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

Matthias Függer, Ulrich Schmid, Gottfried Fuchs
Vienna University of Technology
Embedded Computing Systems Group (E182/2)
Email: {fuegger, s, fuchs}@ecs.tuwien.ac.at

Gerald Kempf
Austrian Aerospace GmbH
Email: gerald.kempf@space.at

Abstract

This paper[1] introduces a simple fault-tolerant tick generation algorithm based on Srikanth & Toueg's consistent broadcast primitive that can be directly implemented in VLSI using asynchronous digital logic. The need for adaption originates from two peculiarities of hardware implementations: (i) Fine-grained parallel asynchronous computations, which undermines the concept of atomic steps common to all distributed computing models, and (ii) very limited resources, which makes even apparently simple operations prohibitively costly. We prove that the resulting algorithm is correct, and give analytic expressions for performance metrics like worst case precision and accuracy. Moreover, we outline the major building blocks of our synthesizable VHDL implementation and provide some measurement results from our FPGA prototype. Our results hence provide the required basis for investigating robust alternatives to synchronous clocking in VLSI Systems-on-Chip and similar applications.

1 Motivation

Shrinking feature sizes and increasing clock speeds are the most visible signs of the tremendous advances in VLSI design, which will accommodate billions of transistors on a single chip in the near future [12]. This comes at the price of increased system-level complexity, however: With today's deep submicron technology with GHz clock speeds, wiring delays dominate transistor switching delays, and signals cannot traverse the whole die within a single clock cycle any more. Moreover, the reduced voltage swing needed for high clock speeds and low power consumption dramatically increases the adverse effects of single event upsets like α-particle or neutron hits. The resulting increase of the transient failure rate (soft-error rate) [17] and crosstalk sensitivity [23] has raised concerns about the dependability of future generation VLSI chips [5]. In fact, a modern VLSI chip can no longer be viewed as a monolithic block of synchronous hardware, where all state transitions occur simultaneously. Rather, VLSI chips are nowadays considered as systems of interacting subsystems — the advent of Systems-on-Chip (SoC). Due to the problems listed above, however, SoCs have much in common with the loosely-coupled distributed systems that have been studied by the fault-tolerant distributed algorithms community for decades. This paper explores whether it is possible to utilize some of this research for SoCs and similar VLSI devices.

More specifically, in the context of our DARTS-Project (*ti.tuwien.ac.at/darts*), which is a joint project between Vienna University of Technology and Austrian Aerospace, we will explore an alternative approach (patented in [26]) to synchronous clocking in VLSI chips and PCB-level system designs. As shown in Fig. 1, the idea is to replace the external quartz oscillator and the clock tree, which supplies the clock signal to the different functional units (Fu_i) on a traditional chip, using a GALS-like approach [4]: Every functional unit has attached a dedicated fault-tolerant tick generation block (TS-Alg), which generates the Fu's local clock signal. In contrast to GALS, however, our approach ensures that the local clock signals of different Fu's are closely synchronized to each other. To accomplish this, all TS-Alg blocks communicate with each other over a simple "network" of clock signals (TS-Net). This alternative clocking approach has a number of advantages, which makes it particularly promising for certain application domains: First of all, it does not need a quartz oscillator, which is an expensive and sensitive device (shock, vibration, temperature etc.). The generated clock always runs at the maxi-

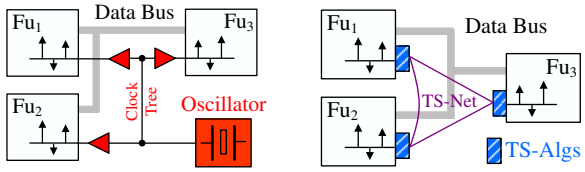

Figure 1. Replacing synchronous clocking by fault-tolerant distributed tick generation.

[1]Supported by the Austrian bm:vit FIT-IT project *DARTS* (proj. no. 809456-SCK/SAI) and the Austrian FWF project *Theta* (proj. no. P17757).

mum speed and adapts to the current operating conditions[2]. Moreover, the approach tolerates transient failures in TS-Algs and TS-Net and avoids the cumbersome clock tree engineering issue [2,9]. And last but not least, as different Fus are driven by slightly different clock signals, our approach alleviates EM radiation and ground bouncing problems [19] that typically plague devices using synchronous clocking.

Contributions: This paper shows that it is indeed possible to adapt fault-tolerant distributed algorithms to the particular needs of VLSI implementations. More specifically:

(i) We adapt the simple variant of Srikanth & Toueg's [27] consistent broadcasting introduced in [29] to the peculiarities of VLSI hardware implementations, namely, inherent fine-grained parallelism and very limited resources. Our major modifications are the enforcement of some atomic actions (interlocking) via implicit handshaking, and the replacement of k-bit messages by anonymous rising or falling signal transitions (zero-bit messages).

