.06	9.07 ± 0.37
32	7.23 ± 0.07	7.92 ± 0.06	9.09 ± 0.34
33	7.22 ± 0.06	7.92 ± 0.06	9.09 ± 0.34
34	7.23 ± 0.06	7.91 ± 0.06	9.05 ± 0.36
35	7.23 ± 0.06	7.92 ± 0.06	9.06 ± 0.35

Table 3.2: Average and standard deviation of $\hat{\sigma}_\ell^{\min}$, $\hat{\sigma}_\ell^{\text{avg}}$, and $\hat{\sigma}_\ell^{\max}$, taken over 300 test sets with 1 run each, of scenario (iii).

layer ℓ	$\hat{\sigma}_\ell^{\min}$	$\hat{\sigma}_\ell^{\text{avg}}$	$\hat{\sigma}_\ell^{\max}$
1	2.90 ± 0.76	9.36 ± 0.37	15.49 ± 0.26
2	5.84 ± 0.69	10.74 ± 0.26	15.58 ± 0.21
3	6.29 ± 0.50	10.67 ± 0.18	15.52 ± 0.19
4	6.46 ± 0.47	10.74 ± 0.16	15.54 ± 0.21
5	6.57 ± 0.42	10.54 ± 0.14	15.51 ± 0.19
6	6.70 ± 0.35	10.57 ± 0.12	15.50 ± 0.18
7	6.77 ± 0.37	10.36 ± 0.11	15.50 ± 0.19
8	6.79 ± 0.30	10.35 ± 0.10	15.49 ± 0.19
9	6.85 ± 0.31	10.14 ± 0.10	15.48 ± 0.20
10	6.85 ± 0.32	10.13 ± 0.09	15.46 ± 0.19
11	6.94 ± 0.25	9.92 ± 0.08	15.46 ± 0.20
12	6.93 ± 0.26	9.90 ± 0.08	15.43 ± 0.21
13	6.97 ± 0.26	9.68 ± 0.09	15.41 ± 0.20
14	7.00 ± 0.23	9.67 ± 0.08	15.41 ± 0.20
15	7.00 ± 0.24	9.45 ± 0.08	15.40 ± 0.21
16	7.06 ± 0.19	9.44 ± 0.07	15.39 ± 0.22
17	7.09 ± 0.20	9.23 ± 0.07	15.38 ± 0.23
18	7.09 ± 0.20	9.20 ± 0.08	15.32 ± 0.24
19	7.11 ± 0.18	8.98 ± 0.08	15.29 ± 0.24
20	7.14 ± 0.14	8.95 ± 0.08	15.28 ± 0.26
21	7.14 ± 0.15	8.75 ± 0.08	15.23 ± 0.27
22	7.17 ± 0.12	8.72 ± 0.07	15.18 ± 0.28
23	7.18 ± 0.09	8.51 ± 0.07	15.09 ± 0.35
24	7.19 ± 0.09	8.47 ± 0.07	14.98 ± 0.43
25	7.20 ± 0.05	8.27 ± 0.07	14.55 ± 0.75
26	7.21 ± 0.05	8.23 ± 0.07	13.96 ± 1.10
27	7.22 ± 0.06	8.05 ± 0.07	10.78 ± 1.01
28	7.21 ± 0.06	8.02 ± 0.07	10.34 ± 0.83
29	7.22 ± 0.06	7.99 ± 0.06	10.03 ± 0.71
30	7.22 ± 0.06	7.98 ± 0.07	9.85 ± 0.66
31	7.22 ± 0.06	7.97 ± 0.06	9.71 ± 0.61
32	7.22 ± 0.06	7.96 ± 0.06	9.60 ± 0.54
33	7.21 ± 0.06	7.95 ± 0.06	9.49 ± 0.51
34	7.22 ± 0.06	7.95 ± 0.06	9.41 ± 0.47
35	7.21 ± 0.06	7.94 ± 0.06	9.34 ± 0.46

Table 3.3: Average and standard deviation of $\hat{\sigma}_\ell^{\min}$, $\hat{\sigma}_\ell^{\text{avg}}$, and $\hat{\sigma}_\ell^{\max}$, taken over 300 test sets with 1 run each, of scenario (iv).

direct effect on their neighbors only, we introduced the variable h that represents the hop distance to a faulty node. Using this variable, we could exclude every node which is reachable within at most h hops from a faulty node, along directed links, from the skew calculations: As it is certain that the skews near faulty nodes diverge more from the average, it is of interest how fast the remaining grid recovers from these effects.

The data provided in the following tables are the averages and the standard deviations of the stated operations. For the intra-layer skew, average and maximum are provided, for the inter-layer skew, also the minimum. Due to the unknown distribution of the skews, the average with the standard deviation would at most provide a feeling for the probability mass around the average. To allow a better understanding of the distribution of the skews, we also provide the 5% and 95% quantile of the skews. These two values give a perspective on the concentration of the respective distribution, which is, as we will see, more interesting than the distribution itself.

Byzantine Faults In Tables 3.4 and 3.5, the results for scenario (iii) resp. (iv) under isolated Byzantine faults are presented. The average values of both intra- and inter-layer skew are rather stable with respect to the increase of h, which is, considering the large number of 2500 nodes, not surprising. A noticeable improvement is only visible for relatively large ($f \geq 10$) numbers of faulty nodes. For $\hat{\sigma}^{\max}$, we see an immediate improvement from $h = 0$ to $h = 1$, which was expected, as faulty neighbors can easily increase the skew. In scenario (iv), the behavior of $\hat{\sigma}^{\max}$ is fairly constant with further increments of h, whereas scenario (iii) tends towards minor improvements. For relatively small f (< 10), the behavior of $\hat{\sigma}^{\min}$ is reasonably independent of the scenario, as we see generally a more stable behavior for small f. Nonetheless, under scenario (iv), the skew improvement, with increasing h, is more prominent, which is not surprising considering the large difference between q_5 and q_{95} (see below).

These observations support the assumption that a faulty node strongly influences its direct neighbors, only.

We now take a closer look at the quantiles, which are generally more stable. $\hat{q}_5$ is almost independent of h, except in scenario (iv) with $f \geq 5$ faults, where the values vary marginally and suffer from an increasing standard deviation. As $\hat{\sigma}^{\min}$ also has a higher variation in these cases, this is most likely caused by the clock source skews coupled with the power of the Byzantine faults. Surprisingly, $\hat{q}_{95}$ shows a different behavior than $\hat{q}_5$: While the standard deviation is higher for all settings, under scenario (iv), the values themselves only show marginal dependence on f and h compared to those of scenario (iii). A possible explanation for the phenomenon may be the fact that the large layer 0 skews in scenario (iv) are already sufficient to cause some neighbors to exhibit the associated worst-case skews, whereas additional faulty nodes are required in scenario (iii) to generate such large skews. The behavior of q_5 and q_{95} are similar to those of $\hat{q}_5$ and $\hat{q}_{95}$ and are thus not elaborated further.

Generally, the quantiles show a rather concentrated distribution of the skews around their average, with high probability mass for higher skews. Under scenario (iv), this asymmetry is more prominent, as $\hat{q}_{95}$ is closer to $\hat{\sigma}^{\max}$ than $\hat{\sigma}^{\text{avg}}$ in most cases; the same holds for the intra-layer skews.

f	h	q_5	σ^{avg}	q_{95}	$\sigma^{\max}$	$\hat{\sigma}^{\min}$	$\hat{q}_5$	$\hat{\sigma}^{\text{avg}}$	$\hat{q}_{95}$	$\hat{\sigma}^{\max}$
1	0	0.03 ± 0.00	0.54 ± 0.05	1.47 ± 0.22	7.78 ± 0.91	6.95 ± 0.36	7.32 ± 0.01	7.98 ± 0.03	8.81 ± 0.09	15.58 ± 1.00
1	1	0.03 ± 0.00	0.54 ± 0.05	1.46 ± 0.21	7.36 ± 0.56	7.12 ± 0.06	7.32 ± 0.01	7.98 ± 0.02	8.80 ± 0.09	15.15 ± 0.61
1	2	0.03 ± 0.00	0.53 ± 0.05	1.45 ± 0.21	7.22 ± 0.56	7.13 ± 0.03	7.32 ± 0.01	7.97 ± 0.02	8.80 ± 0.09	15.00 ± 0.61
1	3	0.03 ± 0.00	0.53 ± 0.05	1.44 ± 0.20	7.11 ± 0.56	7.13 ± 0.02	7.32 ± 0.01	7.97 ± 0.02	8.80 ± 0.09	14.90 ± 0.60
2	0	0.03 ± 0.00	0.60 ± 0.07	1.74 ± 0.38	8.15 ± 0.87	6.75 ± 0.79	7.31 ± 0.01	8.00 ± 0.03	8.90 ± 0.13	16.09 ± 1.11
2	1	0.03 ± 0.00	0.59 ± 0.07	1.71 ± 0.35	7.56 ± 0.48	7.12 ± 0.07	7.32 ± 0.01	8.00 ± 0.03	8.89 ± 0.13	15.39 ± 0.52
2	2	0.03 ± 0.00	0.58 ± 0.06	1.69 ± 0.34	7.36 ± 0.51	7.12 ± 0.06	7.32 ± 0.01	8.00 ± 0.03	8.89 ± 0.13	15.17 ± 0.56
2	3	0.03 ± 0.00	0.57 ± 0.06	1.66 ± 0.32	7.23 ± 0.52	7.13 ± 0.04	7.32 ± 0.01	7.99 ± 0.03	8.88 ± 0.12	15.01 ± 0.59
3	0	0.03 ± 0.00	0.65 ± 0.07	2.08 ± 0.55	8.48 ± 0.87	6.33 ± 1.36	7.31 ± 0.01	8.03 ± 0.04	9.00 ± 0.17	16.47 ± 1.27
3	1	0.03 ± 0.00	0.64 ± 0.07	2.02 ± 0.50	7.73 ± 0.35	7.10 ± 0.12	7.31 ± 0.01	8.03 ± 0.03	8.99 ± 0.16	15.56 ± 0.32
3	2	0.03 ± 0.00	0.63 ± 0.07	1.95 ± 0.45	7.53 ± 0.39	7.12 ± 0.07	7.31 ± 0.01	8.02 ± 0.03	8.98 ± 0.16	15.35 ± 0.36
3	3	0.03 ± 0.00	0.62 ± 0.07	1.91 ± 0.42	7.38 ± 0.44	7.12 ± 0.07	7.31 ± 0.01	8.02 ± 0.03	8.96 ± 0.15	15.21 ± 0.41
5	0	0.04 ± 0.00	0.76 ± 0.10	2.99 ± 0.90	8.91 ± 1.01	5.54 ± 1.99	7.31 ± 0.01	8.09 ± 0.05	9.25 ± 0.30	17.13 ± 1.72
5	1	0.04 ± 0.00	0.74 ± 0.09	2.80 ± 0.82	7.86 ± 0.29	7.05 ± 0.21	7.31 ± 0.01	8.08 ± 0.05	9.21 ± 0.28	15.70 ± 0.28
5	2	0.04 ± 0.00	0.72 ± 0.09	2.64 ± 0.76	7.70 ± 0.30	7.10 ± 0.09	7.31 ± 0.01	8.07 ± 0.05	9.18 ± 0.26	15.53 ± 0.30
5	3	0.04 ± 0.00	0.70 ± 0.09	2.51 ± 0.70	7.59 ± 0.33	7.11 ± 0.07	7.31 ± 0.01	8.06 ± 0.04	9.15 ± 0.25	15.40 ± 0.33
10	0	0.04 ± 0.00	0.95 ± 0.12	4.73 ± 0.98	9.55 ± 1.18	3.62 ± 2.59	7.30 ± 0.01	8.18 ± 0.06	10.02 ± 0.67	18.85 ± 2.52
10	1	0.04 ± 0.00	0.91 ± 0.11	4.36 ± 0.98	8.01 ± 0.38	6.89 ± 0.54	7.30 ± 0.01	8.16 ± 0.06	9.87 ± 0.60	15.84 ± 0.17
10	2	0.04 ± 0.00	0.87 ± 0.11	3.99 ± 0.97	7.84 ± 0.17	7.05 ± 0.15	7.31 ± 0.01	8.14 ± 0.05	9.73 ± 0.53	15.63 ± 0.19
10	3	0.04 ± 0.00	0.84 ± 0.11	3.68 ± 0.94	7.75 ± 0.22	7.07 ± 0.14	7.31 ± 0.01	8.13 ± 0.05	9.63 ± 0.47	15.53 ± 0.22
20	0	0.04 ± 0.00	1.27 ± 0.13	6.57 ± 0.40	10.61 ± 1.46	0.65 ± 2.54	7.29 ± 0.01	8.34 ± 0.07	12.37 ± 0.89	21.29 ± 2.17
20	1	0.04 ± 0.00	1.18 ± 0.13	6.17 ± 0.51	8.45 ± 1.15	6.20 ± 1.51	7.30 ± 0.01	8.31 ± 0.07	11.84 ± 0.92	15.92 ± 0.15
20	2	0.04 ± 0.00	1.11 ± 0.13	5.76 ± 0.60	7.96 ± 0.12	6.96 ± 0.23	7.30 ± 0.01	8.28 ± 0.06	11.32 ± 0.91	15.75 ± 0.18
20	3	0.04 ± 0.01	1.05 ± 0.13	5.35 ± 0.70	7.88 ± 0.16	7.01 ± 0.17	7.30 ± 0.01	8.25 ± 0.06	10.90 ± 0.87	15.64 ± 0.19

