M A G I S T E R A R B E I T

Fault-Tolerant Distributed Clock Generation in VLSI Systems-on-Chip

ausgeführt am Institut für

Technische Informatik, Embedded Computing Systems Group
Technische Universität Wien

unter der Anleitung von
Univ. Prof. Dr. Ulrich Schmid

durch

Matthias Függer
Friedenshöhegasse 36
1130 Wien
Austria
European Union

Wien, 21. März 2006 ________________________ to Michaela, Reinhold, Barbara and Hanna

Zusammenfassung

Zurzeit bewirken konzeptuelle Probleme im synchronen Design-Paradigma gravierenden Zusatzaufwand. Dieser ist erforderlich um die für das synchrone Design notwendige Abstraktion einer diskreten globalen Zeit über einen ganzen Chip hinweg zu gewährleisten.
In dieser Magisterarbeit wird die zentrale Grundlage für eine Alternative zu synchronen Designs vorgestellt, die im Rahmen des DARTS Projektes (einer Kooperation zwischen der Technischen Universität Wien und Austrian Aerospace) näher untersucht wird. Dabei wird der Chip in lose verbundene funktionale Einheiten gegliedert, die ein "System-on-Chip" (SoC) bilden. In der Arbeit wird gezeigt, wie der bekannte Uhren-synchronisations-algorithmus von Srikanth und Toueg für die direkte Implementierung in digitaler Hardware adaptiert und somit dazu verwendet werden kann einen fehlertoleranten verteilten Takt für die funktionalen Einheiten zu generieren. Danach wird formal bewiesen, dass die solcher Art erzeugten Taktsignale nicht unabhängig sind, sondern gewissen Synchronisationsbedingungen (maximale/minimale Frequenz, maximale Phasenabweichung zwischen zwei beliebigen Taktsignalen) genügen. Auf diese Weise wird eine für die Interaktionen zwischen den funktionalen Einheiten wesentliche globale SoC Zeit über dem gesamten Chip geschaffen. Schlussendlich wird gezeigt, dass der präsentierte Algorithmus auch wirklich in Hardware implementierbar ist, indem erste vielversprechende Resultate einer FPGA Implementierung vorgestellt werden.

Abstract

Due to conceptual problems of synchronous design methodologies, considerable effort must currently be spent on maintaining the abstraction of a discrete global time on the entire chip.
This master's thesis presents the pivotal basis of an alternative design principle, which is currently investigated in the DARTS project (a cooperation between the Vienna University of Technology and Austrian Aerospace): A chip is partitioned into loosely coupled functional units, that, together, form a System-on-Chip (SoC). It is shown how the well-known clock synchronization algorithm by Srikanth and Toueg can be adopted to the peculiarities of digital hardware design and hence be used for generating a fault-tolerant distributed clock signal for each functional unit. It is formally proved that the different clock signals are not independent of each other, but satisfy certain synchronization properties (frequency bounds, bounds on maximum phase differences between different clock signals). Consequently a global SoC time can be generated from the local clock signals, which facilitates inter-functional unit communication.
Finally, the feasibility of implementing the proposed algorithm directly in hardware is demonstrated via first promising results of an FPGA implementation.

Acknowledgement

I would like to thank my advisor Prof. Ulrich Schmid, who made me conscious of the field of distributed algorithms and who sparked my interest on their analysis. I would, further, like to thank him for being a great help when writing the proofs and for many interesting discussions.
I am grateful to Prof. Andreas Steininger for his great lectures on digital design, which finally encouraged me to combine digital design with distributed algorithms.
Many thanks go to Gottfried Fuchs and Josef Widder for spending hours discussing the topics of this master's thesis and to Martin Biely, for founding the RGDD (together with Josef Widder) and helping with LaTeX.
The contributions of Gottfried Fuchs (hardware schematics and implementation), Johann Vilanek (Diff-Gate, preliminary simulations and experiments), Markus Ferringer (FPGA implementation) and Thomas Handl (tools and library setup) are gratefully acknowledged.
This master's thesis was funded by the bm:vit FIT-IT project *DARTS* (proj. no. 809456-SCK/SAI, *http://www.ecs.tuwien.ac.at/projects/DARTS*), which is a collaboration between the Vienna University of Technology (at the time of writing represented by: Prof. Ulrich Schmid, Prof. Andreas Steininger, Gottfried Fuchs, Thomas Handl, Josef Widder, Matthias Függer, Markus Ferringer) and Austrian Aerospace (Manfred Sust, Gerald Kempf, Christian Tögel, Georg Grabmayr, Franz Zangerl).
Many thanks go to Josef Widder, Martin Biely, Daniel Albeseder, Gottfried Fuchs, Wilfried Elmenreich and Peter Tummeltshammer for helpful suggestions for the presentation of this master's thesis.
Finally I would like to thank my family and my friends for their graceful support in all aspects!

Contents

List of Figures

1 Introduction

The integration of synchronous digital designs, consisting of several billions of transistors [1] in VLSI chips with decreasing feature size and increasing clock speed is facing more and more problems. In todays designs, aiming at clock speeds in the GHz range, routing delays dominate gate delays by far and lead to ever more complexity in the already intricate design phase. These problems can only be solved by a conceptually new approach, because of the impossibility of bypassing physical laws, like the speed of electromagnetic waves. Assuming a speed of $2/3 \cdot c_0 \approx 20$ [cm/ns] (c_0 denotes the speed of light in vacuum), a 10 GHz clock signal needs a complete clock period to travel 2 cm, a length that lies in the range of common die sizes. In other words, the designer has to cope with up to 100% clock skew[1]. As can be seen from this example, the main advantage of the synchronous design paradigm — developing hardware that performs state transitions at discrete points in time only, at the entire chip — has already vanished. Hence, the effort of designing a clock tree for a monolithic synchronous chip is not a neglectable task anymore. Skew-minimization techniques include: X-Trees, H-Trees and programmable clock delays [12].

But not only the design process is affected. Due to the higher clock frequencies Φ, the chip has to operate at a lower voltage level V to compensate the power dissipation given by $P \propto V^2 \cdot \Phi$ [16]. A reduction of the voltage level, however, increases the risk of bit errors — for example by single-event effects due to cosmic radiation and crosstalk [27]. A classification of possible faults that have to be handled in near-future VLSI designs is given in [7].

To avoid the problems resulting from monolithic synchronous designs, chips are partitioned into loosely coupled functional subsystems — forming a so called System-on-Chip (SoC). Due to the high cohesion and reasonably loose coupling of typical SoC modules, this approach is very well suited to be

[1]The clock skew is the proportion of the difference in time that a clock signal needs to travel to two different places on the chip to the duration of a complete clock cycle.

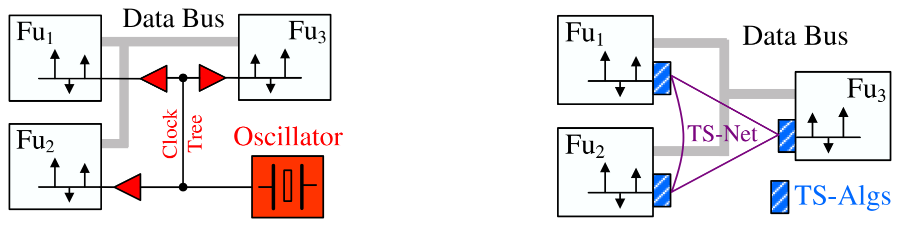


Figure 1: Replacing synchronous clocking with a fault-tolerant distributed tick synchronization algorithm TS-Alg.

modeled as a distributed system and to be analyzed by established methods from the distributed algorithms community.

In this thesis it is investigated whether some of the work on fault-tolerant clock synchronization can be adapted for usage in VLSI SoCs. The thesis is based on a joint project, named DARTS (Distributed Algorithms for Robust Tick Synchronization), between the Vienna University of Technology and Austrian Aerospace. A short overview on the project can be found in [8]. A more thorough description on the project is available as a patent [30] and a technical report [13].

The alternative approach to synchronous clocking proposed in this thesis (and also published in [13]) tries to overcome the problem of designing a global clock tree, but still maintains a reasonably synchronous view. This is achieved by not driving the complete SoC by a single clock generated by a quartz oscillator, but letting the SoC units generate their clocks by themselves, using asynchronous logic. As sketched in Fig. 1, the SoC comprises several functional units Fu_i, each of which is provided with a local clock signal generated by a small additional TS-Alg unit attached to it. The n TS-Algs communicate via the TS-Net (consisting of n broadcast channels) and are responsible for generating the distributed clock signals.

Our approach targets the following properties:

Fault-Tolerance: In contrast to synchronous designs which face the risk of a total breakdown when the clock/oscillator fails, the TS-Algs can tolerate up to f arbitrary faulty TS-Algs, if $n \geq 3f + 2$.

Synchronity: In contrast to the GALS (globally asynchronous, locally synchronous) approach [6], bounds on the maximum phase-difference of clock signals can be given. By dividing the generated clock by the maximum phase-difference, the Fu_i can also communicate with each other (at a reduced clock speed), using a synchronous interface, without facing metastability problems resulting from violated setup and hold constraints.

Graceful Degradation: The clock always runs at the maximum possible speed according to temperature, physical layout and other conditions, allowing graceful-degradation. This is in contrast to synchronous designs which fail completely if improper overclocking occurs.

Reduced EM Radiation: The small variation in the TS-Alg's local clock frequencies leads to a smoother frequency spectrum and avoids peaks at multiples of the common clock period, thus lowering the maximum power spectral density. As the clock signals are not perfectly aligned in phase, the effect of ground bouncing [22] can be reduced, too.

1.1 Related work

There exist numerous papers both on SoC design and on fault-tolerant clock synchronization, but nearly no research addressing both fields.
A good overview of asynchronous design principles, including transition logic, is given in [26]. However, these designs cannot cope with failures. Fault-tolerance is typically achieved at gate-level by using radiation hard gate libraries, adopted factoring processes and cell designs using checksums [25].
Examples for application-independent fault-tolerance implemented in hardware are MAFT [17], SAFEBUS [15], GUARDS [28] and TTP [18]. However,

these systems are designed for large-scale applications, like networks of controllers, and are not suitable for small-scale SoC networks.
In the distributed algorithms community, only few existing works targeted the implementation of algorithms in hardware. An exception is [2], where it has been proved that consensus can be solved with 1-bit messages.
Instead of using quartz oscillators in (partially) synchronous designs, some approaches [23, 11, 10] generate the clock signal by inverting feedback loops. This enables an adoption of clock speed to current environmental conditions and hence allows graceful degradation. However, none of theses solutions provides a fault-tolerant clock signal.

2 Distributed Clock Generation

This section presents an adapted version [33, 35] of the well-known consistent broadcast algorithm by Srikanth and Toueg [31, 29, 20] which can also be used for clock generation. Furthermore, we adapt this algorithm to the peculiarities of VLSI design, which leads to the basic architecture of a TS-Alg (Tick Synchronization Algorithm) unit.

2.1 The Original Algorithm

While Srikanth and Toueg's algorithm [31] needs an *init(k)* (where $k \in \mathbb{N}$) and an *echo(k)* message, the adapted version [33, 35] does only need a single type of message, called *tick(k)*. The algorithm is listed in Fig. 2. A system consists of $n \geq 3f + 1$ distributed processes, where f processes may fail arbitrarily. Assume that end-to-end message delays are bounded by $[\tau^-, \tau^+]$, with the bounds possibly unknown.