(ii) We provide a fault-tolerant distributed tick generation algorithm (TS-Alg), which tolerates up to f Byzantine faulty instances in a system containing $n \geq 3f + 2$ TS-Algs. Examples of Byzantine failures are spurious clock transitions or early timing failures that are perceived inconsistently at different TS-Algs.

(iii) We prove that the resulting algorithm is correct, and derive bounds for its performance metrics like worst case precision and minimal/maximal clock frequency. Since our "system-level proof" rests upon some simple properties of certain digital logic blocks only, which can be easily verified by means of standard design tools, we can guarantee the correctness of any system of $n \geq 3f + 2$ correctly implemented TS-Algs.

(iv) We provide some details of our synthesizable VHDL implementation of the algorithm, and demonstrate the feasibility of our approach by means of some measurement results obtained from an FPGA prototype system. These results will hence allow us to implement our DARTS clock generation scheme in a prototype SoC ASIC.

Related work: Asynchronous distributed systems theory has been applied to VLSI chips for decades [7]: Research on transition signaling [3], delay-insensitivity [18], micropipelines [28], etc. has in fact established a sound basis for dealing with self-timed systems [10]. However, those approaches cannot deal with failures.

Given the importance of dependability issues, there is a huge body of research work devoted to fault-tolerance in VLSI. However, the proposed techniques are very different from the "system-level approach" employed in fault-tolerant distributed algorithms: Fine-grained fault tolerance, e.g. at gate level, and error detection and recovery are the methods of choice in VLSI chips [22]. This research is hence not relevant w.r.t. our approach.

There is also a sizeable body of work devoted to hardware implementations of fault-tolerant algorithms. Well-known examples are MAFT [13], SAFEBUS [11] and TTP [14]. However, in sharp contrast to our problem, these systems incorporate hardware *assistance* only. The major part of the algorithms is still implemented in conventional software and executed on general-purpose processors. Consequently, there was no need to minimize the gate-level resource consumption implied by these algorithms. Somewhat an exception is the paper [1], which shows that consensus can be solved with 1-bit messages. None of the above systems had to deal with fine-grain parallelism inherent in VLSI implementations, though.

There is also some related work on alternative clocking schemes in VLSI. Since we do not consider external clock sources in our approach, we can ignore the sizeable body of work on hardware-assisted fault-tolerant clock synchronization (see [25] for an overview) here. The few approaches for distributed clock generation without external clock sources we are aware of are essentially based on a (distributed) ring oscillator, which is formed by gates arranged in a positive feedback loop. Instead of being dictated by a quartz, the frequency of the generated clock signal is determined by the end-to-end delay of the feedback loop. In [20], a regular structure of closed loops of an odd number of inverters is used for distributed clock generation. Similarly, [8] employs local tick generation cells, arranged in a two-dimensional grid. Since clock synchronization theory [6] reveals that high connectivity is required for bounded synchronization tightness in presence of failures, however, the sparsely connected designs proposed in [8,20] are not fault-tolerant.

Organization of the paper: In Section 2, we informally explain the original tick generation algorithm and the required modifications. Section 3 is devoted to the detailed specification of our hardware tick generation algorithm and its building blocks. In Section 4, the algorithm's correctness proof and the performance analysis are provided. An overview of our VHDL implementation and some experimental results are provided in Section 5. Some conclusions in Section 6 complete the paper.

2 Informal Overview

The TS-Alg developed and analyzed in this paper derives from a simple synchronizer algorithm introduced in [29]. The (core of this) algorithm, which is based on Srikanth & Toueg's well-known consistent broadcasting primitive [16,

[2]Several important questions must still be answered in our DARTS-Project, however: E.g., it is not clear yet how area and power consumption of some reasonable number of TS-Algs relate to area and power consumption of a clock tree. Those problems, which primarily requires comparison of suitable ASIC implementations, are outside the scope of this paper.

```
0:  VAR k : integer := 0;
1:
2:  initially send tick(0) to all [once];
3:
4:  if received tick(ℓ) from at least f + 1 distinct processes with ℓ ≥ k
5:     send tick(k), ..., tick(ℓ) to all [once]; k := ℓ;
6:  if received tick(k) from at least 2f + 1 distinct processes
7:     send tick(k + 1) to all [once]; k := k + 1;
```

Figure 2. Algorithm for generating approximately simultaneous messages [29].