Table 3.4: Average $\pm$ standard deviation of q_5, σ^{avg}, q_{95}, $\sigma^{\max}$, $\hat{\sigma}^{\min}$, $\hat{q}_5$, $\hat{\sigma}^{\text{avg}}$, $\hat{q}_{95}$, and $\hat{\sigma}^{\max}$, excluding all nodes with directed distance $\leq h$ from f randomly placed isolated Byzantine faulty nodes, over 300 test sets with 1 run each, of scenario (iii).

f	h	q_5	σ^{avg}	q_{95}	$\sigma^{\max}$	$\hat{\sigma}^{\min}$	$\hat{q}_5$	$\hat{\sigma}^{\text{avg}}$	$\hat{q}_{95}$	$\hat{\sigma}^{\max}$
1	0	0.04 ± 0.01	1.50 ± 0.24	7.54 ± 0.07	8.49 ± 1.39	2.45 ± 1.74	7.26 ± 0.05	8.40 ± 0.10	14.44 ± 0.10	16.40 ± 1.59
1	1	0.04 ± 0.01	1.50 ± 0.24	7.54 ± 0.07	8.27 ± 0.88	2.68 ± 1.05	7.26 ± 0.05	8.40 ± 0.10	14.43 ± 0.11	15.89 ± 0.13
1	2	0.04 ± 0.01	1.50 ± 0.24	7.54 ± 0.07	8.14 ± 0.10	2.74 ± 0.97	7.26 ± 0.05	8.40 ± 0.10	14.42 ± 0.11	15.89 ± 0.13
1	3	0.04 ± 0.01	1.49 ± 0.24	7.54 ± 0.07	8.14 ± 0.02	2.77 ± 0.95	7.26 ± 0.05	8.40 ± 0.10	14.41 ± 0.11	15.89 ± 0.13
2	0	0.04 ± 0.02	1.58 ± 0.31	7.55 ± 0.08	9.30 ± 2.78	1.89 ± 2.64	7.25 ± 0.05	8.44 ± 0.14	14.47 ± 0.12	17.11 ± 2.28
2	1	0.04 ± 0.02	1.57 ± 0.31	7.55 ± 0.08	8.53 ± 1.61	2.49 ± 1.33	7.25 ± 0.05	8.43 ± 0.14	14.46 ± 0.12	15.91 ± 0.12
2	2	0.04 ± 0.02	1.56 ± 0.31	7.55 ± 0.08	8.17 ± 0.34	2.69 ± 1.08	7.25 ± 0.05	8.43 ± 0.14	14.45 ± 0.12	15.89 ± 0.12
2	3	0.04 ± 0.02	1.56 ± 0.31	7.55 ± 0.08	8.14 ± 0.04	2.75 ± 1.02	7.26 ± 0.05	8.43 ± 0.14	14.44 ± 0.14	15.89 ± 0.12
3	0	0.04 ± 0.01	1.62 ± 0.28	7.56 ± 0.07	10.03 ± 3.52	1.13 ± 3.32	7.25 ± 0.06	8.45 ± 0.13	14.49 ± 0.11	17.86 ± 2.74
3	1	0.04 ± 0.01	1.61 ± 0.28	7.55 ± 0.07	8.90 ± 2.19	2.19 ± 1.50	7.25 ± 0.06	8.45 ± 0.12	14.48 ± 0.11	15.90 ± 0.13
3	2	0.04 ± 0.01	1.60 ± 0.28	7.54 ± 0.07	8.21 ± 0.40	2.51 ± 1.15	7.25 ± 0.07	8.44 ± 0.12	14.46 ± 0.12	15.89 ± 0.13
3	3	0.04 ± 0.01	1.59 ± 0.28	7.54 ± 0.07	8.14 ± 0.05	2.59 ± 1.09	7.25 ± 0.07	8.44 ± 0.13	14.44 ± 0.15	15.89 ± 0.13
5	0	0.05 ± 0.06	1.78 ± 0.52	7.59 ± 0.10	10.82 ± 3.92	0.24 ± 3.75	7.23 ± 0.15	8.53 ± 0.22	14.57 ± 0.16	18.75 ± 2.93
5	1	0.05 ± 0.06	1.76 ± 0.52	7.57 ± 0.10	9.25 ± 2.63	1.87 ± 1.72	7.23 ± 0.15	8.52 ± 0.22	14.55 ± 0.16	15.92 ± 0.13
5	2	0.05 ± 0.07	1.75 ± 0.53	7.57 ± 0.11	8.25 ± 0.54	2.37 ± 1.24	7.23 ± 0.15	8.51 ± 0.22	14.52 ± 0.17	15.89 ± 0.12
5	3	0.05 ± 0.07	1.74 ± 0.54	7.57 ± 0.11	8.14 ± 0.03	2.52 ± 1.18	7.23 ± 0.14	8.51 ± 0.23	14.50 ± 0.19	15.88 ± 0.12
10	0	0.06 ± 0.04	2.12 ± 0.72	7.60 ± 0.58	12.20 ± 4.68	-1.33 ± 4.21	7.15 ± 0.37	8.67 ± 0.29	14.64 ± 0.64	21.06 ± 2.65
10	1	0.06 ± 0.04	2.08 ± 0.72	7.57 ± 0.59	10.05 ± 3.32	1.10 ± 2.12	7.16 ± 0.32	8.66 ± 0.29	14.61 ± 0.64	15.92 ± 0.23
10	2	0.06 ± 0.04	2.05 ± 0.74	7.56 ± 0.60	8.37 ± 0.94	1.89 ± 1.56	7.17 ± 0.28	8.64 ± 0.30	14.58 ± 0.65	15.88 ± 0.26
10	3	0.06 ± 0.04	2.03 ± 0.77	7.56 ± 0.61	8.12 ± 0.32	2.12 ± 1.53	7.16 ± 0.28	8.63 ± 0.31	14.55 ± 0.65	15.86 ± 0.30
20	0	0.08 ± 0.14	2.58 ± 0.99	7.72 ± 0.20	16.28 ± 8.54	-4.12 ± 4.41	7.10 ± 0.33	8.87 ± 0.41	14.83 ± 0.47	22.66 ± 2.40
20	1	0.08 ± 0.15	2.52 ± 1.01	7.68 ± 0.24	12.79 ± 7.77	-0.08 ± 2.32	7.11 ± 0.31	8.85 ± 0.42	14.77 ± 0.50	16.09 ± 1.43
20	2	0.08 ± 0.16	2.47 ± 1.04	7.67 ± 0.28	9.47 ± 7.21	1.24 ± 1.71	7.11 ± 0.33	8.83 ± 0.44	14.72 ± 0.52	15.97 ± 1.33
20	3	0.09 ± 0.18	2.43 ± 1.07	7.65 ± 0.39	8.99 ± 7.17	1.65 ± 1.74	7.09 ± 0.48	8.81 ± 0.46	14.68 ± 0.55	15.87 ± 0.43

Table 3.5: Average $\pm$ standard deviation of q_5, σ^{avg}, q_{95}, $\sigma^{\max}$, $\hat{\sigma}^{\min}$, $\hat{q}_5$, $\hat{\sigma}^{\text{avg}}$, $\hat{q}_{95}$, and $\hat{\sigma}^{\max}$, excluding all nodes with directed distance $\leq h$ from f randomly placed isolated Byzantine faulty nodes, over 300 test sets with 1 run each, of scenario (iv).

f	h	q_5	σ^{avg}	q_{95}	$\sigma^{\max}$	$\hat{\sigma}^{\min}$	$\hat{q}_5$	$\hat{\sigma}^{\text{avg}}$	$\hat{q}_{95}$	$\hat{\sigma}^{\max}$
1	0	0.03 ± 0.00	0.57 ± 0.04	1.55 ± 0.18	7.83 ± 0.26	7.10 ± 0.11	7.32 ± 0.01	7.99 ± 0.02	8.84 ± 0.08	15.88 ± 0.58
1	1	0.03 ± 0.00	0.56 ± 0.04	1.54 ± 0.18	7.57 ± 0.31	7.13 ± 0.04	7.32 ± 0.01	7.99 ± 0.02	8.84 ± 0.08	15.45 ± 0.31
1	2	0.03 ± 0.00	0.56 ± 0.04	1.53 ± 0.17	7.40 ± 0.38	7.13 ± 0.03	7.32 ± 0.01	7.99 ± 0.02	8.83 ± 0.08	15.25 ± 0.37
1	3	0.03 ± 0.00	0.55 ± 0.04	1.52 ± 0.17	7.26 ± 0.44	7.13 ± 0.02	7.32 ± 0.01	7.98 ± 0.02	8.83 ± 0.08	15.12 ± 0.42
2	0	0.03 ± 0.00	0.65 ± 0.06	2.02 ± 0.40	7.97 ± 0.15	7.05 ± 0.16	7.31 ± 0.01	8.03 ± 0.03	9.00 ± 0.14	16.31 ± 1.02
2	1	0.03 ± 0.00	0.64 ± 0.06	1.98 ± 0.39	7.71 ± 0.22	7.11 ± 0.09	7.31 ± 0.01	8.03 ± 0.03	8.98 ± 0.14	15.58 ± 0.22
2	2	0.03 ± 0.00	0.63 ± 0.06	1.94 ± 0.37	7.55 ± 0.30	7.12 ± 0.06	7.31 ± 0.01	8.02 ± 0.03	8.97 ± 0.14	15.40 ± 0.26
2	3	0.03 ± 0.00	0.62 ± 0.05	1.89 ± 0.36	7.43 ± 0.36	7.12 ± 0.06	7.31 ± 0.01	8.02 ± 0.03	8.96 ± 0.13	15.25 ± 0.33
3	0	0.04 ± 0.00	0.73 ± 0.07	2.64 ± 0.62	8.05 ± 0.21	6.99 ± 0.22	7.31 ± 0.01	8.07 ± 0.03	9.18 ± 0.21	16.62 ± 1.25
3	1	0.04 ± 0.00	0.71 ± 0.07	2.53 ± 0.58	7.83 ± 0.17	7.07 ± 0.14	7.31 ± 0.01	8.07 ± 0.03	9.16 ± 0.20	15.68 ± 0.19
3	2	0.04 ± 0.00	0.69 ± 0.06	2.42 ± 0.54	7.69 ± 0.23	7.10 ± 0.11	7.31 ± 0.01	8.06 ± 0.03	9.13 ± 0.20	15.51 ± 0.23
3	3	0.04 ± 0.00	0.68 ± 0.06	2.33 ± 0.51	7.56 ± 0.30	7.11 ± 0.08	7.31 ± 0.01	8.05 ± 0.03	9.11 ± 0.19	15.38 ± 0.28
5	0	0.04 ± 0.00	0.85 ± 0.09	3.85 ± 0.84	8.13 ± 0.43	6.87 ± 0.46	7.31 ± 0.01	8.14 ± 0.04	9.57 ± 0.38	17.42 ± 1.95
5	1	0.04 ± 0.00	0.82 ± 0.09	3.56 ± 0.82	7.92 ± 0.14	7.04 ± 0.15	7.31 ± 0.01	8.12 ± 0.04	9.51 ± 0.35	15.79 ± 0.19
5	2	0.04 ± 0.00	0.80 ± 0.09	3.31 ± 0.80	7.81 ± 0.19	7.08 ± 0.13	7.31 ± 0.01	8.11 ± 0.04	9.44 ± 0.33	15.61 ± 0.21
5	3	0.04 ± 0.00	0.77 ± 0.08	3.10 ± 0.77	7.71 ± 0.24	7.09 ± 0.12	7.31 ± 0.01	8.10 ± 0.04	9.39 ± 0.31	15.51 ± 0.23
10	0	0.04 ± 0.00	1.12 ± 0.12	5.98 ± 0.54	9.09 ± 1.96	5.95 ± 1.86	7.30 ± 0.01	8.28 ± 0.06	11.27 ± 0.86	19.62 ± 2.44
10	1	0.04 ± 0.00	1.07 ± 0.12	5.56 ± 0.65	8.04 ± 0.30	6.91 ± 0.33	7.30 ± 0.01	8.26 ± 0.06	10.89 ± 0.82	15.91 ± 0.16
10	2	0.04 ± 0.00	1.01 ± 0.11	5.14 ± 0.74	7.93 ± 0.13	6.99 ± 0.17	7.30 ± 0.01	8.23 ± 0.06	10.59 ± 0.76	15.74 ± 0.17
10	3	0.04 ± 0.01	0.97 ± 0.11	4.75 ± 0.82	7.87 ± 0.16	7.03 ± 0.16	7.30 ± 0.01	8.21 ± 0.06	10.35 ± 0.70	15.64 ± 0.20
20	0	0.05 ± 0.01	1.54 ± 0.14	7.13 ± 0.14	11.60 ± 3.50	3.49 ± 3.43	7.29 ± 0.01	8.51 ± 0.07	13.99 ± 0.38	22.22 ± 1.38
20	1	0.05 ± 0.01	1.42 ± 0.14	6.82 ± 0.24	8.42 ± 1.11	6.48 ± 1.07	7.29 ± 0.01	8.46 ± 0.07	13.52 ± 0.51	15.99 ± 0.12
20	2	0.04 ± 0.01	1.33 ± 0.14	6.51 ± 0.33	8.03 ± 0.10	6.87 ± 0.21	7.30 ± 0.01	8.41 ± 0.07	12.95 ± 0.68	15.82 ± 0.15
20	3	0.04 ± 0.01	1.24 ± 0.14	6.18 ± 0.45	7.97 ± 0.12	6.93 ± 0.20	7.30 ± 0.01	8.37 ± 0.07	12.39 ± 0.81	15.73 ± 0.15