The algorithm works as follows: At reset time all correct processes p send *tick(0)*. On the reception of $f + 1$ *tick(ℓ)* messages by p (`line 5`), process p can be sure that at least one of the messages was sent by a correct process. Hence, p can safely send the messages *tick(k)*, …, *tick(ℓ)*. On the reception of $2f + 1$ *tick(k)* messages by p (`line 7`), it can be sure that at least $f + 1$ of those have been sent by correct processes and will therefore be received by all other correct processes, which then execute `line 5` within $\varepsilon = \tau^+ - \tau^-$ and thus send *tick(k)* to all other processes. Hence, every other correct process $q \neq p$ receives $2f + 1$ *tick(k)* messages within another τ^+. It follows that if p emits *tick(k)* at time t every other correct process q emits *tick(k)* no later than $t + \varepsilon + \tau^+$. This eventually guarantees a precision bound $\pi \geq \max |k_p(t) - k_q(t)|$ for all correct processes p and q, where $k_p(t)$ is the number of the last tick received by process p by time t. Note that the algorithm automatically adapts to the instantaneous timing characteristics of all involved computations and message transmissions.

```
VAR $k$ : integer := 0;

/* Initialization */
send tick(0) to all [once];

if received tick($\ell$) from at least $f + 1$ distinct processes with $\ell \geq k$
    send tick($k$), ..., tick($\ell$) to all [once]; $k := \ell$;
if received tick($k$) from at least $2f + 1$ distinct processes
    send tick($k + 1$) to all [once]; $k := k + 1$;
```

Figure 2: Simple algorithm for generating approximately simultaneous messages taken from [33, 35], based on a simplified version of consistent broadcast.

2.2 First Modifications

The algorithm presented in Section 2.1 looks deceptively simple, as it comprises two simple rules only. Note, however, that at the second glance, the code elements used are well-suited for description of algorithms in software but less for algorithms in hardware. Thus a number of modifications must be undertaken before it is applicable for usage in VLSI SoCs:

Bounded space messages: The algorithm shown in Fig. 2 communicates by sending *tick(ℓ)* messages, where $\ell \in \mathbb{N}$. This would require the processes to store the arbitrary large tick numbers locally which is of course not feasible in hardware. It will be shown that communicating via two types of 0-bit messages is sufficient to generate a distributed clock. We denote the 0-bit messages by *tick*-$\uparrow$ /*tick*-$\downarrow$, corresponding to the rising and falling clock signal transitions. Thus the TS-Net can be implemented by n signal lines, each driven by our TS-Alg unit and read by all others.

Atomic execution of rules: In comparison to an algorithm implemented in software, which is executed line by line and can assume serialized execution of local actions, atomicity must be explicitly ensured in hardware, where all computations run in parallel. In our case this means that the hardware must guarantee that the message representing *tick(k)* is not sent twice. Since messages are only numbered modulo 2 ($\uparrow$ / $\downarrow$) and thus provide at-least-once semantics only modulo 2, sending a *tick*-$\downarrow$ message twice has no effect unless a *tick*-$\uparrow$ message was sent in between.

A schematic of a single unit executing the algorithm incorporating the modifications listed above is shown in Fig. 3. The TS-Alg consists of the following components:

+/− **counter:** For each process there exists a +/− counter per (remote and local) TS-Alg. The +/− counter counts the number of tick messages seen so far from every process and compares this with the number of locally generated tick messages. Thus, only the relative differences have to be stored. It is shown later that the maximum number of tick messages a +/− counter has to store can be bounded. Additionally it provides *GR* and *GEQ* signals, where *GR* is true (becomes active/enabled) if and only if (denoted as "iff" further on) the the number of remote ticks seen so far is greater than the number of local ticks seen so far. Analogously, *GEQ* is true iff the number of remote ticks is greater or equal than the number of local tick messages.

Threshold: The *GR* and *GEQ* signals are fed into two Threshold gates, which produce a new *tick*-$\uparrow$ /*tick*-$\downarrow$ message if at least $f + 1$ *GR* or $2f + 1$ *GEQ* signals are true (active), thus realizing `line 5` and `line 7` of the original algorithm (Fig. 2).

Again, the above architecture is deceptively simple. The major problem when trying to implement Fig. 3 in hardware is the lack of a common clock signal that could be used for a synchronous logic design. Rather, a (quasi) delay insensitive asynchronous implementation [14] must be devised. The major

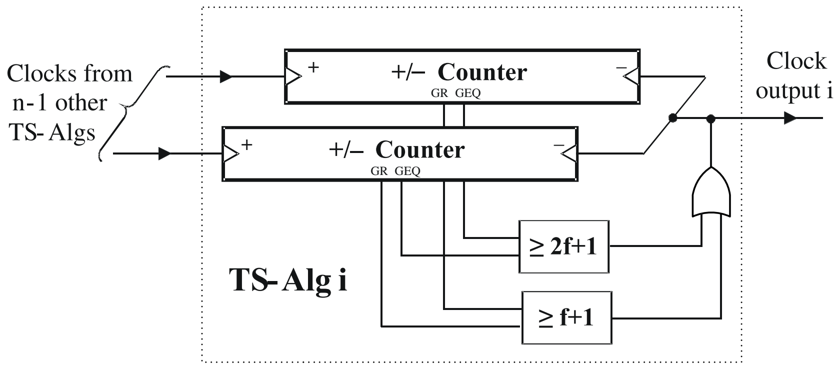


Figure 3: Basic architecture of the TS-Alg.

problem here is to distinguish old *GR* and *GEQ* signals that contributed to previously generated *tick(k)* messages from new *GR* and *GEQ* signals contributing to the next (yet to be generated) tick $k + 1$ message. The solution described here exploits the fact that the transitions of a binary-valued signal must strictly alternate between low-to-high and high-to-low: The *GR* and *GEQ* signals are split up into signals for *tick*-↓ (GR^e, GEQ^e) and *tick*-↑ (GR^o, GEQ^o) messages and fed into different threshold circuits for generating odd ($k \in \mathbb{N}_{odd} := 2\mathbb{N} + 1$) and even clock ticks ($k \in \mathbb{N}_{even} := 2\mathbb{N}$).

3 The Algorithm

It has been highlighted in the previous section that even the simple algorithm presented in Fig. 2 makes use of design elements that are not available or too costly at the hardware design level. In addition, one has to account for the fact that even the simplest (i.e., sequential) control flow comes with some delay since it actually involves sending a signal over a wire, i.e., a zero-bit FIFO message channel (a term that will be defined precisely in Section 3.1). In this section the final TS-Alg obtained by refining the considerations made in Section 2.2 is provided.

3.1 Signals and Zero-bit Message Channels

At the level of abstraction chosen for design and proving correctness, all components of the TS-Algs are digital logic blocks and therefore deal with binary signals only. Given such a signal $\mathcal{SIG}$ (or an arbitrary boolean predicate), with possible values $\perp$ (logical 0, false, inactive) and $\top$ (logical 1, true, active), we say that the event (i.e., state transition) $\mathcal{SIG}{-}\!\uparrow(t^*)$ resp. $\mathcal{SIG}{-}\!\downarrow(t^*)$ occurs when $\mathcal{SIG}$ changes state from $\perp$ to $\top$ resp. from $\top$ to $\perp$ at time t^*. The status $\mathcal{SIG}(t)$ of $\mathcal{SIG}$ at time $t \geq t^*$ is $\mathcal{SIG}(t) = \perp$ resp. $\mathcal{SIG}(t) = \top$ iff the last event at or before t was $\mathcal{SIG}{-}\!\downarrow(t^*)$ resp. $\mathcal{SIG}{-}\!\uparrow(t^*)$. This at-least-once semantics of similar events, automatically provided by the physical signals, will be helpful later when proving correctness. The name "zero-bit" originates from the fact that events at $\mathcal{SIG}$ have to occur strictly alternating. Therefore no information can be transported by a single signal $\mathcal{SIG}$ except the time of occurrence.

In the following analysis we will reason about events and status of binary signals [and predicates]. In order not to clutter notation the following conventions will be employed: Depending on the context of usage $\mathcal{SIG}(t)$ will denote either

- $\mathcal{SIG}$'s status $\mathcal{SIG}(t) \in \{\perp, \top\}$, where t denotes the observation time, or

- the event $\mathcal{SIG}{-}\!\uparrow(t)$, where t denotes the time of the last transition to the active state $\top$ (or reset time).

Note also that the time t sometimes will be dropped from events and status if it is clear from the context.
All system components are interconnected by signal wires, which are modeled as reliable FIFO channels with finite delays that carry zero-bit messages. The semantics of a zero-bit message channel X is as follows: Let X^s be the channel's input signal, which is controlled by a single sender component. It generates the events (i.e., messages) $X^s{-}\!\uparrow(t)$ and $X^s{-}\!\downarrow(t)$, where t denotes the sending time. The associated input state $X^s(t)$ can be viewed as the information content of the last message sent into X. Channel X's output is fed to the receiver, which perceives the event (i.e., message) $X^r{-}\!\uparrow(t')$ resp. $X^r{-}\!\downarrow(t')$ for every send event $X^s{-}\!\uparrow(t)$ resp. $X^s{-}\!\downarrow(t)$ within finite time $t' \geq t$. For analysis purposes, we assume that there is a lower and an upper bound on the channel delay $t' - t$, but those bounds are unknown to the algorithm. The receiving state $X^r(t)$ at time t can again be viewed as the information content of the last received message. At reset time t_0, the channel states $X^r(t_0)$ and $X^s(t_0)$ are initialized to $\perp$.
Obviously, a zero-bit message channel can only convey messages with strictly alternating content. Nevertheless, this type of communication is compatible with Lamport's happened-before "$\rightsquigarrow$" relation [19]: For matching send and receive events, it holds that $X^s{-}\!\uparrow(t) \rightsquigarrow X^r{-}\!\uparrow(t')$ and $X^s{-}\!\downarrow(t) \rightsquigarrow X^r{-}\!\downarrow(t')$. To simplify the notation when using a channel X, we employ the convention that $X{-}\!\uparrow(t)$ and $X{-}\!\downarrow(t)$ abbreviate the send events $X^s{-}\!\uparrow(t)$ and $X^s{-}\!\downarrow(t)$, respectively, whereas $X(t)$ abbreviates the state $X^r(t)$ at the receiving end.

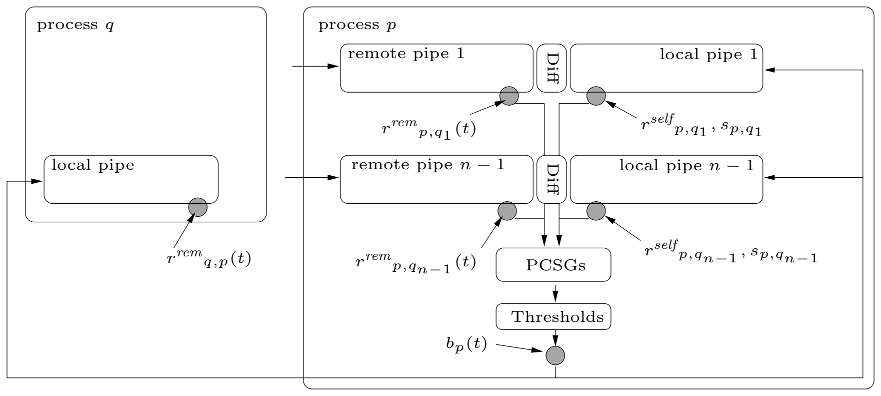


Figure 4: Schematic of a TS-Alg, demonstrating the points of observation $b_p(t)$, $r^{rem}_{p,q}(t)$ and $r^{self}_{p,q}(t)$ used in the algorithm's analysis.

3.2 TS-Alg Architectural Design

Fig. 4 shows a schematic of a TS-Alg executed by process p at the level of digital components. It has been obtained by adopting and refining the components presented in Fig. 3 to the peculiarities of digital hardware design. Note that the core part of the high-level description of the original TS-Alg, the $+/-$ counter, comprises the first three items listed below.

Pairs of elastic pipes: Every TS-Alg/process p incorporates $n-1$ pairs of elastic pipelines. An elastic pipeline is a shift register/FIFO for signal transitions [32], see Section 5.1 for a short description. Each pair of pipelines corresponds to a single remote TS-Alg/process $q \neq p$. Every pair consists of a remote pipeline that can store up to S *tick-*$\uparrow$/ *tick-*$\downarrow$ messages sent by q, and a local pipeline that can hold up to S *tick-*$\uparrow$/ *tick-*$\downarrow$ messages sent by p locally. The number S is an implementation parameter that has to be chosen in accordance with Theorem 4.17.