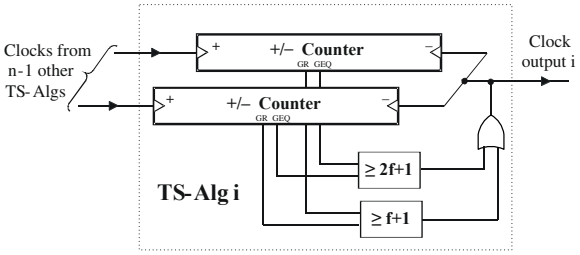

Figure 3. Basic architecture of a TS-Alg.

24, 27], is shown in Fig. 2. When executed in a system of $n = 3f + 1$ processes (= TS-Alg instances) with at most f of them being Byzantine faulty, it generates a sequence of consecutive messages *tick(k)*, $k \geq 0$, at every process. The algorithm ensures that the difference of the points in real-time when two correct processes p and q emit *tick(k)* is bounded by a certain constant precision π, which can be computed from the max. (τ^+) and min. (τ^-) of the end-to-end transmission and computation delay of *tick(.)* messages.

The algorithm is started by sending *tick(0)* in `line 2` and works as follows: If a correct process p receives $f + 1$ *tick(ℓ)* messages (`line 4`), it can be sure that at least one of those was sent by a correct process. Therefore, p can safely catch up and send *tick(k)*, ..., *tick(ℓ)*. If some process p receives $2f + 1$ *tick(k)* messages (`line 6`), one can be sure that at least $f + 1$ of those will be received by every other correct process (which then executes `line 4`) within $\varepsilon = \tau^+ - \tau^-$. Hence, all other correct processes $q \neq p$ receive $2f + 1$ *tick(k)* messages within another τ^+. It follows that if p emits *tick(k)* at time t, every other correct q emits *tick(k)* no later than $t + \varepsilon + \tau^+$. This eventually guarantees a bound π on the synchronization precision, as claimed above. Note that the algorithm automatically adapts to the instantaneous timing characteristics of all involved computations and message transmissions.

Since the algorithm in Fig. 2 looks very simple, it is tempting to conclude that it should be easy to translate into a hardware description language: The k-th generated clock tick occurs when a process sends its *tick(k)* message. It turns out, however, that a number of challenging issues must be solved to accomplish this:

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

How to adapt the original algorithm for zero-bit messages? By just sending anonymous signal transitions, no information except the occurrence time can be conveyed over the TS-Net. Thus, the tick number k contained in a message in the algorithm of Fig. 2 must be maintained at every receiver, individually for every sender.

How to ensure atomicity of actions in a VLSI implementation? Any distributed computing model we are aware of assumes atomic computing steps at the level of a single processor. This abstraction does not apply when an algorithm is implemented directly in hardware, however, since all "computations" are done by several digital logic gates that run concurrently. Explicit synchronization (serialization of actions/interlocking) must be introduced if two local computations must not interfere with each other.

Taking into account the above issues, we arrived at the following basic architecture of a single TS-Alg shown in Fig. 3. The major building blocks of a single TS-Alg are the $n-1$ up/down counters, one for every of the $n-1$ other TS-Algs in the system. Each such device counts the difference of (i) the number of clock ticks seen from the respective peer, and (ii) the number of clock ticks generated locally so far. It is supposed to provide two binary status signals, *GR* and *GEQ*, which are true when the counter's actual value is > 0 and ≥ 0, respectively. In addition, we need $\geq f + 1$ and $\geq 2f + 1$ threshold circuits implementing the rules in (`line 4`) and (`line 6`) in Fig. 2, respectively. Finally, there is a device (shown as an OR-gate in Fig. 3), which is responsible for generating the local clock ticks from the outputs of the threshold gates.

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 [10] must be devised. The major[4] problem here is to distinguish *GR*, *GEQ* signals that contributed to the previously generated tick k from *GR*, *GEQ* signals contributing to the next (to be generated) tick $k + 1$. Our solution exploits the fact that the transitions of a binary-valued signal must strictly alternate between low-to-high and high-to-low: We just provide independent signals *GR*, *GEQ* and 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] Actually, n broadcast links that provide every TS-Alg with the messages from all other TS-Algs are sufficient.

[4] Lacking space does not allow us to discuss all the resulting problems. See Appendix A.1 for an overview.

More specifically, we generate an odd tick $k+1 \in \mathbb{N}_{odd}$ if the last tick generated was even ($k \in \mathbb{N}_{even}$) and either (a) the same or a greater number of ticks have been seen from $\geq 2f+1$ TS-Algs (*GEQ*^{e} true), or (b) a greater number of ticks have been seen from $\geq f+1$ TS-Algs (*GR*^{e} true). Note that (a) ensures that the even tick k has been seen from $\geq 2f+1$ TS-Algs, whereas (b) guarantees that the odd tick $k+1$ has been seen from $\geq f+1$ ones. Analogous rules involving *GEQ*^{o} and *GR*^{o} are applied for generating the even tick $k+1 \in \mathbb{N}_{even}$.