Table 3.6: Average $\pm$ standard deviation of q_5, σ^{avg}, q_{95}, $\sigma^{\max}$, $\hat{\sigma}^{\min}$, $\hat{q}_5$, $\hat{\sigma}^{\text{avg}}$, $\hat{q}_{95}$, and $\hat{\sigma}^{\max}$, excluding all nodes with directed distance $\leq h$ from f randomly placed isolated fail-silent nodes, over 300 test sets with 1 run each, of scenario (iii).

f	h	q_5	σ^{avg}	q_{95}	$\sigma^{\max}$	$\hat{\sigma}^{\min}$	$\hat{q}_5$	$\hat{\sigma}^{\text{avg}}$	$\hat{q}_{95}$	$\hat{\sigma}^{\max}$
1	0	0.04 ± 0.00	1.51 ± 0.05	7.54 ± 0.05	9.51 ± 4.02	1.95 ± 2.98	7.26 ± 0.01	8.41 ± 0.02	14.44 ± 0.07	16.70 ± 1.97
1	1	0.04 ± 0.00	1.50 ± 0.05	7.54 ± 0.05	8.71 ± 1.92	2.63 ± 1.21	7.26 ± 0.01	8.40 ± 0.02	14.43 ± 0.07	15.90 ± 0.13
1	2	0.04 ± 0.00	1.49 ± 0.05	7.54 ± 0.05	8.17 ± 0.26	2.89 ± 0.84	7.26 ± 0.01	8.40 ± 0.02	14.42 ± 0.08	15.89 ± 0.12
1	3	0.04 ± 0.00	1.49 ± 0.05	7.53 ± 0.05	8.14 ± 0.02	2.89 ± 0.84	7.26 ± 0.01	8.40 ± 0.03	14.41 ± 0.09	15.89 ± 0.12
2	0	0.04 ± 0.00	1.58 ± 0.06	7.55 ± 0.05	10.61 ± 5.27	1.22 ± 3.85	7.26 ± 0.01	8.44 ± 0.03	14.49 ± 0.06	17.65 ± 2.60
2	1	0.04 ± 0.00	1.57 ± 0.07	7.55 ± 0.05	9.20 ± 2.58	2.36 ± 1.65	7.26 ± 0.01	8.44 ± 0.03	14.48 ± 0.07	15.92 ± 0.12
2	2	0.04 ± 0.00	1.56 ± 0.07	7.54 ± 0.05	8.26 ± 0.57	2.90 ± 0.86	7.26 ± 0.01	8.43 ± 0.03	14.46 ± 0.08	15.91 ± 0.12
2	3	0.04 ± 0.00	1.55 ± 0.07	7.54 ± 0.05	8.14 ± 0.02	2.90 ± 0.86	7.26 ± 0.01	8.43 ± 0.04	14.44 ± 0.10	15.90 ± 0.12
3	0	0.04 ± 0.01	1.65 ± 0.08	7.56 ± 0.04	12.73 ± 6.58	-0.29 ± 4.74	7.26 ± 0.01	8.48 ± 0.04	14.53 ± 0.06	18.67 ± 2.97
3	1	0.04 ± 0.01	1.63 ± 0.08	7.55 ± 0.04	10.13 ± 3.31	1.86 ± 1.92	7.26 ± 0.01	8.47 ± 0.04	14.51 ± 0.07	15.92 ± 0.12
3	2	0.04 ± 0.01	1.62 ± 0.08	7.54 ± 0.05	8.35 ± 0.75	2.82 ± 0.81	7.26 ± 0.01	8.46 ± 0.04	14.48 ± 0.08	15.90 ± 0.12
3	3	0.04 ± 0.01	1.60 ± 0.08	7.54 ± 0.05	8.14 ± 0.02	2.85 ± 0.82	7.26 ± 0.01	8.46 ± 0.04	14.46 ± 0.11	15.89 ± 0.13
5	0	0.04 ± 0.01	1.77 ± 0.09	7.58 ± 0.04	14.46 ± 6.75	-1.25 ± 5.08	7.25 ± 0.01	8.54 ± 0.05	14.60 ± 0.06	19.98 ± 2.95
5	1	0.04 ± 0.01	1.74 ± 0.09	7.56 ± 0.04	10.74 ± 3.56	1.66 ± 2.14	7.26 ± 0.01	8.53 ± 0.05	14.57 ± 0.07	15.94 ± 0.12
5	2	0.04 ± 0.01	1.71 ± 0.10	7.55 ± 0.04	8.42 ± 0.92	2.88 ± 0.85	7.26 ± 0.01	8.51 ± 0.05	14.53 ± 0.08	15.90 ± 0.12
5	3	0.04 ± 0.01	1.69 ± 0.10	7.55 ± 0.05	8.14 ± 0.02	2.92 ± 0.85	7.26 ± 0.01	8.50 ± 0.05	14.50 ± 0.12	15.89 ± 0.12
10	0	0.05 ± 0.01	2.02 ± 0.13	7.62 ± 0.04	18.38 ± 6.47	-4.03 ± 5.21	7.25 ± 0.01	8.67 ± 0.06	14.73 ± 0.06	22.04 ± 2.09
10	1	0.05 ± 0.01	1.96 ± 0.13	7.59 ± 0.04	12.68 ± 3.92	0.59 ± 2.61	7.25 ± 0.01	8.64 ± 0.07	14.68 ± 0.07	15.97 ± 0.12
10	2	0.05 ± 0.01	1.91 ± 0.14	7.57 ± 0.05	8.74 ± 1.33	2.92 ± 1.00	7.25 ± 0.01	8.62 ± 0.07	14.62 ± 0.09	15.91 ± 0.11
10	3	0.05 ± 0.01	1.87 ± 0.15	7.57 ± 0.05	8.15 ± 0.22	3.03 ± 0.94	7.25 ± 0.01	8.60 ± 0.08	14.58 ± 0.13	15.89 ± 0.11
20	0	0.05 ± 0.01	2.40 ± 0.16	7.71 ± 0.04	22.58 ± 4.20	-7.39 ± 3.60	7.24 ± 0.01	8.87 ± 0.08	14.91 ± 0.05	23.19 ± 1.16
20	1	0.05 ± 0.01	2.29 ± 0.16	7.64 ± 0.04	15.26 ± 3.03	-1.08 ± 2.66	7.24 ± 0.01	8.83 ± 0.08	14.83 ± 0.06	16.03 ± 0.11
20	2	0.05 ± 0.01	2.20 ± 0.17	7.61 ± 0.05	9.49 ± 1.85	2.81 ± 0.97	7.25 ± 0.01	8.78 ± 0.09	14.75 ± 0.08	15.92 ± 0.11
20	3	0.05 ± 0.01	2.12 ± 0.19	7.61 ± 0.06	8.15 ± 0.15	3.11 ± 0.91	7.25 ± 0.01	8.74 ± 0.10	14.69 ± 0.12	15.89 ± 0.12

Table 3.7: Average $\pm$ standard deviation of q_5, σ^{avg}, q_{95}, $\sigma^{\max}$, $\hat{\sigma}^{\min}$, $\hat{q}_5$, $\hat{\sigma}^{\text{avg}}$, $\hat{q}_{95}$, and $\hat{\sigma}^{\max}$, excluding all nodes with directed distance $\leq h$ from f randomly placed isolated fail-silent nodes, over 300 test sets with 1 run each, of scenario (iv).

Fail-Silent Faults In Tables 3.6 and 3.7, the results for scenario (iii) resp. (iv) under isolated fail-silent faults are presented. The behavior of the grid under fail-silent faults is similar to that with Byzantine faults w.r.t. f and h, but there are differences in the value range.

Under scenario (iii), we can observe higher values for the intra-layer skews than in Table 3.4. The inter-layer skews are similar for small f, but for larger f, the values are also higher than with Byzantine faults. The major reason for this counter-intuitive discrepancy is the fact that our Byzantine faulty nodes do not slow down pulse propagation as much as fail-silent nodes: As our Byzantine nodes can send messages with $d^+ = 0$ ns, they can cause speed-ups, which reduces the chance of skews to accumulate (cf. Lemma 2.3.7), unlike in the case of fail-silent faults. Hence, a neighbor of a fail-silent node could trigger later than a neighbor of a Byzantine node.

Under scenario (iv), the intra-layer skew shows a lower standard deviation, but with similar average values. An exception to this is $\sigma^{\max}$, which has higher values and a larger standard deviation. For the inter-layer skews, we see a lower standard deviation for the quantiles and averages with similar values, and a $\hat{\sigma}^{\max}$ that is slightly higher but more stable than the results with Byzantine faults. By contrast, $\hat{\sigma}^{\min}$ is smaller for fail-silent nodes, which is probably due to the fact that large clock source skews in this scenario are not masked by speed-ups caused by the Byzantine faults.

As fail-silent faults can only slow down the system, the difference to the Byzantine faults observed were to be expected. Besides the above differences, we still see the same properties as before: the effects of the faults are local and the skew distribution is concentrated around the average, especially for small f.

3.3.3 Self-Stabilization Evaluations

In order to evaluate stabilization times, it is important to define when a system is stable. In the case of the HEX grid, this is rather simple: Every non-faulty node fires exactly once per pulse and the skews have to be within the proven bounds. The first property requires that the 1–local fault hypothesis is not violated. The second property is more problematic, since the proven bounds hold only for the fault-free case. Thus, we cannot expect that a system with faulty nodes will oblige to the bounds of Theorem 2.3.10, especially during stabilization. Consequently, determining stabilization times was also done with several increased bounds, which are interesting in settings with many faults (where the proven bound would erroneously result in no stabilization or unsatisfactory stabilization times).

As for evaluating the skews, the HEX grid used for these simulations also consists of 100 layers and 25 columns. The link wire delays were chosen uniformly distributed in the interval of $[7, 8]$ ns. Adding the processing delays of the HEX node implementation, mentioned in Section 3.1.4, the end-to-end link delays $[d^-, d^+]$ are in the interval of $[7.125, 8.165]$ ns. The simulation results were acquired with 100 test sets consisting of 10 runs each. In each test case, the link wire delays were randomly chosen in the above mentioned interval, and varied after every run. The layer 0 clock skews were randomly chosen once for every test set. The placement of the faulty nodes, and their fault-type, was also randomly chosen once for each test set. The Byzantine faulty nodes behave, on a per-link basis, either fail-silent or continuously send a (fast) trigger message.