For description and analysis purposes some more notation is introduced: $r^{rem}_{p,q}(t)$ resp. $r^{self}_{p,q}(t)$ denotes the number of messages that arrived at the end of the remote, resp., local pipe by time t. Moreover, $s_{p,q}(t)$ denotes the number of tick messages stored in the local pipeline at time t. Note carefully that those quantities are not available to the algorithm. Rather, the algorithm uses only the binary status signals $r_{p,q}^{rem}(t) \geq r_{p,q}^{self}(t)$ and $r_{p,q}^{rem}(t) > r_{p,q}^{self}(t)$ in conjunction with $s_{p,q}(t) = 1$. Upon reset, all pipelines are initialized to contain a single even *tick*-$\downarrow$ message. This can be formalized by assuming $r_{p,q}^{rem}(t_{0,p}) = r_{p,q}^{self}(t_{0,p}) = 0$ and $s_{p,q}(t_{0,p}) = 1$ at reset time $t_{0,p}$.

Diff-Gate: To avoid the need for infinite space, which would be conceptually required in the algorithm listed in Fig. 2 in Section 2.1 as k is ever increasing, each pair of pipelines is equipped with a special circuit that removes tick messages contained in both pipes. The behavior of a Diff-Gate is as follows: If $[r_{p,q}^{rem}(t) \geq r_{p,q}^{self}(t)] \wedge [s_{p,q}(t) > 1]$, there is some $t' \in t + [\tau_{Diff}{}^{-}, \tau_{Diff}{}^{+}]$ such that $s_{p,q}(t') = s_{p,q}(t' - dt) - 1$, for some infinitesimally small $dt > 0$. Note that the TS-Alg is not losing information due to this removal of "common" tick messages, since the algorithm is only interested in the difference of the number of messages received in a pair of pipes.

Pipe Compare Signal Generators (PCSGs): There exists a dedicated detection circuit for each pair of pipes which generates the status signals $GEQ_{p,q}^{o/e}(t)$ and $GR_{p,q}^{o/e}(t)$. In particular, $GEQ_{p,q}^{o}(t')$ becomes active (i.e., $GEQ_{p,q}^{o}(t') = \top$, thereby generating the event $GEQ_{p,q}^{o}{-}\uparrow$ at t', if the previous state was $\perp$) at some time $t' \in t + [\tau_{GEQ}{}^{-}; \tau_{GEQ}{}^{+}]$ when

(i) $r_{p,q}^{self}(t) \in \mathbb{N}_{odd}$ and

(ii) $[r_{p,q}^{rem}(t) \geq r_{p,q}^{self}(t)] \wedge [s_{p,q}(t) = 1]$

Similarly, $GR_{p,q}^{o}(t')$ becomes active at time $t' \in t + [\tau_{GR}{}^{-}; \tau_{GR}{}^{+}]$ when

(i) $r_{p,q}^{self}(t) \in \mathbb{N}_{odd}$ and

(ii) $[r^{rem}_{p,q}(t) > r^{self}_{p,q}(t)] \wedge [s_{p,q}(t) = 1]$.

The signals $GEQ^{e}_{p,q}(t)$ and $GR^{e}_{p,q}(t)$ have the same definition, except that (i) reads $r^{self}_{p,q}(t) \in \mathbb{N}_{even}$ here. Note that a PCSG may become active only if $s_{p,q}(t) = 1$. The reason for this restriction is the ease of implementation in hardware. The additional term in the premise circumvents the need for large and slow comparators which would otherwise have to consider all possible states of both the remote and the local pipe.

Threshold: If the number of active $GEQ^{o/e}_{p,q}(t)$ resp. $GR^{o/e}_{p,q}(t)$ signals exceeds the $2f+1$ resp. $f+1$ threshold at time t, the corresponding threshold signal $TH^{o/e}_{GEQ}(t')$ resp. $TH^{o/e}_{GR}(t')$ becomes active at time $t' \in t + [\tau_{TH}{}^{-}; \tau_{TH}{}^{+}]$.

Tick Broadcast: The next tick message is sent when

(i) both threshold signals for the previously generated tick, say, TH^{o}_{GEQ} and TH^{o}_{GR}, are inactive again and

(ii) at least one threshold signal, TH^{e}_{GEQ} or TH^{e}_{GR}, for the current tick becomes active.

In the following analysis, $b_p(t)$ denotes the number of tick messages generated (broadcast) by process p by time t, with $b_p(t_0) = 0$ at reset time t_0. The value of $b_p(t)$ at the time of sending a tick is defined as:

Definition 3.1. *If t_k denotes the time when process p generates tick k, it is defined that $b_p(t_k) = b_p(t_k - dt) + 1$ for an infinitesimally small $dt > 0$, i.e., $b_p(t_k)$ gives the number of ticks including the new tick k.*

The detailed description of the TS-Alg based on the architectural model is given in Fig. 5. Note that the algorithm's if-clauses are only evaluated when the validity of the if-clause's premise changes, i.e., upon a state transition of the enabling condition [which also happens, by convention, upon reset]. This restriction has no effect on the validity of the algorithm, because of the idempodence in sending messages of the same type and is only due to proof-technical reasons. In case of the threshold gates (`line 17` to `line 26`) it is assumed w.l.o.g. that a (possibly idempotent) output event is sent upon any change of the state of any input.

```
/* Initialization */
$\forall q : r^{rem}_{p,q}(t_{0,p}) = r^{self}_{p,q}(t_{0,p}) = 0;\ s_{p,q}(t_{0,p}) = 1$
$\forall channels\, X : X^r(t_{0,p}) = X^s(t_{0,p}) = \perp$

/* code for PCSGs at process $p$ for remote process $q$ */
if $[r^{rem}_{p,q}(t) \geq r^{self}_{p,q}(t)] \wedge [r^{self}_{p,q}(t) \in \mathbb{N}_{odd}] \wedge [s_{p,q}(t) = 1]$
    send $GEQ^o_{p,q_i}(t)$–$\uparrow$
else send $GEQ^o_{p,q_i}(t)$–$\downarrow$
if $[r^{rem}_{p,q}(t) > r^{self}_{p,q}(t)] \wedge [r^{self}_{p,q}(t) \in \mathbb{N}_{odd}] \wedge [s_{p,q}(t) = 1]$
    send $GR^o_{p,q_i}(t)$–$\uparrow$
else send $GR^o_{p,q_i}(t)$–$\downarrow$

if $[r^{rem}_{p,q}(t) \geq r^{self}_{p,q}(t)] \wedge [r^{self}_{p,q}(t) \in \mathbb{N}_{even}] \wedge [s_{p,q}(t) = 1]$
    send $GEQ^e_{p,q_i}(t)$–$\uparrow$
else send $GEQ^e_{p,q_i}(t)$–$\downarrow$
if $[r^{rem}_{p,q}(t) > r^{self}_{p,q}(t)] \wedge [r^{self}_{p,q}(t) \in \mathbb{N}_{even}] \wedge [s_{p,q}(t) = 1]$
    send $GR^e_{p,q_i}(t)$–$\uparrow$
else send $GR^e_{p,q_i}(t)$–$\downarrow$

/* code for thresholds at process $p$ */
if $GEQ^o_{p,q_i}(t)$ for at least $2f+1$ remote processes $q_i$
    send $TH^o_{GEQ}$–$\uparrow$
else send $TH^o_{GEQ}$–$\downarrow$
if $GR^o_{p,q_i}(t)$ for at least $f+1$ remote processes $q_i$
    send $TH^o_{GR}$–$\uparrow$
else send $TH^o_{GR}$–$\downarrow$
if $GEQ^e_{p,q_i}(t)$ for at least $2f+1$ remote processes $q_i$
    send $TH^e_{GEQ}$–$\uparrow$
else send $TH^e_{GEQ}$–$\downarrow$
if $GR^e_{p,q_i}(t)$ for at least $f+1$ remote processes $q_i$
    send $TH^e_{GR}$–$\uparrow$
else send $TH^e_{GR}$–$\downarrow$

/* code for sending tick messages at process $p$ */
if $[TH^o_{GR}(t) \vee TH^o_{GEQ}(t)] \wedge \neg[TH^e_{GR}(t) \vee TH^e_{GEQ}(t)]$
    send $tick$-$\downarrow$
if $[TH^e_{GR}(t) \vee TH^e_{GEQ}(t)] \wedge \neg[TH^o_{GR}(t) \vee TH^o_{GEQ}(t)]$
    send $tick$-$\uparrow$
```

Figure 5: TS-Alg tick generation algorithm for process p after being adopted for VLSI implementation.

4 Correctness Proofs

In this section we define an appropriate system and failure model. We specify the level of abstraction necessary for the analysis by defining the components and the possibilities of their correct and incorrect interaction. After that we will proceed with detailed correctness proofs of both global and local properties of the considered system. The resulting properties can then be used as design guidelines and guaranteed performance metrics for any particular hardware implementation.

4.1 System and Failure Model

For the correctness proof and performance analysis, the following system and failure model is employed: Let P be a set of n distributed processes, executing TS-Algs, which communicate via simulated broadcasting (multiple point-to-point sends) over a fully connected network of reliable zero-bit FIFO message passing links. Every link carries strictly alternating *tick*-$\uparrow$ and *tick*-$\downarrow$ messages only. The transmission delay satisfies some upper and lower bounds, which are unknown to the algorithm, i.e., introduced solely for analysis purposes: The time of a locally generated tick message to reach the end of any local pipeline is bounded by $[{\tau_{loc}}^-; {\tau_{loc}}^+]$, whereas the time to reach the end of any remote pipeline, at any remote process, is bounded by $[{\tau_{rem}}^-; {\tau_{rem}}^+]$. Note that this assumption has some impact on initialization as well:
Every process p can be initialized to $b_p(t_{0,p}) = r^{rem}_{p,q}(t_{0,p}) = r^{self}_{p,q}(t_{0,p}) = 0$ and $s_{p,q}(t_{0,p}) = 1$ even at (slightly) different reset times $t_{0,p} \in [0; {\tau_{rem}}^-)$: The lower delay bound ${\tau_{rem}}^-$ ensures that no tick message can be lost during reset. The latencies $[{\tau_{GEQ}}^-; {\tau_{GEQ}}^+]$, $[{\tau_{GR}}^-; {\tau_{GR}}^+]$, $[{\tau_{Diff}}^-; {\tau_{Diff}}^+]$ and $[{\tau_{TH}}^-; {\tau_{TH}}^+]$ have already been introduced in Section 3.2.
The TS-Alg will allow up to f processes to fail arbitrarily (so called Byzantine failures, i.e., no assumption can be made for process failures), provided that $n \geq 3f + 2$. This is slightly more than the required lower bound of

$n \geq 3f + 1$ for clock synchronization [9], but ensures better precision and accuracy. In the context of TS-Algs, the adverse power of arbitrary failures lies in the ability of a process to generate wrong (early or even spurious) clock ticks, which are perceived inconsistently at different receiver processes. Such failures may be the consequence of particle hits or electromagnetic interference, which can very well affect different receivers differently, depending upon wire length and signal detection sensitivity. Less severe faults comprise crash failures which can occur due to stuck-at faults in hardware, for example.

4.2 Concept of Direct Causality

The formal treatment of the TS-Alg will be started with Definition 4.1, which will be employed frequently in the following proofs.

Definition 4.1. (Direct Causality). *Let $I(t')$ and $O(t)$ be two events of some specific signal input and output, respectively, of a correct component C. Then $I(t')$ and $O(t)$ are* directly causally related*, denoted by $I(t') \rightarrow O(t)$, if*

(i) they are causally related, i.e. $I(t') \rightsquigarrow O(t)$, and

(ii) there is neither an $\uparrow$- nor a $\downarrow$-event $I'(t'')$ on the same input in between, i.e., $\nexists I'(t'') : I(t') \rightsquigarrow I'(t'') \rightsquigarrow O(t)$.

If the input and output events $I(t')$ and $O(t)$ of a correct component C with latency $\in [\tau_C^-, \tau_C^+]$ are directly causally related, then $\tau_C^- \leq t - t' \leq \tau_C^+$.