The output of the threshold circuits (say TH^o_{GR} and TH^o_{GEQ}) that generated the even tick k are ignored when the next tick $k+1$ to be generated is odd. This "gap" thus allows *GR*^{o} and *GEQ*^{o} (which were responsible for activating TH^o_{GR} or TH^o_{GEQ}) for tick k to first become inactive and then become active again for the tick $k+2$. As revealed by Lemma 4.5, this is sufficient to avoid mixing up old and new instances of *GR*^{o} and *GEQ*^{o}.

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 (= 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. In this section, we will provide the architectural design model of a TS-Alg that meets those requirements.

3.1 Signals and Zero-bit Message Channels

All components of our TS-Algs, which are digital logic blocks, deal with binary signals only. Given such a signal S [or an arbitrary boolean predicate], with possible values $\bot$ (= logical 0, false, inactive) and $\top$ (= logical 1, true, active), we say that the event (= state transition) $S{-}\uparrow\ (t^*)$ resp. $S{-}\downarrow\ (t^*)$ occurs when S changes state from $\bot$ to $\top$ resp. from $\top$ to $\bot$ at time t^*. The status $S(t)$ of S at time $t \geq t^*$ is $S(t) = \bot$ resp. $S(t) = \top$ iff the last event at or before t was $S{-}\downarrow\ (t^*)$ resp. $S{-}\uparrow\ (t^*)$.

In our analysis, we will reason about events and status of binary signals [and predicates]. In order not to clutter our notation, we will employ the convention that, depending on the context of usage, $S(t)$ will denote either: (i) S's status $S(t) \in \{\bot, \top\}$, where t denotes the observation time, or (ii) the event $S{-}\uparrow\ (t)$, where t denotes the time of the last transition to the active state $\top$ (or reset time).

Note also that we will sometimes drop the time t from events and status if it is clear from the context.

All components in our system are interconnected by signal wires, which are modeled as reliable FIFO channels

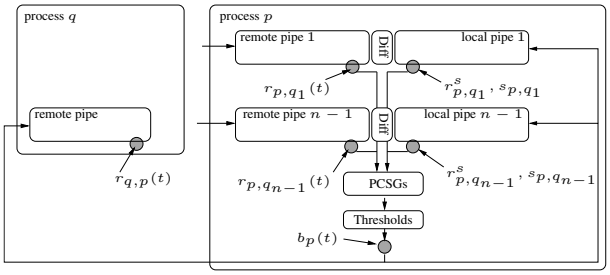

Figure 4. TS-Alg architecture, including the points of observation $b_p(t)$, $r_{p,q}(t)$ and $r^s_{p,q}(t)$

with finite delay 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 (= 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 (= message) $X^r{-}\uparrow\ (t')$ resp. $X^r{-}\downarrow\ (t')$ for every sent event $X^s{-}\uparrow\ (t)$ resp. $X^s{-}\downarrow\ (t)$ within finite time $t' \geq t$. 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 state $X^r(t_0)$ is initialized to $\bot$.

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 relation [15]: For matching send and receive events, it holds that $X^s{-}\uparrow\ (t) \rightarrow X^r{-}\uparrow\ (t')$ and $X^s{-}\downarrow\ (t) \rightarrow X^r{-}\downarrow\ (t')$. To simplify the notation when using a channel X, we will 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.

3.2 TS-Alg Component and Architecture Specification

In this section we will specify the TS-Alg components' behavior and determine their arrangement. The architecture is depicted in Fig. 4. Note that a $+/-$ counter in Fig. 3 comprises the first 3 components listed below.

Pairs of elastic pipes: Every TS-Alg/process p incorporates $n-1$ pairs of elastic pipelines (= a shift register/FIFO for signal transitions [28]), each of which corresponds to a single remote TS-Alg/process $q \in P \setminus \{p\}$. 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 4.13.

For description and analysis purposes, we will need some notation: $r_{p,q}(t)$ resp. $r^s_{p,q}(t)$ denotes the number of

messages that arrived at the end of the remote resp. local pipe at time t. Moreover, $s_{p,q}(t)$ denotes the number of tick messages stored in the local pipeline by t. Note that those quantities are not available to the algorithm. Rather, the algorithm uses only the binary status signals $r_{p,q}(t) \geq r^s_{p,q}(t)$ and $r_{p,q}(t) > r^s_{p,q}(t)$ in conjunction with $s_{p,q}(t) = 1$. Upon reset, all pipelines are initialized to contain a single even tick$-\downarrow$ message. We assume that $r_{p,q}(t_{0,p}) = r^s_{p,q}(t_{0,p}) = 0$ and $s_{p,q}(t_{0,p}) = 1$ at reset time $t_{0,p}$.