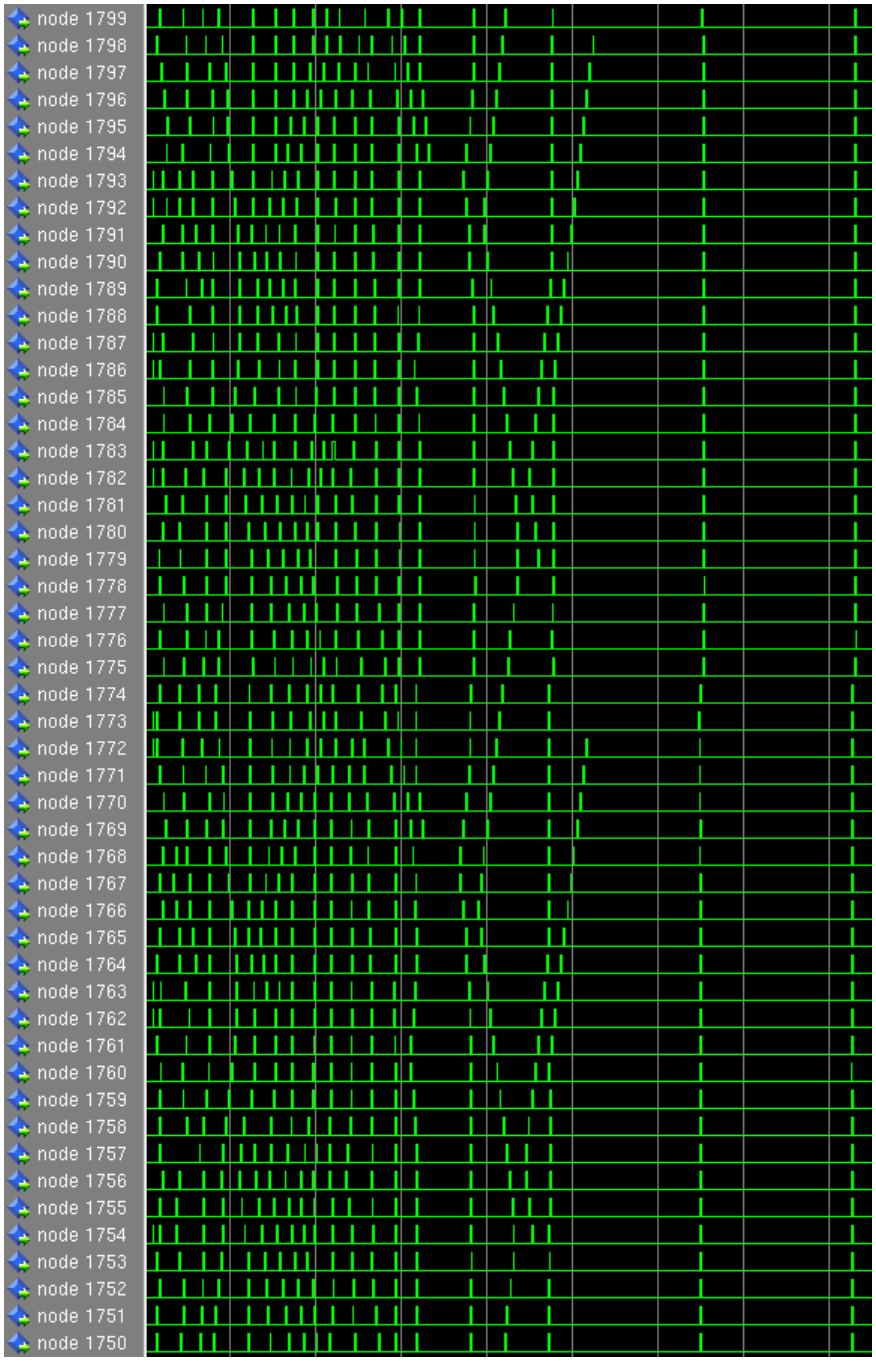


Figure 3.21: A screenshot of the stabilization phase of a HEX grid with $L = 100$ and $W = 25$. On the left, we see chaotic firing due to the induced random state in the HEX nodes. On the right two valid pulses are already propagating through the network.
Note the minor difference in triggering time between node 1774 and 1775 (after stabilization): This is due to the inter-layer skew between the layers in which these two nodes reside.

		Default Inter-Layer Skew Bound				d^+ Inter-Layer Skew Bound			$2d^+$ Inter-Layer Skew Bound		
f	h	S^{avg}	$S^{\max}$	$\Sigma(S)$	$\Gamma^{\min}$	S^{avg}	$S^{\max}$	$\Sigma(S)$	S^{avg}	$S^{\max}$	$\Sigma(S)$
0	0	1.81 ± 0.42	3	100	177.40	1.81 ± 0.42	3	100	1.81 ± 0.42	3	100
1	0	1.70 ± 0.47	2	27	182.85	1.77 ± 0.42	2	100	1.77 ± 0.42	2	100
1	1	1.78 ± 0.51	3	27	182.85	1.77 ± 0.42	2	100	1.77 ± 0.42	2	100
1	2	2.28 ± 1.83	10	29	182.85	1.77 ± 0.42	2	100	1.77 ± 0.42	2	100
1	3	2.80 ± 2.34	10	30	182.85	1.77 ± 0.42	2	100	1.77 ± 0.42	2	100
5	0	2.00 ± 0.00	2	1	213.74	1.83 ± 0.41	3	99	1.83 ± 0.41	3	99
5	1	2.00 ± 0.00	2	1	214.41	1.83 ± 0.40	3	100	1.83 ± 0.40	3	100
5	2	2.00 ± 0.00	2	1	214.41	1.83 ± 0.40	3	100	1.83 ± 0.40	3	100
5	3	7.00 ± 0.00	7	1	214.41	1.83 ± 0.40	3	100	1.83 ± 0.40	3	100
10	0	0.00 ± 0.00	$\times$	0	175.35	1.93 ± 0.97	9	99	1.93 ± 0.97	9	99
10	1	0.00 ± 0.00	$\times$	0	175.35	1.80 ± 0.40	2	100	1.80 ± 0.40	2	100
10	2	0.00 ± 0.00	$\times$	0	175.35	1.80 ± 0.40	2	100	1.80 ± 0.40	2	100
10	3	0.00 ± 0.00	$\times$	0	175.35	1.80 ± 0.40	2	100	1.80 ± 0.40	2	100
20	0	0.00 ± 0.00	$\times$	0	200.32	2.24 ± 1.63	10	86	2.24 ± 1.63	10	86
20	1	0.00 ± 0.00	$\times$	0	200.32	1.82 ± 0.41	3	100	1.82 ± 0.41	3	100
20	2	0.00 ± 0.00	$\times$	0	200.32	1.82 ± 0.41	3	100	1.82 ± 0.41	3	100
20	3	0.00 ± 0.00	$\times$	0	200.32	1.82 ± 0.41	3	100	1.82 ± 0.41	3	100

Table 3.8: Average and standard deviation of the stabilization time S^{avg}, the maximal stabilization time $S^{\max}$, and the minimal pulse separation time $\Gamma^{\min}$. If the system stabilized under the given setting, the number of stabilized runs $\Sigma(S)$ is also given. The test sets used to compile this list where made of 10 runs and f randomly placed isolated Byzantine faults; for determining the stabilization time, all HEX nodes within a directed distance of h from every fault were excluded and the inter-layer skew bound stated in the heading was applied. For every setting, 100 test sets under scenario (iii) were conducted.

		Default Inter-Layer Skew Bound				d^+ Inter-Layer Skew Bound			$2d^+$ Inter-Layer Skew Bound		
f	h	S^{avg}	$S^{\max}$	$\Sigma(S)$	$\Gamma^{\min}$	S^{avg}	$S^{\max}$	$\Sigma(S)$	S^{avg}	$S^{\max}$	$\Sigma(S)$
0	0	1.81 ± 0.42	3	100	177.40	1.81 ± 0.42	3	100	1.81 ± 0.42	3	100
1	0	2.00 ± 0.00	2	1	182.85	1.77 ± 0.42	2	100	1.77 ± 0.42	2	100
1	1	2.00 ± 0.00	2	1	182.85	1.77 ± 0.42	2	100	1.77 ± 0.42	2	100
1	2	2.00 ± 0.00	2	3	182.85	1.77 ± 0.42	2	100	1.77 ± 0.42	2	100
1	3	4.00 ± 4.00	10	4	182.85	1.77 ± 0.42	2	100	1.77 ± 0.42	2	100
5	0	0.00 ± 0.00	$\times$	0	158.52	1.81 ± 0.42	3	100	1.81 ± 0.42	3	100
5	1	0.00 ± 0.00	$\times$	0	158.52	1.81 ± 0.42	3	100	1.81 ± 0.42	3	100
5	2	0.00 ± 0.00	$\times$	0	158.52	1.81 ± 0.42	3	100	1.81 ± 0.42	3	100
5	3	0.00 ± 0.00	$\times$	0	158.52	1.81 ± 0.42	3	100	1.81 ± 0.42	3	100
10	0	0.00 ± 0.00	$\times$	0	181.54	1.83 ± 0.40	3	100	1.83 ± 0.40	3	100
10	1	0.00 ± 0.00	$\times$	0	181.54	1.82 ± 0.41	3	100	1.82 ± 0.41	3	100
10	2	0.00 ± 0.00	$\times$	0	181.54	1.82 ± 0.41	3	100	1.82 ± 0.41	3	100
10	3	0.00 ± 0.00	$\times$	0	181.54	1.81 ± 0.42	3	100	1.81 ± 0.42	3	100
20	0	0.00 ± 0.00	$\times$	0	185.53	1.88 ± 0.48	3	99	1.88 ± 0.48	3	99
20	1	0.00 ± 0.00	$\times$	0	185.53	1.87 ± 0.49	3	100	1.87 ± 0.49	3	100
20	2	0.00 ± 0.00	$\times$	0	185.53	1.85 ± 0.48	3	100	1.85 ± 0.48	3	100
20	3	0.00 ± 0.00	$\times$	0	185.53	1.85 ± 0.48	3	100	1.85 ± 0.48	3	100

Table 3.9: Average and standard deviation of the stabilization time S^{avg}, the maximal stabilization time $S^{\max}$, and the minimal pulse separation time $\Gamma^{\min}$. If the system stabilized under the given setting, the number of stabilized runs $\Sigma(S)$ is also given. The test sets used to compile this list where made of 10 runs and f randomly placed isolated fail-silent faults; for determining the stabilization time, all HEX nodes within a directed distance of h from every fault were excluded and the inter-layer skew bound stated in the heading was applied. For every setting, 100 test sets under scenario (iii) were conducted.

To provide a general idea of how the stabilization process looks like in a HEX grid, Figure 3.21 provides a screenshot of a ModelSim simulation during the stabilization.

Tables 3.8 and 3.9 show the results in a setting similar to the single pulse scenario (iii) used in Section 3.3.2.3. The difference to the previous simulations is that, in each execution of a test set, the nodes were put into a random initial state before the 10 pulses per test set were generated by the clock sources. Again, the parameter h is used to exempt faults and their direct neighborhood, in the analysis.

The tables provide the average, the standard deviation and the maximum of the stabilization time S (including the first stable pulse), the number of runs $\Sigma(S)$ that stabilized, and the minimal gap $\Gamma^{\min}$ between two consecutive pulses of any node after stabilization.

Note that Lemma 2.5.2 predicts a worst-case stabilization time of 100 pulses in our setting, thus runs where we observed no stabilization would just need more simulated pulses,[5] provided that faults in the grid do not prohibit stabilization at all. One instance of this can be seen in the case of $f = 20$ in Table 3.9, where all but one test set stabilized within 3 pulses. In the case of Byzantine faults in Table 3.8, the longest stabilization time was 10, and 14 sets did not stabilize within 10 pulses.

The first set of columns in the tables show the results obtained by using the proven skew bounds from Theorem 2.3.10 for determining stabilization. Note that in this set the observed maximal intra-layer skew, rather than the theoretical worst-case skew bound, has been used to determine the inter-layer skew bound employed for checking stabilization. The middle set used d^+ as the inter-layer skew bound, which, combined with $h > 0$, results in always stabilizing runs for every number of faults. Interestingly, a further increment of the skew bound to $2d^+$ (provided in the right column) provides no further improvement in terms of stabilization time or stabilized runs with $h = 0$. A possible explanation for this phenomenon may lie in the proven stabilization time of L pulses (cf. Lemma 2.5.2): Due to the large number of 100 layers, there may still be nodes in the upper layers that fire incorrectly due to faults after 10 pulses. This explanation is supported by the fact that the stabilization results for a skew bound of $3d^+$, which is not provided in the tables, are the same as those with a skew bound of $2d^+$.

It is interesting to note that in settings where every run of the system stabilized, the maximum number of pulses until the system stabilized was only 3 pulses, with an average close to 2 pulses.

The above simulations were done with a layer 0 pulse separation time of 339 ns, which was calculated based on Lemmas 2.5.1 and 2.5.2. The $\Gamma^{\min}$ given in the tables is the smallest time between two pulses of any correct node, also including nodes which have not stabilized yet. Thus, the minimal pulse separation time provides a guideline by how much the pulse separation time actually could be decreased: In principle, $\Gamma^{\min}$ could be as small as the maximum sleeping time T^+ plus a few d^+ as a safety margin to account for faults and the fact that the data was acquired by means of simulations. Since $T^+ \approx 29$ ns in our setting, it is apparent that scenarios that require a large pulse separation time are actually rare.