An instrumental part in the correctness proof of the tick generation algorithm TS-Alg in Fig. 5 is to make sure that a process generates tick $k+1$ messages "based on" tick k messages only. The notion of "based on" is precisely defined in Definition 4.2 and 4.3. Informally it means that a process p does not incorporate stale information from earlier ticks $\ell < k$ when sending tick number $k + 1$. Lemma 4.4 below will deduce some simple properties from these definitions.

Definition 4.2. (Notion of Basis on remote process q's tick ℓ). *Abbreviate*

$$\begin{aligned}\mathcal{PC}_{p,q}^{GEQ,\ell}(t) &= [r_{p,q}^{rem}(t) \geq r_{p,q}^{self}(t) = \ell] \wedge [s_{p,q}(t) = 1] \textit{ and} \\ \mathcal{PC}_{p,q}^{GR,\ell}(t) &= [r_{p,q}^{rem}(t) > r_{p,q}^{self}(t) = \ell] \wedge [s_{p,q}(t) = 1]. \qquad (1)\end{aligned}$$

Then a correct process p's tick $k+1$ is based on correct process q's tick ℓ, *if there exists at least one of the following chains of direct causal dependencies: For $k+1 \in \mathbb{N}_{even}$,*

$$\begin{aligned}\mathcal{PC}_{p,q}^{GEQ,\ell}(t'') &\rightarrow GEQ_{p,q}^{o}(t') \rightarrow TH_{GEQ}^{o}(t'_{k+1}) \rightarrow [b_p(t_{k+1}) = k+1] \quad (2) \\ \mathcal{PC}_{p,q}^{GR,\ell}(t'') &\rightarrow GR_{p,q}^{o}(t') \rightarrow TH_{GR}^{o}(t'_{k+1}) \rightarrow [b_p(t_{k+1}) = k+1] \quad (3)\end{aligned}$$

(and analogous for $k+1 \in \mathbb{N}_{odd}$).

Definition 4.3. (Notion of Basis on tick ℓ). *A correct process p's tick $k+1$ is said to be* based on *tick ℓ, if, for all correct processes q, it is based on q's tick ℓ_q with $\ell_q \geq \ell$ and $\exists q_i : \ell_{q_i} = \ell$, i.e. $\ell = \min(\ell_{q_i})$.*

Lemma 4.4. *The time instants t'_{k+1} and t'' in the direct causal chains (2) and (3) of Definition 4.3 satisfy $\tau_{TH}{}^{-} + \min(\tau_{GR}{}^{-}, \tau_{GEQ}{}^{-}) \leq t'_{k+1} - t'' \leq \tau_{TH}{}^{+} + \max(\tau_{GR}{}^{+}, \tau_{GEQ}{}^{+})$. Moreover, the predicate $\mathcal{PC}_{p,q}^{GEQ,\ell}(t)$ resp. $\mathcal{PC}_{p,q}^{GR,\ell}(t)$ holds true for every $t \in [t'', t_{k+1} - \tau_{TH}{}^{+} - \tau_{GEQ}{}^{+}]$ resp. $t \in [t'', t_{k+1} - \tau_{TH}{}^{+} - \tau_{GR}{}^{+}]$, provided those time intervals are non-empty.*

Proof. The first statement of Lemma 4.4 is a trivial consequence of Definition 4.1. Now we will prove that the predicates continue to hold true during the given intervals. We will first show the proof for $k \in \mathbb{N}_{odd}$.
Recall from the algorithm in Fig. 5 that tick $k+1$ is sent at time t_{k+1}, i.e., $b_p(t_{k+1}) = k+1$, iff $[TH_{GR}^{o}(t_{k+1}) \vee TH_{GEQ}^{o}(t_{k+1})] \wedge \neg[TH_{GR}^{e}(t_{k+1}) \vee TH_{GEQ}^{e}(t_{k+1})] \wedge b_p(t_{k+1} - dt) = k \in \mathbb{N}_{odd}$ according to Definition 3.1 where $dt > 0$ is infinitesimally small: At least one of the threshold gates responsible for the even tick $k+1$ must be active, while both threshold gates for the

previous odd tick k must be inactive. Furthermore, $b_p(t_{k+1} - dt) = k \in \mathbb{N}_{odd}$ must obviously hold for generating tick $k+1$.
Since $b_p(t_{k+1}) = k+1$ occurs at t_{k+1}, all enabling conditions are met. In particular, TH^o_{GEQ} resp. TH^o_{GR} cannot have further changed state within the time interval $(t'_{k+1}, t_{k+1}]$ by delimitation of direct causality. It is now shown, that the predicates (1) must also maintain the same state for some time. Suppose, by way of contradiction, that e.g. $\mathcal{PC}^{GEQ,\ell}_{p,q}(t)$ reverted to false at some time t with $t'' < t \le t_{k+1} - {\tau_{TH}}^+ - {\tau_{GEQ}}^+$. Then, the GEQ^o threshold gate must have generated another event $TH^o_{GEQ}(t''_{k+1})$ (maybe an idempotent one) at time $t''_{k+1} = t + {\tau_{GEQ}}^+ + {\tau_{TH}}^+ \le t_{k+1}$ at latest, and clearly $TH^o_{GEQ}(t''_{k+1}) \to b_p(t_{k+1}) = k+1$. This contradicts direct causality of $TH^o_{GEQ}(t'_{k+1}) \to b_p(t_{k+1}) = k+1$, however.
The proof for $k \in \mathbb{N}_{odd}$ is analogous. □

4.3 Interlocking

Now the major Lemma 4.6 can be proved, stating that every correct process p's tick $k+1$ is based on tick k only, provided that the following Constraint 4.5 is satisfied. If this is the case, there is no danger of mixing up new and old instances of the signals GR^o, GEQ^o, etc.

Constraint 4.5. (Interlocking Constraint). *With the abbreviations*

$$\begin{aligned} D^+ &= {\tau_{TH}}^+ + \max({\tau_{GR}}^+, {\tau_{GEQ}}^+) + {\tau_{loc}}^+ \text{ and} \\ D^- &= {\tau_{TH}}^- + \min({\tau_{GR}}^-, {\tau_{GEQ}}^-) + {\tau_{loc}}^- + {\tau_{Diff}}^- \end{aligned} \tag{4}$$

it must hold that $D^+ \le D^- + D^-_{dis}$*, where* $D^-_{dis} = D^- - {\tau_{Diff}}^-$*.*

Lemma 4.6. (Interlocking). *Provided that Constraint 4.5 holds, every correct process p's tick $k+1$ is based on tick k.*

Proof. The proof is by induction on the number of tick messages sent by a correct process p.

Induction basis: Tick 1 is based on tick 0 [established upon reset] only. Tick 2 can be based on tick 1 only, since even ticks are solely generated from GR^o and GEQ^o—and hence from odd ticks—in the algorithm of Fig. 5.
Induction step: Suppose that all ticks m up to tick $k \geq 2$ are based on tick $m-1$, but that tick $k+1$ generated by process p is not based on tick k but rather on some tick $\ell \in \{k-2, k-4, \dots\}$. Assuming w.l.o.g. $k \in \mathbb{N}_{odd}$, there must be some correct process q_i satisfying (2) and/or (3), say, $\mathcal{PC}^{GEQ,\ell}_{p,q_i}(t''_{q_i}) \to GEQ^o_{p,q_i}(t'_{q_i}) \to TH^o_{GEQ}(t'_{k+1}) \to b_p(t_{k+1}) = k{+}1$ according to Definition 4.3. Lemma 4.4 reveals that $\mathcal{PC}^{GEQ,\ell}_{p,q_i}(t') = [r^{rem}_{p,q_i}(t') \geq r^{self}_{p,q_i}(t') = \ell] \wedge [s_{p,q_i}(t') = 1]$ must evaluate to true also at time t' with $t_{k+1} - t' \leq {\tau_{TH}}^+ + {\tau_{GEQ}}^+ \leq {\tau_{TH}}^+ + \max({\tau_{GR}}^+, {\tau_{GEQ}}^+)$.
Since $\mathcal{PC}^{GEQ,\ell}_{p,q_i}(t')$ and thus $r^{self}_{p,q_i}(t') = \ell$, tick $k-1$ must have been received at process p's local pipe corresponding to q_i at time $t'' > t'$ and it must have been sent by process p at some time $t_{k-1} \geq t'' - {\tau_{loc}}^+$. It follows that

$$t_{k-1} > t_{k+1} - {\tau_{TH}}^+ - \max({\tau_{GR}}^+, {\tau_{GEQ}}^+) - {\tau_{loc}}^+ = t_{k+1} - D^+. \qquad (5)$$

Since tick $k-1$ arrives at any local pipe at $\tilde{t} \geq t_{k-1} + {\tau_{loc}}^-$ earliest, $[r^{self}_{p,q}(\tilde{t}) = k-1] \wedge [s_{p,q}(\tilde{t}) = 1]$ is obtained not before $\tilde{t} + {\tau_{Diff}}^-$, as the previous tick $k-2$ must be removed from the local pipe before $s_{p,q}(\cdot) = 1$ can become true. Due to the induction hypothesis, tick k is based on tick $k-1$ only. Lemma 4.4 hence implies that tick k cannot be sent before

$$t_k \geq \tilde{t} + {\tau_{Diff}}^- + \min({\tau_{GR}}^-, {\tau_{GEQ}}^-) + {\tau_{TH}}^- \geq t_{k-1} + D^-. \qquad (6)$$

Recall that p can only send tick $k+1$ at t_{k+1} if $\neg[TH^e_{GR}(t_{k+1}) \vee TH^e_{GEQ}(t_{k+1})]$ is true, i.e., when all threshold gates that generated the previous tick k are inactive at t_{k+1}. Process p generates tick k when sufficiently many correct processes q_j satisfy (2) and/or (3), i.e., contribute to TH^o_{GEQ} and/or TH^o_{GR}. The latter signals can only be disabled at t_{k+1} if there exists at least one correct process q among those, for which

$$\begin{aligned} &\neg\mathcal{PC}_{p,q}^{GEQ,k-1}(\hat{t}_q) = \\ \neg\{[r_{p,q}^{rem}(\hat{t}_q) \geq r_{p,q}^{self}(\hat{t}_q)] \wedge [r_{p,q}^{self}(\hat{t}_q) = k-1] \wedge [s_{p,q}(\hat{t}_q) = 1]\} &\rightarrow \\ \neg GEQ_{p,q}^{e}(t') \rightarrow \neg\, TH_{GEQ}^{e}(t_{k+1}) & \\ \neg\mathcal{PC}_{p,q}^{GR,k-1}(\hat{t}_q) = & \\ \neg\{[r_{p,q}^{rem}(\hat{t}_q) > r_{p,q}^{self}(\hat{t}_q)] \wedge [r_{p,q}^{self}(\hat{t}_q) = k-1] \wedge [s_{p,q}(\hat{t}_q) = 1]\} &\rightarrow \\ \neg GR_{p,q}^{e}(t') \rightarrow \neg\, TH_{GR}^{e}(t_{k+1}) & \end{aligned}$$

holds at some time $\hat{t}_q$, with $t_k < \hat{t}_q$ and $\hat{t}_q + \min(\tau_{GR}{}^{-}, \tau_{GEQ}{}^{-}) + \tau_{TH}{}^{-} \leq t_{k+1}$. $\mathcal{PC}_{p,q}^{GEQ,k-1}(\cdot)$ can only become false if tick k has arrived at the local queue corresponding to q. Thus, $t_k + \tau_{loc}{}^{-} \leq \hat{t}_q$ and hence

$$t_{k+1} \geq t_k + \min(\tau_{GR}{}^{-}, \tau_{GEQ}{}^{-}) + \tau_{TH}{}^{-} + \tau_{loc}{}^{-} = t_k + D_{dis}^{-}. \tag{7}$$

Combining the results on t_{k+1} and t_k given in (6) and (7), we obtain $t_{k+1} \geq t_{k-1} + D^{-} + D_{dis}^{-}$. Substituting the result for t_{k-1} from (5), gives $t_{k+1} > t_{k+1} - D^{+} + D^{-} + D_{dis}^{-}$ and hence $D^{+} > D^{-} + D_{dis}^{-}$, which contradicts Constraint 4.5. Therefore, tick $k+1$ cannot be based on $\ell < k$, which was to be shown. $\square$

4.4 Local and Global Performance Metrics

In this section a sequence of simple lemmas which are needed for establishing the major results Theorem 4.13 (precision) and Theorem 4.14 (accuracy) will be provided. These results will lead to the most important measures about local system properties: The terms accuracy and precision have been coined in the context of clock synchronization where accuracy establishes the deviation of a clock w.r.t. Newtonian real-time and precision determines the mutual difference of any two correct clocks at a given point in real-time.