Diff-Gate: To avoid the need for infinite storage, 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 $r_{p,q}(t) \geq r^s_{p,q}(t) \wedge s_{p,q}(t) > 1$, there is some $t' \in t + [\tau^-_{\mathit{Diff}}, \tau^+_{\mathit{Diff}}]$ such that $s_{p,q}(t') = s_{p,q}(t'-dt)-1$, for some infinitesimally small $dt > 0$. Note that we are not loosing information due to this removal of "common" tick messages, since the algorithm is only interested in the difference of the number of messages received.

Pipe Compare Signal Generators (PCSGs): There exists a dedicated detection circuit for each pair of pipes, which generates the status signals $GEQ^{o/e}_{p,q}(t)$ and $GR^{o/e}_{p,q}(t)$. In particular, $GEQ^o_{p,q}(t')$ becomes active (i.e., $GEQ^o_{p,q}(t') = \top$, thereby generating the event $GEQ^o_{p,q}-\uparrow$ at t' if the previous state was $\bot$) at some time $t' \in t + [\tau^-_{\mathit{GEQ}}; \tau^+_{\mathit{GEQ}}]$ when (i) $r^s_{p,q}(t) \in \mathbb{N}_{\mathit{odd}}$ and (ii) $[r_{p,q}(t) \geq r^s_{p,q}(t)] \wedge [s_{p,q}(t) = 1]$. Similarly, $GR^o_{p,q}(t')$ becomes active at time $t' \in t + [\tau^-_{\mathit{GR}}; \tau^+_{\mathit{GR}}]$ when (i) $r^s_{p,q}(t) \in \mathbb{N}_{\mathit{odd}}$ and (ii) $[r_{p,q}(t) > r^s_{p,q}(t)] \wedge [s_{p,q}(t) = 1]$. The signals $GEQ^e_{p,q}(t)$ and $GR^e_{p,q}(t)$ have the same definition, except that (i) reads $r^s_{p,q}(t) \in \mathbb{N}_{\mathit{even}}$ here. Note that a PCSG may become active only if $s_{p,q}(t) = 1$.

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, the corresponding threshold signal $TH^{o/e}_{GEQ}(t)$ resp. $TH^{o/e}_{GR}(t)$ becomes active within $[\tau^-_{\mathit{TH}}; \tau^+_{\mathit{TH}}]$.

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} resp. TH^e_{GR} for the current tick becomes active. We denote by $b_p(t)$ the number of tick messages generated by process p by time t, with $b_p(t_0) = 0$ at reset time t_0. If t_k denotes the time when p generates tick k, we assume 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 our 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 upon reset). In case of the threshold gates (`line 18–line 29`), we conceptually assume that a (possibly idempotent) output event is sent upon any change of the state of any input.

```
0:  /* Initialization */
1:  ∀q : r_{p,q}(t_{0,p}) = r^s_{p,q}(t_{0,p}) = 0; s_{p,q}(t_{0,p}) = 1
2:  ∀channels X : X^r(t_{0,p}) = ⊥
3:
4:  /* code for PCSGs at process p for remote process q*/
5:  if [r_{p,q}(t) ≥ r^s_{p,q}(t)] ∧ [r^s_{p,q}(t) ∈ N_odd] ∧ [s_{p,q}(t) = 1]
6:      → send GEQ^o_{p,q_i}(t)−↑
7:  else → send GEQ^o_{p,q_i}(t)−↓
8:  if [r_{p,q}(t) > r^s_{p,q}(t)] ∧ [r^s_{p,q}(t) ∈ N_odd] ∧ [s_{p,q}(t) = 1]
9:      → send GR^o_{p,q_i}(t)−↑
10: else → send GR^o_{p,q_i}(t)−↓
11: if [r_{p,q}(t) ≥ r^s_{p,q}(t)] ∧ [r^s_{p,q}(t) ∈ N_even] ∧ [s_{p,q}(t) = 1]
12:     → send GEQ^e_{p,q_i}(t)−↑
13: else → send GEQ^e_{p,q_i}(t)−↓
14: if [r_{p,q}(t) > r^s_{p,q}(t)] ∧ [r^s_{p,q}(t) ∈ N_even] ∧ [s_{p,q}(t) = 1]
15:     → send GR^e_{p,q_i}(t)−↑
16: else → send GR^e_{p,q_i}(t)−↓
17: /* code for thresholds at process p */
18: if GEQ^o_{p,q_i}(t) for at least 2f + 1 remote processes q_i
19:     → send TH^o_{GEQ}−↑
20: else → send TH^o_{GEQ}−↓
21: if GR^o_{p,q_i}(t) for at least f + 1 remote processes q_i
22:     → send TH^o_{GR}−↑
23: else → send TH^o_{GR}−↓
24: if GEQ^e_{p,q_i}(t) for at least 2f + 1 remote processes q_i
25:     → send TH^e_{GEQ}−↑
26: else → send TH^e_{GEQ}−↓
27: if GR^e_{p,q_i}(t) for at least f + 1 remote processes q_i
28:     → send TH^e_{GR}−↑
29: else → send TH^e_{GR}−↓
30: /* code for sending tick messages at process p */
31: if [TH^o_{GR}(t) ∨ TH^o_{GEQ}(t)] ∧ ¬[TH^e_{GR}(t) ∨ TH^e_{GEQ}(t)]
32:     → send tick−↓
33: if [TH^e_{GR}(t) ∨ TH^e_{GEQ}(t)] ∧ ¬[TH^o_{GR}(t) ∨ TH^o_{GEQ}(t)]
34:     → send tick−↑
```