The reason why non-stabilized nodes were included in the calculation of $\Gamma^{\min}$ is a bit counterintuitive: As non-stabilized nodes can have excessive skews, their minimal pulse separation time can be smaller than the minimal pulse separation of correct nodes. Thus a reduction by $\Gamma^{\min}$

[5] Since multi-pulse simulations took considerable time, simulating more than 10 pulses per test set would have caused excessive simulation times.

calculated only for correct and stabilized nodes could result in non-stabilized nodes being asleep when the next pulse arrives. This can lead to an increase of the stabilization time and would also have a bad effect on applications using the HEX pulse, e.g., the high-frequency clock described in Chapter 4.

CHAPTER 4

High-Frequency Clock

HEX provides a well-synchronized clock pulse at every correct node in the grid, even in the presence of Byzantine faulty nodes and when started from an arbitrary initial state. To guarantee these fault-tolerance properties, and due to the grid structure in general, some minimal pulse separation time is needed. In conjunction with the large jitter of the generated HEX pulses, this considerably decreases the maximal possible frequency of the clock generation in layer 0. This chapter elaborates on techniques to extend the HEX node with local high-frequency clocks, dubbed *fast clocks*, which are synchronized to the HEX pulses.

4.1 Design Challenges

The "native" clock signal provided by the HEX nodes, through the HEX pulses, has a low frequency and a large jitter. The low frequency is caused by the pulse separation time which is required for the correct functionality and the self-stabilizing property of the HEX grid.

The large jitter stems from various sources, including the clock generation algorithm at layer 0, link delays, and faults in the HEX grid. According to Lemma 2.5.2, the link delays alone can produce a worst-case jitter as large as $\mathcal{O}(L\varepsilon)$, and the clock generation algorithm contributes the skew potential Δ_0. An important source of additional jitter are faults: Byzantine faults can decrease the time until a node triggers, albeit this is bounded by the link delay of the second trigger message. If a faulty nodes slows down or completely omits the sending of a trigger message to one of its upper layer neighbors, then the affected node requires a trigger message from its left or right neighbor. This however, involves an additional link and thereby may increase the triggering time by as much as d^+. Additionally, these effects accumulate, i.e., may increase the potential jitter with every layer.

Hence, providing a stable high-frequency clock while being fault-tolerant are conflicting targets in the current design of HEX. Increasing the node degree, such that every node has at least three neighbors in the lower layer, as well as in the upper layer, e.g., would decrease the capabilities of a fault and thus reduce the induced jitter. The price is increased complexity of both the grid structure and the HEX nodes, which is not always acceptable.

Overall, the design challenge here is to provide a high-frequency clock with reasonable skew at every node, without changing the underlying HEX grid. Our solution will employ a special kind of frequency multiplication for this purpose.

4.2 Design Requirements

The goal is to design a frequency multiplier, as shown in Figure 4.1, which takes the HEX clock as its input and provides a high-frequency clock as its output. The HEX clock has a low frequency, is well-synchronized to the neighboring nodes, and has a potentially large jitter. The high-frequency clock shall, obviously, have a higher frequency than the HEX clock. Furthermore, it shall also be synchronized to the neighbor nodes, in the sense that it has bounded precision π (recall Section 1.3.1).

4.3 Design Selection

As the fast clock must be synchronized to the HEX clock, classic clock generation schemes, e.g., free-running quartz oscillators, are unsuitable due to the missing synchronization capabilities. Thus, we have to resort to clock multiplication techniques like PLLs, which are typically used for such problems.

PLL: A PLL is a circuit, which uses a loopback to determine the difference of the phase between the generated output clock and the input clock. This difference is then used to adapt the output frequency. When the difference is zero, the PLL has locked on the input clock. By using clock dividers in the loopback and in the input signal path, different multiplication factors for the output clock can be achieved.

Unfortunately, this method for generating a high-frequency clock does not fit the above requirements for the fast clock. PLLs are sensitive to jitter on the input clock, as this causes phase variations that could prohibit the PLL from locking on the HEX clock, thus prohibiting the generation of a proper fast clock signal [63, 75]. Even if a specially designed PLL could withstand the jitter, a complete removal of the resulting frequency variations, while providing a stable frequency, seems unlikely. In fact, the challenge here is to maintain a strict relation between the input pulses and the output pulses, which is inevitable for guaranteeing bounded precision of the fast clocks: Typically, the continuous generation of a clock signal leads, over time, to a violation of the bounded synchrony requirement between neighboring nodes.

All Digital PLL: An ADPLL is a completely digitized version of a PLL, i.e., no analog circuit components are required. An ADPLL is typically less dependent on environmental factors, including input jitter.

Approaches similar to [50] promise a simple design with minimal area requirements, but they seem prone to metastability issues, especially with an unstable input clock like the HEX clock. Designs like [74] do not have these metastability issues, but try to lock on the input signal like a standard PLL, which may not succeed in the presence of large input jitter.

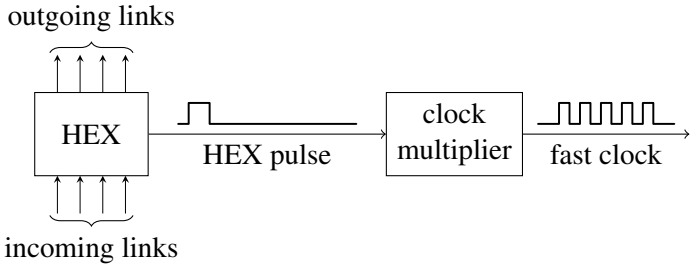


Figure 4.1: The composition of HEX with a clock multiplier. The HEX pulse generated from the HEX node is used by the clock to generate a fast clock.

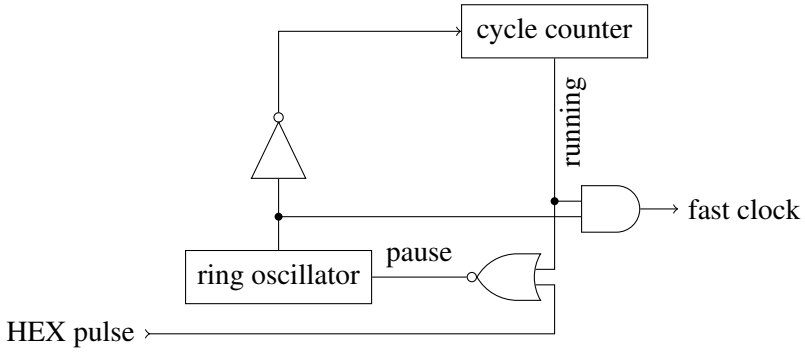


Figure 4.2: The logic elements building the fast clock. The HEX pulse enables the oscillator, which in turn drives the cycle counter. The cycle counter enables the output of the oscillator as the fast clock, for a predefined amount of clock cycles, and stops the oscillator after the burst has been completed.

Thus, like conventional PLLs, we do not consider ADPLLs suitable for building our fast clocks.

Digitally Controlled Clock Multiplier: A digitally controlled clock multiplier, simply put, generates a fixed amount of clock cycles on the output clock after every rising edge on the input clock, using a local oscillator.

Given that using a free-running solution has the same problem as a PLL, namely, the possible precision violation due to the continuous clock generation, a design like [51], which stops the clock generation after a fixed amount of cycles (a burst) is preferable.

In the following sections, we will show that this design fulfills all our design requirements for the fast clock, provided the *burst length* and the fast clock frequency are suitable chosen.

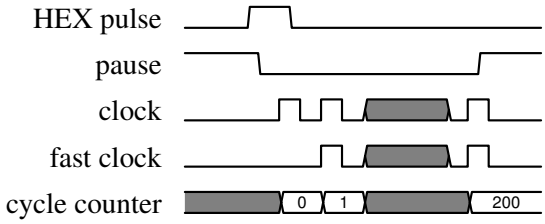


Figure 4.3: Timing diagram of the fast clock generated by the clock multiplier. After the HEX pulse is received, the pause signal is disabled and the ring oscillator starts generating a clock. This clock starts the cycle counter, which then enables the fast clock output. This is done until the predefined value of the cycle counter, in this case 200, is reached. Then, pause will be reenabled and the generation of the fast clock stops.

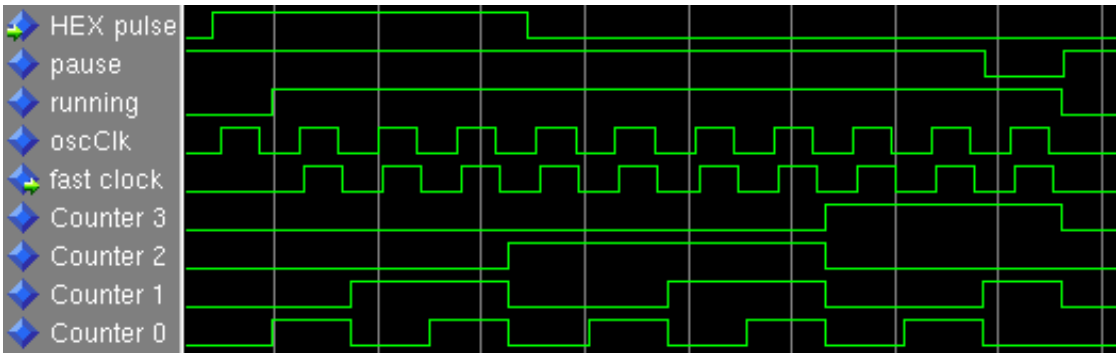


Figure 4.4: A screenshot of a simulation of the clock multiplier circuit generating the fast clock. On top is the HEX pulse signal, which disables the high-active pause signal of the ring oscillator. The cycle counter, whose activity is represented by the running signal, starts counting as soon as a clock by the oscillator is provided. The running signal holds the pause signal low and enables the output of the fast clock. Note that the cycle counter is running with the inverted clock of the oscillator, thus signal running goes high when oscClk is low. This is necessary to avoid shortened fast clock cycles. Below the fast clock, the bits of the cycle counter are shown.

4.4 Implementation

Based on the reasoning above, a digitally controlled clock multiplier was chosen for our fast clock. An overview of the interconnection between the HEX node and the clock multiplier is shown in Figure 4.1, a schematic of its implementation is shown in Figure 4.2.

The clock multiplier consists of a start/stoppable ring oscillator, a cycle counter, and a few logic gates. The output of the fast clock signal is enabled by the running signal, due to which the cycle counter can control the burst length. The ring oscillator, which uses the same design as described in Section 3.1.3, is started with the rising edge of a HEX pulse. It is stopped after the completion of the last cycle of the burst, which is determined by the count of the cycle counter: Starting from a counter value of 0, the running signal is enabled. When the predefined counter value, i.e., the burst length, is reached, running is disabled. Then, the counter is reset to 0 again.

Note that the HEX pulse must be wide enough so that the running signal is able to keep the

oscillator running before the HEX pulse's falling transition. See Figure 4.3 for a timing diagram of the clock multiplier, and Figure 4.4 for the result of a timing simulation.

The clock frequency of the ring oscillator, and thus of the fast clock, must be chosen so that the running signal of the cycle counter is stable after half the clock period. Otherwise, the first or last clock cycle of the fast clock could suffer from an incorrect timing that, in the worst case, could cause metastability or other faults in the circuit using the fast clock. Note that this is also the reason why the cycle counter runs with the inverted oscillator clock: Otherwise, the output of the cycle counter could enable the fast clock at some time in the high-phase of the clock cycle of the oscillator.

4.5 Analysis

The first question of interest is the amount of time available for the generation of a burst of the fast clock. This time is lower-bounded by the minimum time between two pulses of any non-faulty node in the grid, called the *minimal pulse separation time*

$$\Gamma^{\min} = \inf_{\substack{\ell\in[L+1],\ i\in[W],\ k\in\mathbb{N} \\ (\ell,i)\text{ correct}}} \left(t_{\ell,i}^{(k+1)} - t_{\ell,i}^{(k)}\right).$$

Assuming that the ring oscillators of all fast clocks run at least with a frequency of $f_{\min}$, then the number B of generated fast clock cycles per HEX pulse, the *burst length*, must satisfy

$$B \leq f_{\min}\Gamma^{\min}.$$

To be precise, the frequency $f_{\min}$ must only be the minimal *average* frequency over $\Gamma^{\min}$. To understand the performance implications resulting from the fact that the average pulse separation time is typically higher than the minimal pulse separation time, we will set those into perspective. Therefore, we assume $B = f_{\min}\Gamma^{\min}$ and consider for simplicity B to be an integer. The long-term average pulse separation time can be calculated as

$$\Gamma^{\text{avg}} = \lim_{k\to\infty} \frac{t_{(0,0)}^{(k)} - t_{(0,0)}^{(0)}}{k}.$$

Note that the node $(0,0)$ was chosen arbitrarily as a reference; any other node could be used as well: After all, the skew between $(0,0)$ and any other node (ℓ,i) is bounded by a function of the parameters $f(W,L,\varepsilon,\dots)$ only, but is independent of the pulse number k. Since

$$\lim_{k\to\infty} \frac{f(W,L,\varepsilon,\dots)}{k} = 0,$$

Γ^{avg} is the same for node (ℓ,i).