Translated into our problem, accuracy determines the maximum and minimum local clock frequency and precision gives the maximum phase difference between any two local clocks.

Lemma 4.7. (Maximum Frequency). *No correct process can send alternating tick messages with a higher frequency than* $1/D^-$.

Proof. Assume that a correct process p sends tick $k+1 \in \mathbb{N}_{odd}$ at time t_{k+1}. Because of Lemma 4.6, tick $k+1$ can only be based on tick number k. This means that $\mathcal{PC}^{GEQ,k}_{p,q_i}(t'_{q_i})$ must have been true by time $t_{k+1} - \tau_{TH}{}^{-} - \min(\tau_{GR}{}^{-}, \tau_{GEQ}{}^{-})$ simultaneously for at least $f+1$ resp. $2f+1$ processes q_i. Thus p must have sent tick k by time $t_k \leq t_{k+1} - \tau_{TH}{}^{-} - \tau_{Diff}{}^{-} - \min(\tau_{GR}{}^{-}, \tau_{GEQ}{}^{-}) - \tau_{loc}{}^{-} \leq t_{k+1} - D^-$, where $\tau_{loc}{}^{-}$ accounts for the minimum delay to reach the end of a local pipe and $\tau_{Diff}{}^{-}$ is the minimal time for removing the previous tick $k-1$ to achieve $s_{p,q_i}(t'_{q_i}) = 1$. Hence, $t_{k+1} - t_k \geq D^-$ as asserted. □

Lemma 4.8. *The first correct process that sends tick* $k+1 \geq 1$ *by time* t *must do so via* TH^o_{GEQ} *if* $k \in \mathbb{N}_{odd}$ *and* TH^e_{GEQ} *if* $k \in \mathbb{N}_{even}$.

Proof. Let $k \in \mathbb{N}_{odd}$ and assume, by contradiction, that the first correct process p does not send tick $k+1$ via $TH^o_{GEQ}(t_{k+1})$. The only other possibility to send tick number $k+1$ is via $TH^o_{GR}(t_{k+1})$. By Lemma 4.6, there exists a time $t'_{q_i} < t_{k+1}$ where $\mathcal{PC}^{GR,k}_{p,q_i}(t'_{q_i})$ holds simultaneously for at least $f+1$ remote processes $q_i \in Q \subseteq P \setminus \{p\}$, i.e., for at least one correct remote process $q \neq p$. For q, $b_q(t'_q) > k$ must hold, too, because there cannot be more messages received by p than there were sent by q, contradicting that p is the first correct process that sends tick k. The proof for $k+1 \in \mathbb{N}_{odd}$ is analogous. □

Theorem 4.9 and its proof are a generalization of the well-known consistent broadcasting results of [31, 33, 29]. The theorem will be used later on, to derive bounds on the accuracy and precision.

Theorem 4.9. (Synchronization Properties). *The algorithm satisfies the synchronization properties Progress (P), Unforgeability (U) and Simultaneity (S) if Constraint 4.5 is satisfied and $n \geq 3f+2$.*

(P) Progress: *If all correct processes send tick k by time t, then every correct process sends at least tick $k+1$ by time $t+T^+$, with $T^+ = {\tau_{Diff}}^+ + \max({\tau_{rem}}^+, {\tau_{loc}}^+) + {\tau_{GEQ}}^+ + {\tau_{TH}}^+$.*

(U) Unforgeability: *If no correct process sends tick k by time t, then no correct process sends tick $k+1$ by time $t + T^-_{first}$, with $T^-_{first} = {\tau_{rem}}^- + {\tau_{GEQ}}^- + {\tau_{TH}}^- + {\tau_{Diff}}^-$.*

(S) Simultaneity: *If some correct process sends tick k by time t, then every correct process sends at least tick k by time $t + T_{sim}$, with $T_{sim} = T^+ + ({\tau_{TH}}^+ - {\tau_{TH}}^-) + ({\tau_{Diff}}^+ - {\tau_{Diff}}^-) + ({\tau_{GR}}^+ - {\tau_{GEQ}}^-) + \varepsilon_{rem}$, where $\varepsilon_{rem} = {\tau_{rem}}^+ - {\tau_{rem}}^-$.*

Proof. The properties Progress (P), Unforgeability (U) and Simultaneity (S) will be shown one after the other:

Progress (P): Assume that all correct processes (at least $2f+2$) sent tick $k \in \mathbb{N}_{odd}$ by time t. Now focus on a correct process p: If p has already sent tick $k+1$ by time t, we are done. If it has not, $b_p(t) = k$ must hold at p. Furthermore, $b_{q_i}(t) \geq k$ must hold for at least $2f+1$ correct remote processes $q_i \in Q \subseteq P \setminus \{p\}$. By time $t' \leq t + \max({\tau_{rem}}^+, {\tau_{loc}}^+)$ all remote and local tick messages with number k must have been received at p, resulting in $r^{rem}_{p,q_i}(t') \geq b_{q_i}(t) \geq k$ and $r^{self}_{p,q_i}(t') \geq b_p(t)$. Now assume that $r^{self}_{p,q_i}(t') > k$. Because there cannot be more messages received by p than there were sent by p, process p must have sent tick $k+1$ by time t' and we are done. So assume that $r^{self}_{p,q_i}(t') = k$. Then, $r^{rem}_{p,q_i}(t') \geq r^{self}_{p,q_i}(t') = k$ must hold for at least $2f+1$ correct processes q_i. Therefore, $\mathcal{PC}^{GEQ,k}_{p,q_i}(t'')$ must hold at time $t'' \leq t' + {\tau_{Diff}}^+$ if still $r^{self}_{p,q_i}(t'') = k$ (otherwise, p must have sent tick $k+1$ by t'' and we are done). This causes at least $2f+1$ $GEQ^o_{p,q_i}{-}\uparrow$ messages to be sent by time $t'' + {\tau_{GEQ}}^{+}$. Thus p will send tick $k+1$ by time $t''' \leq t' + {\tau_{\mathit{Diff}}}^{+} + {\tau_{GEQ}}^{+} + {\tau_{TH}}^{+} \leq t + T^{+}$. The proof for $k \in \mathbb{N}_{even}$ is analogous.

Unforgeability (U): Let $k \in \mathbb{N}_{odd}$ and assume that a correct process p has sent tick $k+1$ at time t_{k+1}, which can only happen if $\mathit{TH}^{o}_{GEQ}(t_{k+1})$ or $\mathit{TH}^{o}_{GR}(t_{k+1})$ is active.

1. Assume that tick $k+1$ was sent by process p because $\mathit{TH}^{o}_{GEQ}(t_{k+1})$ was active. Thus $\mathit{TH}^{o}_{GEQ}{-}\!\uparrow$ must have been sent at time $t' \leq t_{k+1} - {\tau_{TH}}^{-}$. Furthermore, Lemma 4.6 (Interlocking) states that $r^{rem}_{p,q_i}(t''_{q_i}) \geq r^{self}_{p,q_i}(t''_{q_i}) = k \wedge s_{p,q_i}(t''_{q_i}) = 1$ must have been true at time $t''_{q_i} \leq t' - {\tau_{GEQ}}^{-}$ for at least $2f+1$ remote processes q_i. Among those, there must have been at least $f+1$ remote correct processes $q_j \in Q \subseteq P \setminus \{p\}$, all of which must have sent tick k by time $t''' \leq t_{k+1} - {\tau_{TH}}^{-} - {\tau_{GEQ}}^{-} - {\tau_{\mathit{Diff}}}^{-} - {\tau_{rem}}^{-}$.

2. Assume tick $k+1$ was sent by process p because $\mathit{TH}^{o}_{GR}(t_{k+1})$ was active. By similar arguments as above, $\mathcal{PC}^{GR,k}_{p,q_i}(t''_{q_i})$ must have been true by time $t''_{q_i} \leq t_{k+1} - {\tau_{TH}}^{-} - {\tau_{GR}}^{-}$ for at least $f+1$ remote processes q_i, which include at least for one correct remote process q. Process q must have sent tick $k+1$ at time $t''' \leq t_{k+1} - {\tau_{TH}}^{-} - {\tau_{GR}}^{-} - {\tau_{\mathit{Diff}}}^{-} - {\tau_{rem}}^{-}$. Thus the Lemma can be applied again for q.

The proof for $k \in \mathbb{N}_{even}$ is similar. To summarize: If *no correct remote* process $q \neq p$ has sent tick k by time t, no correct process p will send tick $k+1$ by time $t + {\tau_{rem}}^{-} + {\tau_{\mathit{Diff}}}^{-} + {\tau_{GEQ}}^{-} + {\tau_{TH}}^{-}$. Since “no correct remote” process is implied by “no correct” process, we are done.

Simultaneity (S): Let p be the first correct process to send tick $k \in \mathbb{N}_{odd}$ at time t_k. By Lemma 4.8 it must do so via TH^{o}_{GEQ}. Because of Lemma 4.6 (Interlocking), $\mathcal{PC}^{GEQ,k-1}_{p,q_i}(t'_{q_i})$ must have held at time $t'_{q_i} \leq t_k - {\tau_{TH}}^{-} - {\tau_{GEQ}}^{-}$ for at least $2f+1$ remote processes, and hence for at least $f+1$ correct remote processes $\bar{q}_i \in Q \subseteq P \setminus \{p\}$. Every process $\bar{q}_i$ must have sent tick $k-1$ at time $t^{(1)}_{\bar{q}_i} \leq t_k - {\tau_{TH}}^{-} - {\tau_{GEQ}}^{-} - {\tau_{\mathit{Diff}}}^{-} - {\tau_{rem}}^{-}$. This, however, means that

tick $k-1$ from at least $f+1$ correct remote processes $\bar{q}_i$ will arrive at every correct process q_j by time $t^{(2)}_{q_j} \leq t_k - \tau_{TH}{}^- - \tau_{GEQ}{}^- - \tau_{Diff}{}^- + \varepsilon_{rem}$. Thus, at all correct processes q_j, $r^{rem}_{q_j,\bar{q}_i}(t^{(2)}_{q_j}) \geq k-1$ must be true for at least $f+1$ processes and all q_j will send tick $k-1$ by time $t^{(3)} \leq t_k - \tau_{TH}{}^- - \tau_{GEQ}{}^- - \tau_{Diff}{}^- + \varepsilon_{rem} + \tau_{Diff}{}^+ + \tau_{GR}{}^+ + \tau_{TH}{}^+$ if they have not already done so.
Assume that p is not the first correct process that sends tick number $k \in \mathbb{N}_{odd}$. Then there must be some other correct process which already sent tick k at time $t^- \leq t_k$. Now the same arguments can be applied for this process and all correct processes will send tick $k-1$ by time $t^{(3),-} \leq t^- - \tau_{TH}{}^- - \tau_{GEQ}{}^- - \tau_{Diff}{}^- + \varepsilon_{rem} + \tau_{Diff}{}^+ + \tau_{GR}{}^+ + \tau_{TH}{}^+$. Finally, Progress (P) can be applied once, stating that all correct processes must send tick k by time $t^{(4)} \leq t_k - \tau_{TH}{}^- - \tau_{GEQ}{}^- - \tau_{Diff}{}^- + \varepsilon_{rem} + \tau_{Diff}{}^+ + \tau_{GR}{}^+ + \tau_{TH}{}^+ + T^+$. □

Lemma 4.10. (Fastest Progress). *Assume that p is the first correct process that sends tick number k at time t. Then no correct process can send tick $k' > k$ before time $t + (k'-k)T^-_{first}$.*

Proof. The proof is by induction on k'. For $k' = k+1$, the first correct process $q \in P$ that sends tick $k+1$ must do so after time $t + (k'-k)T^-_{first}$ because of Unforgeability (U). Now assume that p is the first correct process that sends tick k'. It does this not before $t + (k'-k)T^-_{first}$ by the induction hypothesis. Because of (U), no other correct process can send tick $k'+1$ by time $t + (k'-k)T^-_{first} + T^-_{first} = t + (k'+1-k)T^-_{first}$. □

The following Lemma 4.11 relates the system's progress of ticks generated by the all processes to progress in real-time.