Figure 5. TS-Alg tick generation algorithm adopted for VLSI implementations.

4 Correctness Proofs

For our correctness proof and performance analysis, we employ the following system and failure model: 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 $[\tau^-_{\mathit{loc}}; \tau^+_{\mathit{loc}}]$, whereas the time to reach the end of any remote pipeline, at any remote process, is bounded by $[\tau^-_{\mathit{rem}}; \tau^+_{\mathit{rem}}]$. Note that this assumption has some impact on initialization as well: Every process p can be initialized to $b_p(t_{0,p}) = r_{p,q}(t_{0,p}) = r^s_{p,q}(t_{0,p}) = 0$ and $s_{p,q}(t_{0,p}) = 1$ even at (slightly) different reset times $t_{0,p} \in [0; \tau^-_{\mathit{rem}})$: The lower delay bound τ^-_{rem} ensures that no tick message can be lost

here. The latencies $[\tau^-_{GEQ};\tau^+_{GEQ}]$, $[\tau^-_{GR};\tau^+_{GR}]$, $[\tau^-_{Diff};\tau^+_{Diff}]$ and $[\tau^-_{TH};\tau^+_{TH}]$ have already been introduced earlier.

Our tick generation algorithm will allow up to f processes to fail arbitrarily, provided that the total number of processes is $n \geq 3f+2$. This is slightly more than the required lower bound of $n \geq 3f+1$ for clock synchronization [6], but facilitates a much better precision and accuracy (attained by counting only remote messages when calculating the $f+1$ resp. $2f+1$ thresholds; including self reception would lead to $\tau^-_{rem} = \tau^-_{loc}$ in Theorem 4.11 (Precision)). In our context, the adverse power of arbitrary failures lies in the ability of a process to generate wrong (early timing failures 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.

We will start our formal treatment with Definition 4.1, which will be employed frequently in our 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. We say that they are* directly causally related*, denoted by $I(t') \rightarrow O(t)$, if they are (i) causally related and (ii) there is neither an $\uparrow$- nor a $\downarrow$-event $I'(t'')$ on the same input in between, i.e., $\nexists I'(t'') : I(t') \rightarrow I'(t'') \rightarrow O(t)$. 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 in Fig.5 is to make sure that a process generates tick $k+1$ messages based on tick k messages only, i.e., does not incorporate stale information from earlier ticks $\ell < k$ here. This will be formalized in the following Definition 4.2. Lemma 4.3 below will deduce some simple properties from this definition.

Definition 4.2. (Notion of Basis). *Abbreviate*

$$\begin{aligned} \mathcal{P}^{GEQ,\ell}_{p,q}(t) = &\quad [r_{p,q}(t) \geq r^s_{p,q}(t) = \ell] \wedge [s_{p,q}(t) = 1], \\ \mathcal{P}^{GR,\ell}_{p,q}(t) = &\quad [r_{p,q}(t) > r^s_{p,q}(t) = \ell] \wedge [s_{p,q}(t) = 1]. \end{aligned} \quad (1)$$