With the definition of the average pulse separation time, the amortized frequency of a node (ℓ,i) can be calculated as

$$\lim_{k\to\infty} \frac{Bk}{t_{(\ell,i)}^{(k)} - t_{(\ell,i)}^{(0)}} = B \lim_{k\to\infty} \frac{k}{t_{(\ell,i)}^{(k)} - t_{(\ell,i)}^{(0)}} = f_{\min}\frac{\Gamma^{\min}}{\Gamma^{\text{avg}}}.$$

With a perfect HEX clock, i.e., $t_{(\ell,i)}^{(k+2)} - t_{(\ell,i)}^{(k+1)} = t_{(\ell,i)}^{(k+1)} - t_{(\ell,i)}^{(k)} = \Gamma^{\min}$ for all i, k, and ℓ, this would result in the amortized frequency being equal to the minimal frequency. The loss of amortized frequency in case of $\Gamma^{\text{avg}} > \Gamma^{\min}$ is caused by three factors:

1. the frequency stability of the layer 0 clock generation.
2. the skew between the nodes on layer 0.
3. the variance of the pulse propagation speed, i.e., of the link delays.

While the first two are out-of-scope for this thesis, the latter is a property of the HEX grid. The first factor is likely to dominate the loss of frequency, however, as it requires a reduction of the minimal pulse separation time. The second and the third factor influence the skews in the system only, and thus have a limited (in fact, negligible) effect, as already mentioned.

We will now analyze the synchronization precision of our fast clocks, which is the maximum difference of the number of fast clock ticks generated by correct neighbors by time t. For simplicity, we define $L_{\ell,i}(t)$ as a mapping from real-time to a real-valued logical clock time in the range of $[0, B)$. As the HEX pulses are anonymous, the clock runs modulo B, i.e., wraps around from $B-1$ to 0. The clock frequency of the logical clock is bounded within $[f_{\min}, f_{\max}]$, which is the range of the real fast clock frequencies and not the amortized frequencies as in the analysis above. As the logical clock is real-valued, the actual (discrete) clock reading at time t is $\lfloor L_{\ell,i}(t) \rfloor$. With these definitions, the clock value for time $t \in \left[t_{\ell,i}^{(k)}, t_{\ell,i}^{(k+1)}\right]$ satisfies

$$\min\left(f_{\min}\left(t - t_{\ell,i}^{(k)}\right), B\right) \leq L_{\ell,i}(t) \leq \min\left(f_{\max}\left(t - t_{\ell,i}^{(k)}\right), B\right),$$

since $B \equiv 0$ when the clock is halted.

The maximal difference between two intra-layer neighbor nodes occurs when (i) they experience the maximal intra-layer skew between their respective HEX triggering time, and (ii) the HEX node which triggers first runs with $f_{\max}$, the other with $f_{\min}$, until they halt, cf. Figure 4.5. Thus, in the interval $t \in \left[t_{\ell,i}^{(k)}, t_{\ell,i}^{(k+1)}\right] \cap \left[t_{\ell,i+1}^{(k)}, t_{\ell,i+1}^{(k+1)}\right]$, the difference between the two logical clocks is

$$\begin{aligned} |L_{\ell,i}(t) - L_{\ell,i+1}(t)| &\leq B - f_{\min}\left(\frac{B}{f_{\max}} - \left|t_{\ell,i}^{(k)} - t_{\ell,i+1}^{(k)}\right|\right) \\ &\leq B - f_{\min}\frac{B}{f_{\max}} + f_{\min}\sigma_\ell \\ &\leq \frac{f_{\max} - f_{\min}}{f_{\max}} \cdot B + f_{\min}\sigma_\ell = \varrho B + f_{\min}\sigma_\ell, \end{aligned} \tag{4.1}$$

where ϱ is the relative drift of the fast clock, and σ_ℓ is the maximal intra-layer skew of layer ℓ.

Now consider the interval $t \in \left[t_{\ell,i}^{(k)}, t_{\ell,i}^{(k+1)}\right] \cap \left[t_{\ell,i+1}^{(k+1)}, t_{\ell,i+1}^{(k+2)}\right]$, where node (ℓ, i) has not triggered pulse $k+1$ yet. The worst-case is that the clock of node $(\ell, i+1)$ runs with $f_{\max}$, but as the clock is reset at $t_{\ell,i+1}^{(k+1)}$, the maximal difference can only occur before that time, and

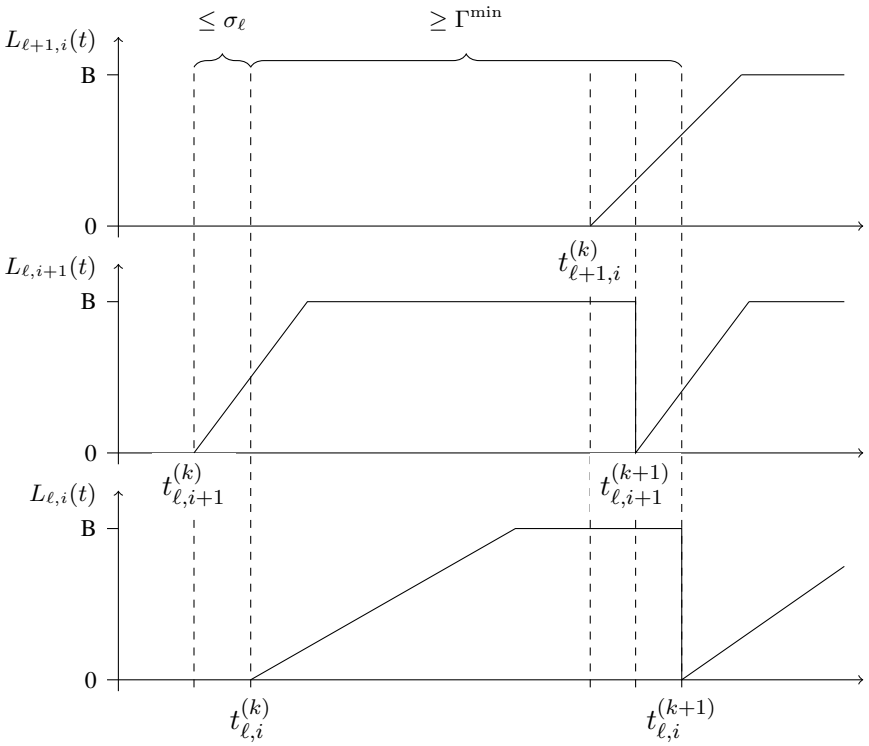


Figure 4.5: Exemplary logical clock function plots. The x-axis is the timeline, the y-axis is the real-valued reading of the logical clock at the respective time. The node $(\ell, i+1)$ fired first, and the fast clock runs at a rate of $f_{\max}$. While the node (ℓ, i) fired later and run at a rate of $f_{\min}$.

is hence subsumed in Equation (4.1), which also holds for $t = t^{(k+1)}_{\ell,i+1}$, cf. Figure 4.5. The case for $t \in \left[t^{(k+1)}_{\ell,i}, t^{(k+2)}_{\ell,i}\right] \cap \left[t^{(k)}_{\ell,i+1}, t^{(k+1)}_{\ell,i+1}\right]$ is symmetric. Thus, all cases have been covered and Equation (4.1) gives the worst-case precision, if for σ_ℓ the worst-case bound $\sigma^{\max}_\ell$ is used; if the average intra-layer skew σ^{avg}_ℓ was used, we would get some bound for the average precision.

The derivation of the inter-layer bound could be done by replacing σ_ℓ with $\hat{\sigma}_\ell$, but as there is a known bias in the inter-layer skew (cf. Figure 3.18), the result would be overly conservative. For the refined analysis of the average case, we employ the approximation $\bar{\sigma}_\ell = \left(\hat{\sigma}^{\max}_\ell + \hat{\sigma}^{\min}_\ell\right)/2$ of the bias, which allows us to shift the clock according to $L'_{\ell,i}(t) = (L_{\ell,i}(t) + f_{\min}\bar{\sigma}_\ell) \bmod B$. As the inter-layer skew is not absolute, contrary to the intra-layer skew, the clock skew has to be considered in both directions. Analogously to the derivation of Equation (4.1), we obtain

$$\begin{aligned} L_{\ell-1,i}(t) - L'_{\ell,i}(t) &\leq B - f_{\min}\left(\frac{B}{f_{\max}} - \left(t^{(k)}_{\ell-1,i} - t^{(k)}_{\ell,i}\right) + \bar{\sigma}_\ell\right) \\ &\leq \frac{f_{\max} - f_{\min}}{f_{\max}} \cdot B + f_{\min}(\hat{\sigma}^{\max}_\ell - \bar{\sigma}_\ell) \\ &\leq \varrho B + f_{\min}\tau_\ell, \end{aligned}$$

with $\tau_\ell = \left(\hat{\sigma}_\ell^{\max} - \hat{\sigma}_\ell^{\min}\right)/2$, and ϱ as before. With the symmetric case

$$\begin{aligned} L'_{\ell,i}(t) - L_{\ell-1,i}(t) &\leq B - f_{\min}\left(\frac{B}{f_{\max}} - \left(t_{\ell,i}^{(k)} - t_{\ell-1,i}^{(k)}\right) - \bar{\sigma}_\ell\right) \\ &\leq \frac{f_{\max} - f_{\min}}{f_{\max}} \cdot B + f_{\min}(-\hat{\sigma}_\ell^{\min} + \bar{\sigma}_\ell) \\ &\leq \varrho B + f_{\min}\tau_\ell, \end{aligned}$$

this leads to $|L'_{\ell,i}(t) - L_{\ell-1,i}(t)| \leq \varrho B + f_{\min}\tau_\ell$. Analogous derivations reveal that the same bound holds for $|L'_{\ell,i}(t) - L_{\ell-1,i+1}(t)| \leq \varrho B + f_{\min}\tau_\ell$.

Obviously, shifting all clock values of two consecutive layers ℓ, $\ell - 1$ by the same amount $\bar{\sigma}_\ell$ has no effect on the inter-layer skew $\hat{\sigma}_\ell$. Thus, inductively applying the $\bar{\sigma}_\ell$ shift at all layers $\ell \geq 1$, i.e., $\sum_{\ell'=1}^{\ell} \bar{\sigma}_{\ell'}$, does not change the inter-layer skew bound of any layer, i.e., the intra- resp. inter-layer precision of the fast clock at layer ℓ are bounded by

$$\begin{aligned} \varrho B + f_{\min}\sigma_\ell^{\max} &\leq f_{\min}\left(\varrho\Gamma^{\min} + \sigma_\ell^{\max}\right) \quad \text{resp.} \\ \varrho B + f_{\min}\tau_\ell &\leq f_{\min}\left(\varrho\Gamma^{\min} + \tau_\ell\right), \end{aligned}$$

which are tight in case of $B = f_{\min}\Gamma^{\min}$. It is apparent that if the minimal pulse separation time $\Gamma^{\min}$ is large compared to $\sigma_\ell^{\max}$ and τ_ℓ, a low drift ϱ of the ring oscillator is needed so that the precision is not dominated by the pulse separation time. We stress that $f_{\min}$ and $f_{\max}$ in our bounds are—unless the clocks are extremely unstable—the amortized frequency upper and lower bounds over B fast clock ticks; large skews require a consistent difference in frequency for roughly $B/f_{\min}$ time. Moreover, ϱ captures the *relative drift* of the clocks. Any frequency change that applies to adjacent nodes in roughly the same way (i.e., a system-wide change in temperature or supply voltage) will not have noticeable effects on the skews.

Determination of the minimal pulse separation time In the analysis above, the pulse separation time $\Gamma^{\min}$ plays a critical role, so accurately knowing its value is instrumental. In a HEX grid with large skews, and hence large jitter, $\Gamma^{\min} < \Gamma^{\text{avg}}$, hence using the allowed frequency $f_{\min}$ is quite conservative. On the other hand, in "static" systems, i.e., constant link delays, $\Gamma^{\min} = \Gamma^{\text{avg}}$, hence using $f_{\min}$ will be (close to) optimal.

There are basically three options for determining the minimal pulse separation time: We can rely on experimental data, analytical bounds, or simulation results. All approaches have pros and cons; for a final system, experimental data is the most valuable, but it is also the most difficult and expensive to create, as one needs a chip and a suitable apparatus for measurements first. Furthermore, it must be possible to induce realistic scenarios in the clock generation scheme and the HEX grid, as data collected under conditions which do not cover the target environment can lead to an underestimation of the minimal pulse separation time. From Section 2.3 and with bounds for the layer 0 clock generation algorithm employed, analytical worst-case bounds can be derived. The results from Section 3.3.2 provide insight into the skew distributions for average-case settings under some fairly conservative assumptions on the parameters.