Lemma 4.11. (Maximum Increase of b^{max}). *Let $b^{max}(t)$ be the maximum of $b_p(t)$ over all correct processes p, and t^{first}_k be the time when b^{max} was increased to k, i.e. when the first process sent tick k. Define the indicator function $I_{usync}(t) = I_{t \notin \{t^{first}_0, t^{first}_1, t^{first}_2, \ldots\}}$ to be 1 if $t \notin \{t^{first}_0, t^{first}_1, t^{first}_2, \ldots\}$ and 0 otherwise. Then, for any $t_1 \leq t_2$, it holds that $b^{max}(t_2) - b^{max}(t_1) \leq \left\lfloor \frac{t_2 - t_1}{T^-_{first}} \right\rfloor + I_{usync}(t_1)$.*

Proof. For $t_1 \in \{t_0^{first}, t_1^{first}, t_2^{first}, \dots\}$, Lemma 4.10 (Fastest Progress) can be applied, which reveals that $b^{max}(t_2) - b^{max}(t_1) \leq \left\lfloor \frac{t_2 - t_1}{T^-_{first}} \right\rfloor$ as needed. For $t_1 \notin \{t_0^{first}, t_1^{first}, t_2^{first}, \dots\}$, it must hold that $t_{k-1}^{first} < t_1 < t_k^{first}$, for some k. Thus $b^{max}(t_2) - b^{max}(t_1) \leq \left\lfloor \frac{t_2 - t_k}{T^-_{first}} \right\rfloor + 1 \leq \left\lfloor \frac{t_2 - t_1}{T^-_{first}} \right\rfloor + 1$ by monotonicity of $\lfloor x \rfloor$. □

Lemma 4.12. (Slowest Progress). *Assume that some process p sends tick k at time t. Then all correct processes must send tick $k' > k$ by time $t + (k' - k)T^+ + T_{sim}$.*

Proof. The proof is by applying Simultaneity (S) once and Progress (P) iteratively. □

The following major Theorem 4.13 gives the algorithm's precision π which guarantees $\forall t : |b_q(t) - b_p(t)| \leq \pi$ for every pair of correct processes p and q.

Theorem 4.13. (Precision). *The precision $\pi \geq |b_q(t) - b_p(t)|$ of our algorithm is bounded by $\pi \leq \left\lfloor \frac{T_{sim}}{T^-_{first}} \right\rfloor + 1$.*

Proof. First of all bounds on the difference of $b^{max}(t')$ and $b_p(t')$ will be established for any t' in between p's instants of sending tick number k and $k+1$, i.e., $t_k^p \leq t' < t_{k+1}^p$.

$$b^{max}(t') \leq b_p(t') + \pi \text{ and } b^{max}(t_{k+1}^p) \leq b_p(t_{k+1}^p) + \pi - 1 \tag{8}$$

Assume that process p sends tick $k+1$ at time t_{k+1}^p. Because of (S), $t_{k+1}^p \leq t_{k+1}^{first} + T_{sim}$ must hold, and because of monotonicity, $b^{max}(t_{k+1}^p - T_{sim}) \leq b^{max}(t_{k+1}^{first}) = k+1$ must hold. According to Lemma 4.11, $b^{max}(t_{k+1}^p) \leq \left\lfloor \frac{T_{sim}}{T^-_{first}} \right\rfloor + I_{usync}(t_{k+1}^p - T_{sim}) + b^{max}(t_{k+1}^p - T_{sim})$. The two possible cases $t_{k+1}^p - T_{sim} = t_{k+1}^{first}$ and $t_{k+1}^p - T_{sim} < t_{k+1}^{first}$ both lead to $I_{usync}(t_{k+1}^p - T_{sim}) + b^{max}(t_{k+1}^p - T_{sim}) = k+1$. Thus the former inequality can be refined to

$b^{max}(t^p_{k+1}) \leq \left\lfloor \frac{T_{sim}}{T^-_{first}} \right\rfloor + k + 1$. Now, for $t^p_k \leq t' < t^p_{k+1}$, $b_p(t') = k$ and $b^{max}(t') \leq b^{max}(t^p_{k+1})$ is obtained and hence $b^{max}(t') - b_p(t') \leq \left\lfloor \frac{T_{sim}}{T^-_{first}} \right\rfloor + 1 = \pi$. For t^p_{k+1}, obviously $b_p(t^p_{k+1}) = k+1$ and hence $b^{max}(t^p_{k+1}) - b_p(t^p_{k+1}) \leq \left\lfloor \frac{T_{sim}}{T^-_{first}} \right\rfloor = \pi - 1$. The inequalities (8) can be generalized to hold at any time t, by finding an appropriate k for which $t^p_k \leq t < t^p_{k+1}$, or $t = t^p_{k+1}$ holds at p and applying (8) within these bounds. Since obviously $\forall t : b_p(t) \leq b^{max}(t)$, any two correct processes p and q must fulfill $\forall t : |b_q(t) - b_p(t)| \leq \pi$. □

The following Theorem 4.14 (accuracy) can be used to bound the number of tick messages generated locally at a correct process p during some real-time interval Δ, i.e., allows to make statements about the local frequency. For example, it reveals that the long-term frequency ($f_\infty := \lim_{\Delta\to\infty} \frac{b_p(\Delta)}{\Delta}$) is within $[1/T^+, 1/T^-_{first}]$.

Theorem 4.14. (Accuracy). *Given* $\Delta = t_2 - t_1$*, the accuracy* $|b_p(t_2) - b_p(t_1)|$ *of any correct process* p *is bounded by* $\max\left\{0, \frac{\Delta - T_{sim} - T^+}{T^+}\right\} \leq |b_p(t_2) - b_p(t_1)| \leq \left\lceil \frac{\Delta}{T^-_{first}} \right\rceil + \min\left\{\pi + 1, \left\lceil \frac{\Delta}{D^-} - \frac{\Delta}{T^-_{first}} \right\rceil\right\}$.

Proof. The upper bound for accuracy will be shown first: It is known that $\forall t : b_p(t) \geq b^{max}(t) - \pi + (1 - I_{usync}(t))$ and $\forall t : b_p(t) \leq b^{max}(t)$ from Lemma 4.13 and Lemma 4.11. Thus $b_p(t_2) - b_p(t_1) \leq b^{max}(t_2) - b^{max}(t_1) + \pi - (1 - I_{usync}(t_1))$. By applying Lemma 4.11, $b_p(t_2) - b_p(t_1) \leq \left\lfloor \frac{t_2 - t_1}{T^-_{first}} \right\rfloor + 2I_{usync}(t_1) - 1 + \pi \leq \left\lfloor \frac{t_2 - t_1}{T^-_{first}} \right\rfloor + \pi + 1 \leq \left\lceil \frac{t_2 - t_1}{T^-_{first}} \right\rceil + \pi + 1$. Moreover, from Lemma 4.7 it follows that $b_p(t_2) - b_p(t_1) \leq \left\lceil \frac{t_2 - t_1}{D^-} \right\rceil$. Hence, $b_p(t_2) - b_p(t_1) \leq \min\left\{\left\lceil \frac{\Delta}{T^-_{first}} \right\rceil + \pi + 1, \left\lceil \frac{\Delta}{D^-} \right\rceil\right\} \leq \left\lceil \frac{\Delta}{T^-_{first}} \right\rceil + \min\left\{\pi + 1, \left\lceil \frac{\Delta}{D^-} \right\rceil - \left\lceil \frac{\Delta}{T^-_{first}} \right\rceil\right\} \leq \left\lceil \frac{\Delta}{T^-_{first}} \right\rceil + \min\left\{\pi + 1, \left\lceil \frac{\Delta}{D^-} - \frac{\Delta}{T^-_{first}} \right\rceil\right\}$ since $\lceil x + y \rceil \leq \lceil x \rceil + \lceil y \rceil$.

To prove the lower bound, first define $b_1 = b_p(t_1)$, $b_2 = b_p(t_2)$ and $t^p_{b_1} \leq t_2$, $t^p_{b_2} \leq t_2$ as the points in time when p sends tick b_1 and b_2. Clearly $t^p_{b_2+1} > t_2$, and by Lemma 4.12 it follows $t_2 - t_1 \leq t^p_{b_2+1} - t^p_{b_1} \leq T_{sim} + (b_2 - b_1 + 1)T^+$. Hence, $\max\left\{0, \frac{\Delta - T_{sim} - T^+}{T^+}\right\} \leq b_p(t_2) - b_p(t_1)$. □

4.5 Design Directives

In Section 4.3, it has been shown that a system of TS-Alg units fulfilling Constraint 4.5 (Interlocking) provides a correct interlocking of tick messages, i.e., does not decide to send a tick on stale information. Based on this result, bounds on accuracy and precision have been shown.
However, it has not been proved until here that a process does not have to store infinitely many ticks from other processes, i.e., that the size of a pair of pipes can be bounded without invalidating the algorithm. The theorem presented in this section establishes that pipeline size is indeed bounded, provided that the following constraint holds:

Constraint 4.15. (Diff-Gate Constraint). *We assume that*

$$\tau_{Diff}{}^+ \leq T^-_{first}.$$

It is first shown that tick $k-1$ is completely removed from all pipelines corresponding to correct processes at most $T_{sim} + \max(\tau_{rem}{}^+, \tau_{loc}{}^+) + \tau_{Diff}{}^+$ after the first correct process sent tick k.

Lemma 4.16. *If Constraint 4.15 holds, every correct process has completely removed tick $k-1 \geq 0$ from the pair of pipelines corresponding to a correct process by time $t_k^{first} + T_{sim} + \max(\tau_{rem}{}^+, \tau_{loc}{}^+) + \tau_{Diff}{}^+$, where t_k^{first} is the time when the first correct process sent tick k.*

Proof. Abbreviating $t_k^{last} = t_k^{first} + T_{sim}$ and denoting by t_k^p resp. t_k^q the time when p resp. q sends tick k, it follows from Simultaneity (S) that $t_k^{first} \leq t_k^p, t_k^q \leq t_k^{last}$. Consequently, tick k arrives in both the local and remote pipe at process p by time $T_k = t_k^{last} + \max(\tau_{rem}{}^+, \tau_{loc}{}^+) = t_k^{first} + T_{sim} +$

$\max({\tau_{rem}}^+, {\tau_{loc}}^+)$. It is shown by induction that tick $k-1$ is removed at most ${\tau_{\mathit{Diff}}}^+$ later:

Induction basis: For $k = 0$, the statement is vacuously true since all processes "receive" tick 0, at reset time, within $[0, {\tau_{rem}}^-]$ [note that ${\tau_{rem}}^- < T_0 = T_{sim} + \max({\tau_{rem}}^+, {\tau_{loc}}^+)$] and there is of course no tick $k-1$ to be removed at all.

Induction step: Now assume that tick $k-1$ has been removed by time $T_k + {\tau_{\mathit{Diff}}}^+$ and consider the removal of tick k. Since $T_{k+1} - T_k = t_{k+1}^{first} - t_k^{first} \geq T_{first}^-$ by Lemma 4.11, $T_{k+1} \geq T_k + T_{first}^- \geq T_k + {\tau_{\mathit{Diff}}}^+$ must hold. Due to the induction hypothesis, the removal of tick k does not face blocking due to the removal of tick $k-1$. Hence, tick k must be successfully removed by $T_{k+1} + {\tau_{\mathit{Diff}}}^+$ as asserted. □

Theorem 4.17. (Queuesize). *The size S of any pipeline in a pair of pipelines corresponding to a correct remote process at a correct local process is bounded by* $S = \left\lfloor \frac{T_{sim} + \max({\tau_{rem}}^+, {\tau_{loc}}^+) + {\tau_{\mathit{Diff}}}^+}{T_{first}^-} \right\rfloor + 1$.