We say that 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 $k+1 \in \mathbb{N}_{even}$,*

$$\begin{aligned} \mathcal{P}^{GEQ,\ell}_{p,q}(t'') \rightarrow GEQ^o_{p,q}(t') \rightarrow \\ TH^o_{GEQ}(t'_{k+1}) \rightarrow b_p(t_{k+1}) = k+1 \end{aligned} \quad (2)$$

$$\begin{aligned} \mathcal{P}^{GR,\ell}_{p,q}(t'') \rightarrow GR^o_{p,q}(t') \rightarrow \\ TH^o_{GR}(t'_{k+1}) \rightarrow b_p(t_{k+1}) = k+1 \end{aligned} \quad (3)$$

(and analogous for $k+1 \in \mathbb{N}_{odd}$). 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 $\exists q_i : \ell_{q_i} = \ell$.*

Lemma 4.3. *The time instants t'_{k+1} and t'' in the direct causal chains (2) and (3) in Definition 4.2 satisfy $\tau^-_{TH} + \min(\tau^-_{GR}, \tau^-_{GEQ}) \leq t'_{k+1} - t'' \leq \tau^+_{TH} + \max(\tau^+_{GR}, \tau^+_{GEQ})$. Moreover, the predicate $\mathcal{P}^{GEQ,\ell}_{p,q}(t)$ resp. $\mathcal{P}^{GR,\ell}_{p,q}(t)$ holds not only at $t = t''$, but continues to hold 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 our lemma is a trivial consequence of Definition 4.1. To prove that the predicates continue to hold true during the given intervals, 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 $[TH^o_{GR}(t_{k+1}) \vee TH^o_{GEQ}(t_{k+1})] \wedge \neg[TH^e_{GR}(t_{k+1}) \vee TH^e_{GEQ}(t_{k+1})] \wedge b_p(t_{k+1} - dt) = k \in \mathbb{N}_{odd}$, 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 $k+1$.

Stipulating that the event $b_p(t_{k+1}) = k+1$ occurs at t_{k+1} implies that all enabling conditions are met. In particular, TH^o_{GEQ} resp. TH^o_{GR} must not have further changed state within the time interval $(t'_{k+1}, t_{k+1}]$. We will show now that the predicates (1) must also maintain the same state for some time. Suppose, by way of contradiction, that e.g. $\mathcal{P}^{GEQ,\ell}_{p,q}(t)$ reverted to false at some time $t'' < t \leq t_{k+1} - \tau^+_{TH} - \tau^+_{GEQ}$. Then, the GEQ^o threshold gate must have generated another event $TH^o_{GEQ}(t''_{k+1})$ (maybe an idempotent one) at time $t''_{k+1} = t + \tau^+_{GEQ} + \tau^+_{TH} \leq t_{k+1}$ at latest, and clearly $TH^o_{GEQ}(t''_{k+1}) \rightarrow b_p(t_{k+1}) = k+1$. This contradicts direct causality of $TH^o_{GEQ}(t'_{k+1}) \rightarrow b_p(t_{k+1}) = k+1$, however. □

On top of this the major Lemma 4.5 can be shown which states that every correct process p's tick $k+1$ is based on tick k only, provided that the following Constraint 4.4 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 4.4. (Interlocking Constraint). *With the abbreviations*

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

$D^+ \leq D^- + D^-_{dis}$ with $D^-_{dis} = D^- - \tau^-_{Diff}$ must hold.

Lemma 4.5. (Interlocking). *Given Constraint 4.4, every correct process p's tick $k+1$ is based on tick k only.*

The proof is by induction on the number of ticks p has sent so far. It is first shown that all ticks up to number 2 are based on their predecessor. Then we assume that some tick k is the first tick that is not based on the precedent tick $k-1$ but rather on some tick $\in \{k-2, k-4, \dots\}$. By investigating the direct causal chains that would lead to this behavior, we obtain a contradiction to Constraint 4.4.

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 up to tick k are based on tick $k-1$, but that tick $k+1$ generated by process p is not based on tick k but rather on tick $\ell < k$. Assuming w.l.o.g. $k \in \mathbb{N}_{odd}$ and hence $\ell \in \{k-2, k-4, \dots\}$, there must be some correct process q_i satisfying (2) and/or (3), say, $\mathcal{P}^{GEQ,\ell}_{p,q_i}(t''_{q_i}) \rightarrow GEQ^o_{p,q_i}(t'_{q_i}) \rightarrow TH^o_{GEQ}(t'_{k+1}) \rightarrow b_p(t_{k+1}) = k+1$ according to Definition 4.2. Lemma 4.3 reveals that $\mathcal{P}^{GEQ,\ell}_{p,q_i}(t') = [r_{p,q_i}(t') \geq r^s_{p,q_i}(t') = \ell] \wedge [s_{p,q_i}(t') = 1]$ must evaluate to true also at time t' with $t_{k+1} - t' \leq \tau^+_{TH} + \tau^+_{GEQ} \leq \tau^+_{TH} + \max(\tau^+_{GR}, \tau^+_{GEQ})$. Hence, tick $k-1$ must have been received at process p's local pipe corresponding to q_i at time $t'' > t'$, since otherwise the first predicate in $\mathcal{P}^{GEQ,\ell}_{p,q_i}(t')$ would be false at t'. Since tick $k-1$ must have been sent by process p at time $t_{k-1} \geq t'' - \tau^+_{loc}$ in this case, we finally arrive at $t_{k-1} > t_{k+1} - \tau^+_{TH} - \max(\tau^+_{GR}, \tau^+_{GEQ}) - \tau^+_{loc} = t_{k+1} - D^+$.