Note carefully that an underestimation of the minimal pulse separation can lead to an interruption of a fast clock burst by its successor, which causes a loss of clock cycles and hence a transient fault of the fast clock.

After $\tilde{\Gamma}^{\min}$ has been determined by some suitable estimation, B can be chosen according to $B \leq f_{\min}\tilde{\Gamma}^{\min}$. Note that a smaller value of B improves the skew of the fast clock to its neighbors, at the price of a reduced frequency.

In all above considerations the handling of faulty nodes was left out. Moreover, due to the self-stabilizing property of HEX, the correct operation of the fast clock during stabilization cannot be guaranteed, as there is no lower bound on the time between two HEX pulses.

CHAPTER 5

Conclusions and Future Work

5.1 Summary of Accomplishments

In this thesis, a fault-tolerant, scalable and self-stabilizing clock distribution scheme named HEX, for hexagonal grid topologies, was presented. A thorough theoretical analysis of the neighbor skew in the fault-free case revealed large worst-case skews, albeit in quite exotic scenarios. HEX could also be proved to be self-stabilizing and a suitable 1–local Byzantine fault model for the HEX grid was developed as well.

To verify and enhance the knowledge gained by the theoretically analysis, thorough simulation experiments were conducted. Two simulation environments were developed for this purpose: A MATLAB based approach, and a more comprehensive ModelSim-based environment built around a VHDL implementation of the HEX algorithm. The simulations were not only used to observe the behavior of the average skews in the fault-free case, but also in settings with faults, where no theoretical analysis is available. In the fault-free case, low average skews were observed, which confirm the hypothesized rareness of the worst-case scenario. Furthermore, the behavior under faults showed that fault-effects are typically visible only in the close neighborhood of a faulty node. The self-stabilizing properties of the HEX grid were also tested by means of multi-pulse simulations, which revealed a very fast stabilization in almost all cases. This allows us to conclude that the fault-tolerance properties of HEX are, in the average-case, much better than the analytically predicted worst-case bounds.

Finally, a clock multiplication scheme was introduced, which allows to increase the frequency of the clock generated by a HEX node, while ensuring a bounded precision of that clock to the neighboring nodes.

Thus HEX is usable as a clock distribution system which allows to provided a high-frequency clock to a local node while maintaining bounded precision to the neighbor nodes.

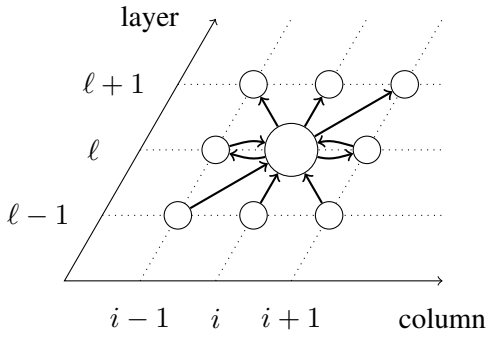


Figure 5.1: Alternative topology for HEX, which should reduce the skews.

5.2 Critical Reflection and Future Work

In this section, we will briefly describe some shortcomings of the existing solution and possible remedies, which have been developed recently [16] or will be part of our future work.

5.2.1 Self-Stabilization

While we have shown that the HEX grid stabilizes within L pulses, we have not considered stabilization in the presence of faults. While the simulations revealed good stabilization times even in the presence of faults, there are settings where a Byzantine fault can inhibit the stabilization of the original algorithm. To circumvent this problem, received trigger messages must be discarded after some time, as presented in [15].

5.2.2 Alternative Grid Topologies

The hexagonal network structure was chosen due to its regularity and planarity. While planarity simplifies the routing of the links, it also limits the possible node degrees of the network. Still, a larger node degree would reduce the power of faults, as a larger fraction of trigger messages from correct neighbors can be relied on. This would have a positive effect on the skews, although yet unknown side effects can of course not be ruled out.

In Figure 5.1, an alternative topology is shown, which has an additional diagonal. This should provide better skew results, as an additional trigger message is available, and thus the probability of an intra-layer triggering is reduced.

In Figure 5.2, a more drastic change in the topology is shown. It would allow to increase the number of faulty nodes in the neighborhood to 2, if the modified algorithm requires 3 trigger messages to trigger a node. Additionally, due to the increased connectivity, the triggering by intra-layer neighbors would become even more rare then with the previously presented topology, which should make large neighbor skews an exception.

Both alternative topologies presented so far have a common drawback: The length of the links differ considerably, which will increase the interval $[d^-, d^+]$. This is also a problem when

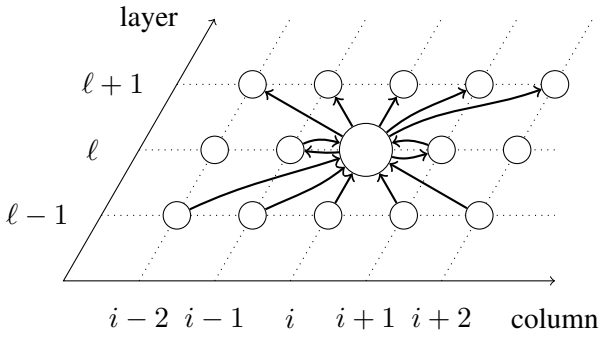


Figure 5.2: Alternative topology for HEX. This topology allows to tolerate 2 faulty neighbors and reduce the skews.

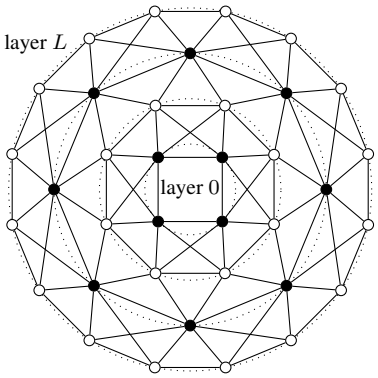


Figure 5.3: Alternative topology for embedded usage of HEX. White nodes are in doubling layers. Doubling layers become less frequent with increasing distance from the center.

the HEX grid is embedded in a chip and the clock sources are located in the center. As suggested in [16], a solution for this problem are doubling layers, as shown in Figure 5.3. In this topology, the layers are arranged in a circular pattern. To avoid ever increasing link wire lengths between the nodes in outer layers, "doubling layers" are inserted, which double the amount of HEX nodes per layer. This topology has several advantages over the cylindric HEX grid: First, doubling layers will help disperse skews and thus may improve the bounds due to the same arguments as for the other topologies. Moreover, since the "initial" width W close to the clock sources is very small and most doubling layers are close to the sources as well, any initial skew will be mitigated very quickly.

5.2.3 Fault Analysis

While the cornerstones of the analysis of the skews with one Byzantine faulty node are outlined in [15], which reveals a increase of $\mathcal{O}(d^+)$ in the skew, the case $f > 1$ results in a combinatorial explosion. The handling of these cases manually seems to be out of reach, but there may be ways

to apply automatic verification to solve the problem.

5.2.4 Real Implementation

The implementation presented in this paper lacks realistic timing information, as no routing was conducted. While we can expect that HEX will work properly on an ASIC, it would be interesting to devise and build a custom transistor cell for a HEX node. This would reduce the processing delay of the node, as well as the area requirements.

5.2.5 Metastability Filtering

The currently used receivers in the implementation can become metastable when the received trigger message is too short. Using elastic pipelines [68] as metastability filters [26] could be used to mitigate this problem. While this transformation alone does not change the implementation of a HEX node, both d^- and d^+ are increased by the additional latency of the pipeline.

5.3 Applications

The HEX clock distribution system can be used to provided a synchronized clock signal in GALS systems. Given the bounded precision of the fast clocks, it is tempting to utilizes this property for implementing synchronous communication between neighbor nodes. In the fault-free case, and without self-stabilization, the scheme presented in [55] could be used to implement metastability-free communication. But as HEX, and thus the fast clock, is self-stabilizing, this scheme cannot be employed, without modifications.

As as a first step sketched in [54], we developed a communication scheme that uses the properties of the fast clocks to provide a pseudo-stabilizing [8], metastability-aware communication between neighboring nodes. Pseudo-stabilizing means that the system will stabilize after all transient faults have ceased, but there is no time bound on when this is going to happen. Turning it into a metastability-free self-stabilizing communication scheme is ongoing work.

Bibliography

[1] M. Abramson and W. O. J. Moser, "More Birthday Surprises," *The American Mathematical Monthly*, vol. 77, no. 8, pp. 856–858, Oct. 1970. [Online]. Available: http://www.jstor.org/stable/2317022

[2] H. Attiya, A. Herzberg, and S. Rajsbaum, "Optimal Clock Synchronization under Different Delay Assumptions," in *Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing*. New York, NY, USA: ACM, 1993, pp. 109–120. [Online]. Available: http://doi.acm.org/10.1145/164051.164067

[3] H. Attiya and J. Welch, *Distributed Computing: Fundamentals, Simulations and Advanced Topics (2nd edition)*. John Wiley Interscience, Mar. 2004.

[4] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[5] R. Baumann, "Radiation-Induced Soft Errors in Advanced Semiconductor Technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 305–316, 2005.

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

[7] V. Bhandari and N. H. Vaidya, "On Reliable Broadcast in a Radio Network," in *Proceedings of the 24th annual ACM Symposium on Principles of Distributed Computing*. New York, NY, USA: ACM, 2005, pp. 138–147. [Online]. Available: http://doi.acm.org/10.1145/1073814.1073841

[8] J. E. Burns, M. G. Gouda, and R. E. Miller, "Stabilization and Pseudo-Stabilization," *Distributed Computing*, vol. 7, no. 1, pp. 35–42, 1993.

[9] D. M. Chapiro, "Globally-Asynchronous Locally-Synchronous Systems," Ph.D. dissertation, Stanford University, Stanford, CA, USA, Oct. 1984.

[10] C. Constantinescu, "Trends and Challenges in VLSI Circuit Reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, 2003.

[11] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: A Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers," 1996.

[12] M. E. Dean, T. E. Williams, and D. L. Dill, "Efficient Self-Timing with Level-Encoded 2-Phase Dual-Rail (LEDR)," in *Proceedings of the 1991 University of California/Santa Cruz conference on Advanced research in VLSI*. Cambridge, MA, USA: MIT Press, 1991, pp. 55–70. [Online]. Available: http://dl.acm.org/citation.cfm?id=112073.112078

[13] E. W. Dijkstra, "Self-Stabilizing Systems in Spite of Distributed Control," *Communications of the ACM*, vol. 17, no. 11, pp. 643–644, 1974.

[14] D. Dolev, "The Byzantine Generals Strike Again," *Journal of Algorithms*, vol. 3, no. 1, pp. 14–30, 1982.

[15] D. Dolev, M. Függer, C. Lenzen, M. Perner, and U. Schmid, "HEX: Scaling Honeycombs is Easier than Scaling Clock Trees," Journal version of [16] (submitted).

[16] ——, "HEX: Scaling Honeycombs is Easier than Scaling Clock Trees," in *Proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures*, 2013.

[17] D. Dolev, M. Függer, C. Lenzen, M. Posch, U. Schmid, and A. Steininger, "FATAL$^+$: A Self-Stabilizing Byzantine Fault-Tolerant Clocking Scheme for SoCs," *Computing Research Repository*, Feb. 2012.

[18] D. Dolev, M. Függer, C. Lenzen, and U. Schmid, "Fault-Tolerant Algorithms for Tick-Generation in Asynchronous Logic: Robust Pulse Generation," in *Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems*. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 163–177. [Online]. Available: http://dl.acm.org/citation.cfm?id=2050613.2050627

[19] S. Dolev, *Self-Stabilization*. MIT Press, Mar. 2000.

[20] S. Dolev and J. L. Welch, "Self-Stabilizing Clock Synchronization in the Presence of Byzantine Faults," *Journal of the ACM*, vol. 51, no. 5, pp. 780–799, Sep. 2004. [Online]. Available: http://doi.acm.org/10.1145/1017460.1017463

[21] S. Fairbanks, "Method and Apparatus for a Distributed Clock Generator," US Patent 7,126,405, 2003.

[22] S. Fairbanks and S. Moore, "Self-Timed Circuitry for Global Clocking," in *Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems*, 2005.

[23] K. M. Fant and S. A. Brandt, "NULL Convention Logic™ System," US Patent 5,305,463, Apr., 1994.

[24] 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*, Oct. 2006.

[25] E. G. Friedman, "Clock Distribution Networks in Synchronous Digital Integrated Circuits," *Proceedings of the IEEE*, vol. 89, no. 5, pp. 665–692, 2001.

[26] G. Fuchs, M. Függer, and A. Steininger, "On the Threat of Metastability in an Asynchronous Fault-Tolerant Clock Generation Scheme," in *Proceedings of the 15th IEEE Symposium on Asynchronous Circuits and Systems*, May 2009, pp. 127–136.