Proof. From Lemma 4.16, we know that tick $k-1$ is completely removed by time $T'_k = t_k^{first} + T_{sim} + \max({\tau_{rem}}^+, {\tau_{loc}}^+) + {\tau_{\mathit{Diff}}}^+$. Hence, the maximum size of any pipe is determined by the maximum number of ticks any process can generate in the interval $T_{sim} + \max({\tau_{rem}}^+, {\tau_{loc}}^+) + {\tau_{\mathit{Diff}}}^+$. According to Lemma 4.11, those are at most $S = \left\lfloor \frac{T_{sim} + \max({\tau_{rem}}^+, {\tau_{loc}}^+) + {\tau_{\mathit{Diff}}}^+}{T_{first}^-} \right\rfloor + 1$. □

5 Hardware Implementation

The following section summarizes the work primarily conducted by Gottfried Fuchs in the DARTS project and shows how the TS-Alg developed in the previous sections is finally mapped to hardware building blocks well-known in asynchronous digital design. Since the TS-Algs — and the components they comprise of — interact with alternating transitions over zero-bit message channels, it is natural to build the hardware with transition signaling logic [4].

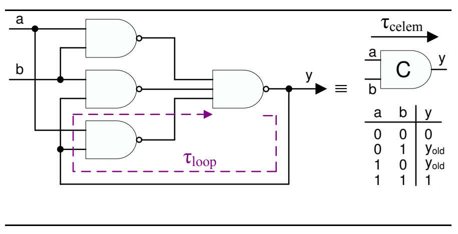
The section starts with a short overview on important design elements in a bottom up manner and then proceeds with showing how to synthesize a complete TS-Alg design out of those.

5.1 Common Building Blocks

One of the most important design elements in transition logic is the **Muller C-Element** [24]. It can be characterized informally as a logical-AND for transitions. When the last transitions that occurred at the two inputs both read $\uparrow$ resp. $\downarrow$ the Muller C-Element produces a possibly idempotent $\uparrow$ resp. $\downarrow$ transition at its output.

In state semantics, rather than transition semantics, the behavior can be formalized as: If both input states read the same value, this value is copied to the output, otherwise the output is not changed. The description in state semantics already hints at the existence of a feedback loop to store the old output value.

A common gate-level implementation of a Muller C-Element can be found in Fig. 6. From this figure one can see that the output signal has to traverse the complete feedback loop, lasting τ_{loop}, before the next transition in the opposite direction may occur at any input without invalidating the specified behavior. This behavior is not specific to this implementation but is a general property of all implementations known to the author.



a	b	y
0	0	0
0	1	y_{old}
1	0	y_{old}
1	1	1

Figure 6: A possible implementation of a Muller C-Element at gate-level.

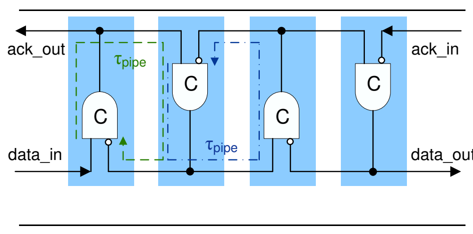


Figure 7: Elastic pipeline for up to four transitions, consisting of Muller C-Elements and inverters.

In [32], the concept of asynchronous FIFO storages, called micropipelines or **elastic pipelines**, is introduced. An elastic pipeline consists of control logic, implementing the two-phase bundled data convention protocol [32], and of a queue of latches, triggered by the control logic and storing the FIFO data. Since we only need the possibility to store zero-bit $\uparrow$ / $\downarrow$ transitions, an elastic pipeline without any latches will suffice. An implementation of such a minimized elastic pipeline can be found in Fig. 7.

The $\uparrow$ / $\downarrow$ transitions are simply fed into the pipeline via the `data_in`, and read out via the `data_out` line. Each pipeline stage comprises a Muller C-Element and an inverter and can store exactly one $\uparrow$ / $\downarrow$ transition. The stages can then be chained forming an elastic pipeline.

Note that correct behavior can only be guaranteed if the signal of a pipeline stage has propagated through the complete feedback path and has already stabilized before the next transition occurs at the input. Thus the minimal time τ_{min} between two consecutive transitions at the pipeline input must fulfill $\tau_{min} > \max(\tau_{pipe}, \tau_{loop})$, where $\tau_{pipe} = 2\tau_{celem} + \tau_{inverter} + \tau_{wires}$ is the feedback delay of a single pipeline stage.

5.2 TS-Alg Component Implementations

This section provides a possible implementation of the TS-Alg components that were given in Section 3.2. A detailed schematic of the complete TS-Alg implementation can be found in Fig. 8.

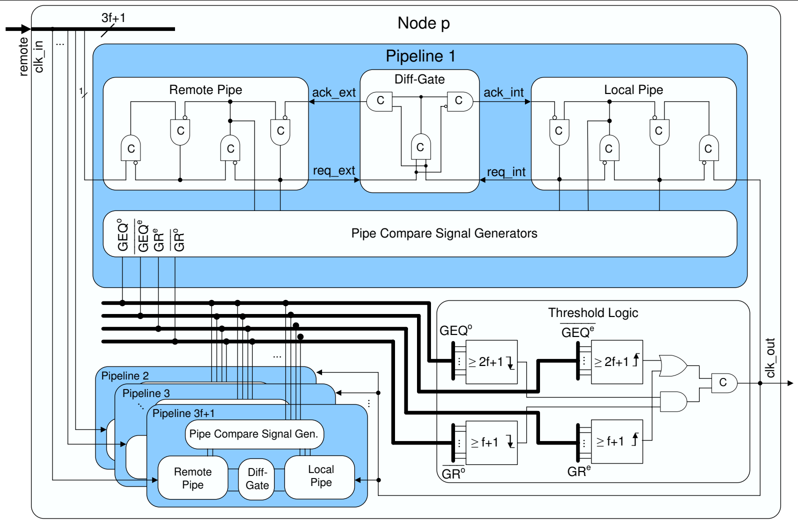


Figure 8: TS-Alg hardware implementation for a single node p.

Pairs of elastic pipes: A pair of elastic pipes simply consists of two elastic pipelines, with the `data_in` port driven by a remote (remote pipeline) resp. local clock (local pipeline). The `ack_out` port is left unused since no feedback for received clock signals is needed by the algorithm. The consumer interfaces of the elastic pipes are connected to Diff-Gates.

Note, that an implementation using elastic pipelines of size S is correct, if

(i) the clock signals fed into the remote and local pipelines fulfill: $\tau_{min} \geq D^{-} - 2\max\{(\tau_{loc}{}^{+} - \tau_{loc}{}^{-}), (\tau_{rem}{}^{+} - \tau_{rem}{}^{-})\} > \max(\tau_{pipe}, \tau_{loop})$ and

(ii) $S \geq \left\lfloor \frac{T_{sim} + \max(\tau_{rem}{}^{+}, \tau_{loc}{}^{+}) + \tau_{Diff}{}^{+}}{T^{-}_{first}} \right\rfloor + 1$, according to Theorem 4.17. Note, however, that S is almost technology independent, since it depends only on the ratio of maximum and minimum delays, similar to systems adhering to the Θ-Model [20, 34].

Diff-Gate: The Diff-Gate removes matching $\uparrow$ / $\downarrow$ transitions in a pair of elastic pipelines. It does this by implementing the two-phase bundled data convention protocol and sending acknowledges when detecting corresponding tick messages at the consumer interfaces of both pipelines. An implementation of the Diff-Gate is shown in Fig. 9. Note that the Diff-Gate is not completely symmetric. The C-Element C_4 is equipped with an inverter at one input. This small difference has the effect that matching transitions are always deleted first from the remote pipe before deleting them from the local pipe. This will play an important role when implementing the PCSG. See [13] for detailed considerations why the presented implementation does not suffer from violated timing conditions due to too close opponent transitions at the C-Element inputs.

Pipe Compare Signal Generators (PCSGs): The PCSG compares the number of $\uparrow$ / $\downarrow$ clock transitions in a pair of pipes. Hence, it acts on states rather than transitions and must be implemented using combinatorial logic instead of transition logic.

The PCSG must be completely glitch-free, i.e., it must not activate the $GR^{o/e}_{p,q}(t)$ resp. $GEQ^{o/e}_{p,q}(t)$ signals unless $[r^{rem}_{p,q}(t) > r^{self}_{p,q}(t) \in \mathbb{N}_{odd/even}] \wedge [s_{p,q}(t) = 1]$, as specified by the algorithm in Fig. 5, Section 3.2.

This is made possible only because the Diff-Gate removes matching transitions from the remote pipe first. If not, the PCSG could enable one of its outputs at $t' \in (t_1, t_2)$, although $r^{rem}_{p,q}(t_1) = r^{rem}_{p,q}(t_2) < r^{self}_{p,q}(t_1) = r^{self}_{p,q}(t_2)$, thus producing a glitch.

Another challenge is to decide whether $r^{rem}_{p,q}(t) > r^{self}_{p,q}(t)$, resp., $r^{rem}_{p,q}(t) \geq r^{self}_{p,q}(t)$ on all, including unstable, pipeline states. To circumvent this, which would, even if possible, suffer from combinatorial explosion, the clause $s_{p,q}(t) = 1$ has been added. Thus output signals are only activated if the local pipe contains exactly one transition. It has been

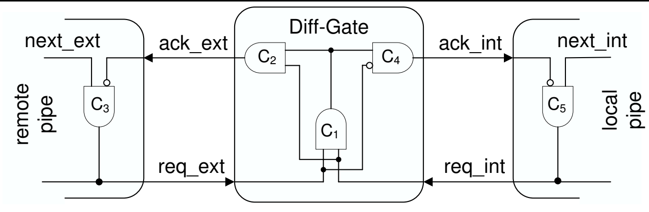


Figure 9: Diff-Gate in combination with remote and local pipeline interfaces.

shown in Section 4, that the algorithm still maintains correct behavior under this restriction. The PCSG then turns out to be implementable glitch-free with a few combinatorial gates only.

Threshold: There are four threshold gates per functional unit FU_i, two operating on the $GR^{o/e}_{p,q}$ state signals and two on the $GEQ^{o/e}_{p,q}$ state signals. The threshold gates operating on the odd signals are low-active (being responsible for generating the $\downarrow$ clock transitions), while the threshold gates operating on the even signals are high-active.

Currently, the threshold gates are implemented as ROMs, with the GR and GEQ signals forming address bus (bit-width n). Unfortunately this has certain drawbacks like the large size and slow access. Appropriate alternatives are still looked for and will be discussed briefly in Section 6.

Tick Broadcast: The two low-active Threshold gates are connected to AND-gate inputs, producing a $\downarrow$ transition at the output if at least one of the low-active gates has produced a $\downarrow$ transition.

Analogously, the two high-active Threshold gates are connected to OR-gate inputs, producing a $\uparrow$ transition at the output if at least one of the high-active gates has produced a $\uparrow$ transition.

The output of the AND and OR gates are then connected to a Muller C-Element as depicted in Fig. 8.

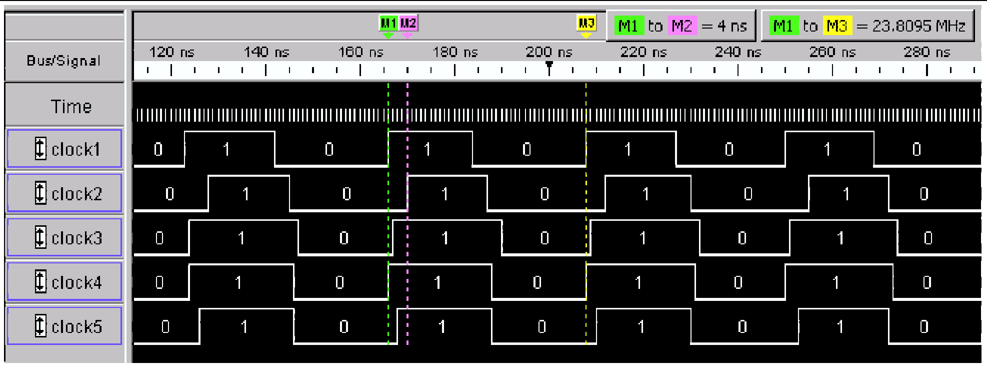


Figure 10: DARTS approach with 5 TS-Alg units running on an FPGA.