[27] G. Fuchs and A. Steininger, "VLSI Implementation of a Distributed Algorithm for Fault-Tolerant Clock Generation," *Journal of Electrical and Computer Engineering*, vol. 2011, no. 936712, 2011.

[28] M. Függer and U. Schmid, "Reconciling Fault-Tolerant Distributed Computing and Systems-on-Chip," *Distributed Computing*, vol. 24, pp. 323–355, 2012. [Online]. Available: http://dx.doi.org/10.1007/s00446-011-0151-7

[29] M. Függer, U. Schmid, G. Fuchs, and G. Kempf, "Fault-Tolerant Distributed Clock Generation in VLSI Systems-on-Chip," *6th European Dependable Computing Conference*, Oct. 2006.

[30] A. Giridhar and P. Kumar, "Distributed Clock Synchronization over Wireless Networks: Algorithms and Analysis," in *Proceedings of the 45th IEEE Conference on Decision and Control*, Dec. 2006, pp. 4915–4920.

[31] V. Gutnik and A. Chandrakasan, "Active GHz Clock Network Using Distributed PLLs," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 11, pp. 1553–1560, 2000.

[32] J. Y. Halpern, N. Megiddo, and A. A. Munshi, "Optimal Precision in the Presence of Uncertainty," *Journal of Complexity*, vol. 1, no. 2, pp. 170–196, 1985. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0885064X8590010X

[33] IEEE-SA Standards Board, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, 2008.

[34] C.-Y. Koo, "Broadcast in Radio Networks Tolerating Byzantine Adversarial Behavior," in *Proceedings of the 23rd annual ACM Symposium on Principles of Distributed Computing*. New York, NY, USA: ACM, 2004, pp. 275–282. [Online]. Available: http://doi.acm.org/10.1145/1011767.1011807

[35] A. Korniienko, E. Colinet, G. Scorletti, E. Blanco, D. Galayko, and J. Juillard, "A Clock Network of Distributed ADPLLs Using an Asymmetric Comparison Strategy," in *Proceedings of the 2010 Symposium on Circuits and Systems*, 2010, pp. 3212–3215.

[36] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978. [Online]. Available: http://doi.acm.org/10.1145/359545.359563

[37] L. Lamport and P. M. Melliar-Smith, "Synchronizing Clocks in the Presence of Faults," *Journal of the ACM*, vol. 32, no. 1, pp. 52–78, Jan. 1985. [Online]. Available: http://doi.acm.org/10.1145/2455.2457

[38] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, Jul. 1982. [Online]. Available: http://doi.acm.org/10.1145/357172.357176

[39] D.-J. Lee, M.-C. Kim, and I. Markov, "Low-Power Clock Trees for CPUs," in *Proceedings of the 2010 Conference on Computer-Aided Design*, 2010, pp. 444–451.

[40] D.-J. Lee and I. Markov, "Multilevel Tree Fusion for Robust Clock Networks," in *Proceedings of the 2011 Conference on Computer-Aided Design*, 2011, pp. 632–639.

[41] I. Loi, S. Mitra, T. H. Lee, S. Fujita, and L. Benini, "A Low-Overhead Fault Tolerance Scheme for TSV-based 3D Network on Chip Links," in *Proceedings of the 2008 International Conference on Computer-Aided Design*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 598–602. [Online]. Available: http://dl.acm.org/citation.cfm?id=1509456.1509589

[42] J. Lundelius and N. A. Lynch, "An Upper and Lower Bound for Clock Synchronization," *Information and Control*, vol. 62, no. 2/3, pp. 190–204, 1984.

[43] C.-L. Lung, Y.-S. Su, S.-H. Huang, Y. Shi, and S.-C. Chang, "Fault-Tolerant 3D Clock Network," in *Proceedings of the 48th Design Automation Conference*. New York, NY, USA: ACM, 2011, pp. 645–651. [Online]. Available: http://doi.acm.org/10.1145/2024724.2024872

[44] L. R. Marino, "General Theory of Metastable Operation," *IEEE Transactions on Computers*, vol. 30, no. 2, pp. 107–115, Feb. 1981.

[45] M. S. Maza and M. L. Aranda, "Interconnected Rings and Oscillators as Gigahertz Clock Distribution Nets," in *Proceedings of the 13th Great Lakes Symposium on VLSI*, 2003, pp. 41–44.

[46] A. McAuley, "Four State Asynchronous Architectures," *IEEE Transactions on Computers*, vol. 41, no. 2, pp. 129–142, 1992.

[47] D. G. Messerschmitt, "Synchronization in Digital System Design," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 8, pp. 1404–1419, Oct. 1990.

[48] D. Mills, "Internet Time Synchronization: The Network Time Protocol," *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[49] D. E. Muller and W. S. Bartky, "A Theory of Asynchronous Circuits," in *Proceedings of the International Symposium on Theory of Switching*. Harvard University Press, Cambridge, Mass., 1959, pp. 204–243.

[50] T. Olsson and P. Nilsson, "Portable Digital Clock Generator for Digital Signal Processing Applications," *Electronics Letters*, vol. 39, no. 19, pp. 1372 – 1374, Sep. 2003.

[51] T. Olsson, P. Nilsson, T. Meincke, A. Hemam, and M. Torkelson, "A Digitally Controlled Low-Power Clock Multiplier for Globally Asynchronous Locally Synchronous Designs," in *Proceedings of the 2000 Symposium on Circuits and Systems*, vol. 3, 2000, pp. 13–16.

[52] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *Journal of the ACM*, vol. 27, no. 2, pp. 228–234, Apr. 1980. [Online]. Available: http://doi.acm.org/10.1145/322186.322188

[53] A. Pelc and D. Peleg, "Broadcasting with Locally Bounded Byzantine Faults," *Information Processing Letters*, vol. 93, no. 3, pp. 109–115, Feb. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.ipl.2004.10.007

[54] M. Perner, M. Sigl, U. Schmid, and C. Lenzen, "Byzantine Self-Stabilizing Clock Distribution with HEX: Implementation, Simulation, Clock Multiplication," in *Proceedings of the 6th Conference on Dependability*, 2013.

[55] T. Polzer, T. Handl, and A. Steininger, "A Metastability-Free Multi-synchronous Communication Scheme for SoCs," in *Stabilization, Safety, and Security of Distributed Systems*, ser. Lecture Notes in Computer Science, R. Guerraoui and F. Petit, Eds. Springer Berlin Heidelberg, 2009, vol. 5873, pp. 578–592. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-05118-0_40

[56] G. A. Pratt and J. Nguyen, "Distributed Synchronous Clocking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 3, pp. 314–328, Mar. 1995.

[57] P. J. Restle, T. G. McNamara, D. A. Webber, P. J. Camporese, K. F. Eng, K. A. Jenkins, D. H. Allen, M. J. Rohn, M. P. Quaranta, D. W. Boerstler, C. J. Alpert, C. A. Carter, R. N. Bailey, J. G. Petrovick, B. L. Krauter, and B. D. McCredie, "A Clock Distribution Network for Microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 5, pp. 792–799, 2001.

[58] M. Saint-Laurent and M. Swaminathan, "A Multi-PLL Clock Distribution Architecture for Gigascale Integration," in *Proceedings of the 2001 IEEE Computer Society Workshop on VLSI*, 2001, pp. 30–35.

[59] F. B. Schneider, "Understanding Protocols for Byzantine Clock Synchronization," Ithaca, NY, USA, Tech. Rep., Aug. 1987. [Online]. Available: http://hdl.handle.net/1813/6699

[60] Y. Semiat and R. Ginosar, "Timing Measurements of Synchronization Circuits," in *Proceedings of the 9th International Symposium on Asynchronous Circuits and Systems*, 2003, pp. 68–77.

[61] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli,

"SIS: A System for Sequential Circuit Synthesis," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M92/41, 1992. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/1992/2010.html

[62] R. Shelar, "Routing with Constraints for Post-Grid Clock Distribution in Microprocessors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 2, pp. 245–249, 2010.

[63] X. Shi, K. Imfeld, S. Tanner, M. Ansorge, and P. A. Farine, "A Low-Jitter and Low-Power CMOS PLL for Clock Multiplication," in *Proceedings of the 32nd European Solid-State Circuits Conference*, 2006, pp. 174–177.

[64] J. Sparsø, "Asynchronous Circuit Design - A Tutorial," in *Principles of Asynchronous Circuit Design - A Systems Perspective*. Boston / Dordrecht / London: Kluwer Academic Publishers, Dec. 2001, pp. 1–152. [Online]. Available: http://www2.imm.dtu.dk/pubdb/p.php?855

[65] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design: A Systems Perspective*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[66] T. K. Srikanth and S. Toueg, "Optimal Clock Synchronization," in *Proceedings of the 4th annual ACM Symposium on Principles of Distributed Computing*. New York, NY, USA: ACM, 1985, pp. 71–86. [Online]. Available: http://doi.acm.org/10.1145/323596.323603

[67] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, "Clock Synchronization for Wireless Sensor Networks: A Survey," *Ad Hoc Networks*, vol. 3, no. 3, pp. 281 – 323, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870505000144

[68] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, pp. 720–738, Jun. 1989. [Online]. Available: http://doi.acm.org/10.1145/63526.63532

[69] S. Tam, J. Leung, and R. Limaye, "Clock Generation & Distribution for a 45nm, 8-Core Xeon® Processor with 24MB Cache," in *Proceedings of the 2009 Symposium on VLSI Circuits*, 2009, pp. 154–155.

[70] P. Teehan, M. Greenstreet, and G. Lemieux, "A Survey and Taxonomy of GALS Design Styles," *IEEE Design and Test of Computers*, vol. 24, no. 5, pp. 418–428, 2007.

[71] UMC, "90 nm." [Online]. Available: http://www.umc.com/English/process/g.asp

[72] M. Vai, *VLSI Design*, ser. VLSI circuits series. CRC Press, 2001.

[73] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI Test Principles and Architectures: Design for Testability (Systems on Silicon)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[74] T. Watanabe and S. Yamauchi, "An All-Digital PLL for Frequency Multiplication by 4 to 1022 with Seven-Cycle Lock Time," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 2, pp. 198–204, 2003.

[75] T. Weigandt, B. Kim, and P. Gray, “Analysis of Timing Jitter in CMOS Ring Oscillators,” in *Proceedings of the 1994 IEEE International Symposium on Circuits and Systems*, vol. 4, 1994, pp. 27–30.

[76] J. L. Welch and N. Lynch, “A new Fault-Tolerant Algorithm for Clock Synchronization,” in *Information and Computation*, 1988, pp. 75–88.

[77] C. Yeh, G. Wilke, H. Chen, S. Reddy, H. Nguyen, T. Miyoshi, W. Walker, and R. Murgai, “Clock Distribution Architectures: A Comparative Study,” in *Proceedings of the 7th International Symposium on Quality Electronic Design*, Mar. 2006, pp. 7–91.

Nomenclature

$[X]$	Defines the set $\{0, \dots, X-1\}$.	Page 17
$((\ell, i), (\ell', i'))$	Annotates the link from node (ℓ, i) to node (ℓ', i').	Page 17
d^+	Maximal link delay.	Page 18
d^-	Minimal link delay.	Page 18
ε	Difference between d^+ and d^-.	Page 18
$p_{\text{left}}^{i' \to (\ell,i)}$	The left zig-zag path to node (ℓ, i).	Page 19
r	The relative width of a left zig-zag path.	Page 19
$t_{\ell,i}^{(k)}$	Defines the time when the node (ℓ, i) executes its k^{th} tick. When k is clear from the context, it is omitted.	Page 20
$t_{-}^{(k)}$	Denotes the time when the first node in layer 0 fires pulse k.	Page 31
$t_{+}^{(k)}$	Denotes the time when the last node in layer 0 fires pulse k.	Page 31
Δ_ℓ	The skew potential of layer ℓ.	Page 21
σ_ℓ	The intra-layer skew of layer ℓ.	Page 27
$\hat{\sigma}_\ell$	The inter-layer skew of layer ℓ.	Page 50
σ_ℓ^{op}	Defines the application of op on the intra-layer skew of layer ℓ.	Page 50
$\hat{\sigma}_\ell^{\text{op}}$	Defines the application of op on the inter-layer skew of layer ℓ.	Page 50
q_x	Defines the x^{th} quantile of the global intra-layer skew.	Page 50
$\hat{q}_x$	Defines the x^{th} quantile of the global inter-layer skew.	Page 50
σ^{op}	Defines the application of op on the global intra-layer skew.	Page 50
$\hat{\sigma}^{\text{op}}$	Defines the application of op on the global inter-layer skew.	Page 50

h	Defines the directed hop distance from a faulty node. Every node which is reachable from a faulty node via h hops is excluded from skew calculations. Page 53
S	The stabilization time. Page 31
$\Sigma(S)$	The number of runs which stabilized. Page 65
$\Gamma^{\min}$	The minimal pulse separation time. Page 65