5.3 Experimental Evaluation

The TS-Algs components described in Section 5.2 have been implemented in synthesizable VHDL. Additionally a system consisting of $n = 5$ TS-Alg units has been composed out of the single units and has been loaded into an Altera APEX EP20K1000 FPGA. Note that an evaluation at an FPGA was just used as a "proof of concept", i.e., to show the general possibility of synthetisation and will be substituted by an ASIC design later on in the DARTS project.

The design was evaluated by logging the clock outputs of the 5 units with a logic analyzer. The result is shown in Fig. 10. The measurements of the clock frequency stabilized at about 24 MHz with a maximum clock skew of up to 4 ns (approximately 9.5% of total clock period).

6 Outlook

In this section, topics of current and future research addressed in the DARTS project are pointed at and shortly described.

6.1 Alternative Implementation of Threshold Gates

In Section 5.2 it has been indicated that the current implementation of the Threshold gates as ROMs is problematic. There exists numerous work on Threshold gates in literature, especially from the artificial neural network community. An up-to-date survey can be found in [3]. In the following, a short overview on alternative Threshold gate designs is given:

Combinatorial: The most apparent alternative is to implement the Threshold gate by combinatorial means. The boolean function of an $f+1$ out of $n = 3f+2$ Threshold with inputs x_i and output o would read in disjunctive normal form:

$$o \equiv \bigvee_{(x_1,\dots,x_n)\in\{0,1\}^n : \sum x_i \geq f+1} \left[\bigwedge_{x_i=1} x_i \wedge \bigwedge_{x_i=0} \overline{x}_i \right] \tag{9}$$

Unfortunately an implementation like this would consist of

$$\sum_{a=f+1}^{n} \binom{n}{a} \binom{n}{n-a}$$

clauses, if unoptimized. A first optimization would be to substitute $\sum x_i \geq f+1$ with $\sum x_i < f+1$ in (9) and afterwards negate the output o. This reduces the number of sets of clauses significantly.

Adder: Another idea is to use an adder with n single-bit inputs, or a pyramidally layered structure of adders with only 2 inputs: $\lceil n/2 \rceil$ adders with 1-bit input at the first layer, $\lceil n/4 \rceil$ adders with 2-bit input at the second layer, and so on, repeating the process for all $\lceil \mathrm{ld}(n) \rceil$ layers. Finally the result must be tested for fulfilling $\geq f+1$, resp., $\geq 2f+1$.

The propagation delay of this approach is clearly determined by the propagation delay of $\mathrm{ld}(n)$ adders, which can be quite high.

Analog Thresholds: Most of the implementations summarized in [3] make use of design elements not available in standard digital cell libraries. The general idea is the usage of analog summing amplifiers succeeded by analog thresholds. This can be achieved by different means: CMOS capacitive solutions (like the νMOS transistor [5]) and conductance based solutions (e.g. output wired inverters [21]), to name only two out of them.

6.2 Removing Transient Faults

Currently the TS-Alg units cannot distinguish between transient and permanent faults, i.e., transient faults are mapped to permanent ones. The reason is that the TS-Alg units do not maintain the complete number of tick messages received and sent so far, but do only measure the difference of received and sent messages.

A possible solution to detect transient faults would be to halt the complete system by deactivating the progress rules (`line 17` and `line 23` of the algorithm shown in Fig. 5 in Section 3.2) long enough, such that all tick messages can arrive at all pipes and no more messages are in transit. Hence, all TS-Algs must stop at the same clock tick. By comparing the number of ticks in a pair of pipes, it can be determined whether transient faults have occurred. Alternatively, the pipes can simply be reset to 0 by default.

Of course, a periodic reset — as described above — would solve the problem. Unfortunately, this means that the TS-Algs would produce a clock signal which becomes inactive for a longer time periodically.

6.3 Booting

A simple solution to the booting problem has been given in Section 4.1. It states that, if the reset times $t_{0,p}$ fulfill $\forall p, q \in P : t_{0,p} - t_{0,q} \in [0, \tau^-)$, no messages can be lost and the system behavior is correct.
However, some designs (for example if some TS-Algs reside on different chips) may impose more relaxed synchronization bounds for resetting. This task is related to the removal of transient faults and could be solved by analogous means as described in Section 6.2.

6.4 Predictability

A great advantage of classical synchronous systems is the possibility to give bounds on the system's progress of computation. But even predicting the behavior of synchronous designs seems to approach limits, e.g. when we consider pipelined designs where flushes etc. may occur.
Anyhow it is required to predict the run-time behavior of designs such that we have to evaluate the performance of our asynchronous design. We have done so in Section 4.4 where we have derived bounds on the clock frequencies of our TS-Algs and thus on the progress of the Fu_i's computations. Note, that obtaining such results for completely asynchronous designs is more challenging. However, there is still need for future work, most notable on frequency stabilization.

7 Conclusions

It has been indicated that synchronous monolithic designs are increasingly facing problems that cannot be solved satisfactorily by more advanced manufacturing processes. A conceptually new solution — currently under research in the DARTS project — has been presented: The monolithic chip design is substituted with a set of loosely coupled functional units residing on the same chip, thereby forming a System-on-Chip. The functional units are extended by small TS-Alg units, each generating a fault-tolerant clock for the functional unit it is attached to. These local clock signals are not independent of each other, but rather adhere to certain synchronization properties (bounded frequency, bounded phase difference) permitting predictability and protecting the inter-functional unit communication from setup/hold violations.
By modeling the TS-Algs as processors which communicate with each other via a special TS-Net, our SoC clock generation scheme was analyzed formally with techniques from the distributed algorithms community. By adapting some of these well established techniques to hardware design, it has been shown that a SoC comprising of an arbitrary number of functional units will generate fault-tolerant distributed clock signals fulfilling the synchronization properties precision and accuracy. Note that a comparably general result could not have been shown by means of model checking.
Finally an FPGA hardware implementation was presented, proving the principal feasibility of synthetisation and already showing promising results for the forthcoming ASIC implementation currently under development in our DARTS project.

References

[1] International technology roadmap for semiconductors, 2001.

[2] A. Bar-Noy and D. Dolev. Consensus algorithms with one-bit messages. *Distributed Computing*, 4:105–110, 1991.

[3] V. Beiu, J. M. Quintana, and M. J. Avedillo. VLSI Implementations of Threshold Logic — A Comprehensive Survey. *IEEE Transactions on Neural Networks*, 14(5):1217–1243, Sept. 2003.

[4] D. L. Black. On the existince of delay-insensitive fair arbiters: Trace theory and its limitations. *Distributed Computing*, 1:205–225, 1986.

[5] J. B. Burns and R. A. Powlus. Threshold Circuit Utilizing Field Effect Transistors. *U.S. Patent 3 260 863*, July 1966.

[6] D. M. Chapiro. *Globally-Asynchronous Locally-Synchronous Systems.* PhD thesis, Stanford University, Oct. 1984.

[7] C. Constantinescu. Impact of deep submicon technology on dependability of VLSI circuits. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'02)*, pages 205–209, June 2002.

[8] *DARTS homepage. http://www.ecs.tuwien.ac.at/projects/DARTS.*

[9] D. Dolev, J. Y. Halpern, and H. R. Strong. On the possibility and impossibility of achieving clock synchronization. *Journal of Computer and System Sciences*, 32:230–250, 1986.

[10] S. Fairbanks. Method and apparatus for a distributed clock generator, 2004. US patent no. US2004108876.

[11] S. Fairbanks and S. Moore. Self-timed circuitry for global clocking. In *Proceedings of the Eleventh International IEEE Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 86–96, Mar. 2005.

[12] E. G. Friedman. Clock distribution networks in synchronous digital integrated circuits. *Proceedings of the IEEE*, 89(5):665–692, May 2001.

[13] M. Függer, U. Schmid, G. Fuchs, and G. Kempf. Fault-Tolerant Distributed Clock Generation in VLSI Systems-on-Chip. Research Report 12/2006, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2006.

[14] S. Hauck. Asynchronous design methodologies: An overview. *Proceedings of the IEEE*, 83(1):69–93, Jan. 1995.

[15] K. Hoyme and K. Driscoll. Safebus. In *Proceedings IEEE/AIAA 11th Digital Avionics Systems Conference*, pages 68–73, 1992.

[16] S. G. Intel, R. Ronen, I. Anati, A. Berkovits, T. Kurts, A. Naveh, A. Saeed, Z. Sperber, and R. C. Valentine. The Intel Pentium M processor: Microarchitecture and performance. *Intel Technology Journal*, 7(2), May 2003.

[17] R. M. Kieckhafer, C. J. Walter, A. M. Finn, and P. M. Thambidurai. The MAFT architecture for distributed fault tolerance. *IEEE Transactions on Computers*, 37:398–405, Apr. 1988.

[18] H. Kopetz and G. Grünsteidl. TTP-A protocol for fault-tolerant real-time systems. *Computer*, 27(1):14–23, 1994.

[19] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[20] G. Le Lann and U. Schmid. How to implement a timer-free perfect failure detector in partially synchronous systems. Technical Report 183/1-127, Department of Automation, Technische Universität Wien, January 2003.

[21] J. B. Lerch. Threshold Gate Circuit Employing Field-Effect Transistors. *U.S. Patent 3 715 603*, Feb. 1973.

[22] M. S. Maza and M. L. Aranda. Analysis of clock distribution networks in the presence of crosstalk and groundbounce. In *Proceedings International IEEE Conference on Electronics, Circuits, and Systems (ICECS)*, pages 773–776, 2001.

[23] M. S. Maza and M. L. Aranda. Interconnected rings and oscillators as gigahertz clock distribution nets. In *GLSVLSI '03: Proceedings of the 13th ACM Great Lakes symposium on VLSI*, pages 41–44. ACM Press, 2003.

[24] R. E. Miller. *Sequential Circuits and Machines*, volume 2 of *Switching Theory.* wiley, 1965.

[25] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim. Robust system design with built-in soft-error resilience. *IEEE Computer*, 38(5):43–52, Feb. 2005.

[26] C. J. Myers. *Asynchronous Circuit Design.* Wiley, 2001.

[27] A. K. Palit, V. Meyer, W. Anheier, and J. Schloeffel. Modeling and analysis of crosstalk coupling effect on the victim interconnect using the ABCD network

model. In *Proceedings of the 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'04)*, pages 174–182, Oct. 2004.

[28] D. Powell, J. Arlat, L. Beus-Dukic, A. Bondavalli, P. Coppola, A. Fantechi, E. Jenn, C. Rabejac, and A. Wellings. GUARDS: A generic upgradable architecture for real-time dependable systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):580–599, June 1999.

[29] U. Schmid. How to model link failures: A perception-based fault model. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'01)*, pages 57–66, Göteborg, Sweden, July 1–4, 2001.

[30] U. Schmid and A. Steininger. Dezentrale Fehlertolerante Taktgenerierung in VLSI Chips. Research Report 69/2004, Technische Universität Wien, Institut für Technische Informatik, 2004. (Österr. Patentanmeldung A 1223/2004).

[31] T. K. Srikanth and S. Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, July 1987.

[32] I. E. Sutherland. Micropipelines. *Commun. ACM*, 32(6):720–738, June 1989.

[33] J. Widder. *Distributed Computing in the Presence of Bounded Asynchrony.* PhD thesis, Vienna University of Technology, Fakultät für Informatik, May 2004.

[34] J. Widder, G. Le Lann, and U. Schmid. Failure detection with booting in partially synchronous systems. In *Proceedings of the 5th European Dependable Computing Conference (EDCC-5)*, volume 3463 of *LNCS*, pages 20–37, Budapest, Hungary, Apr. 2005. Springer Verlag.

[35] J. Widder and U. Schmid. Achieving synchrony without clocks. Research Report 49/2005, Technische Universität Wien, Institut für Technische Informatik, 2005. (submitted).