Since tick $k-1$ arrives at any local pipe at $\tilde{t} \geq t_{k-1} + \tau^-_{loc}$ earliest, $r^s_{p,q}(\tilde{t}) = k-1$ is obtained not before $\tilde{t} + \tau^-_{Diff}$, as the previous tick $k-2$ must be removed from the local pipe before $s_{p,q}(t') = 1$ can become true. Due to the induction hypothesis, tick k is based on tick $k-1$ only. Lemma 4.3 hence implies that tick k can not be sent before $t_k \geq \tilde{t} + \tau^-_{Diff} + \min(\tau^-_{GR}, \tau^-_{GEQ}) + \tau^-_{TH} \geq t_{k-1} + D^-$.

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 again. Process p generated tick k when sufficiently many correct processes q_j satisfied (2) and/or (3), i.e., contributed 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

$$\begin{aligned}\neg\{[r_{p,q}(\hat{t}_q) \geq r^s_{p,q}(\hat{t}_q)] \wedge [r^s_{p,q}(\hat{t}_q) = k-1] \\ \wedge [s_{p,q}(\hat{t}_q) = 1]\} \rightarrow \\ \neg GEQ^e_{p,q}(t') \rightarrow \neg TH^e_{GEQ}(t_{k+1}) \text{ or} \\ \neg\{[r_{p,q}(\hat{t}_q) > r^s_{p,q}(\hat{t}_q)] \wedge [r^s_{p,q}(\hat{t}_q) = k-1] \\ \wedge [s_{p,q}(\hat{t}_q) = 1]\} \rightarrow \\ \neg GR^e_{p,q}(t') \rightarrow \neg TH^e_{GR}(t_{k+1})\end{aligned}$$

holds at some time $\hat{t}_q$, with $t_k < \hat{t}_q$ and $\hat{t}_q + \min(\tau^-_{GR}, \tau^-_{GEQ}) + \tau^-_{TH} \leq t_{k+1}$. The first predicate can only become true if tick k has arrived at the local queue corresponding to q_j. Thus, $t_k + \tau^-_{loc} \leq \hat{t}_q$ and hence $t_{k+1} \geq t_k + \min(\tau^-_{GR}, \tau^-_{GEQ}) + \tau^-_{TH} + \tau^-_{loc} = t_k + D^-_{dis}$.

Combining the latter results on t_{k+1} and t_k, we obtain $t_{k+1} \geq t_{k-1} + D^- + D^-_{dis}$. Substituting the result for t_{k-1}, we find $t_{k+1} > t_{k+1} - D^+ + D^- + D^-_{dis}$ and hence $D^+ > D^- + D^-_{dis}$, which contradicts Constraint 4.4. Therefore, tick $k+1$ cannot be based on $\ell < k$ and we are done. □

We will now provide a sequence of simple lemmas, which are needed for establishing our major results Theorem 4.11 (Precision), Theorem 4.12 (Accuracy) and Theorem 4.13 (Pipeline size).

Lemma 4.6. (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 4.5, tick $k+1$ can only be based on tick number k. This means that $\mathcal{P}^{GEQ,k}_{p,q_i}(t'_{q_i})$ 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 τ^-_{loc} accounts for the minimum delay to reach the end of a local pipe and τ^-_{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. □

Theorem 4.7 and its proof are a generalization of the well-known consistent broadcasting results of [24, 27, 29].

Theorem 4.7. (Synchronization Properties). *The algorithm satisfies the synchronization properties Progress (P), Unforgeability (U) and Simultaneity (S) if Constraint 4.4 and* $n \geq 3f+2$ *hold.*

(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 $T^-_{first} = \tau^-_{rem} + \tau^-_{GEQ} + \tau^-_{TH} + \tau^-_{Diff}$ or earlier.*

(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 $T_{sim} = -\tau^-_{TH} - \tau^-_{GEQ} - \tau^-_{Diff} + \varepsilon + \tau^+_{Diff} + \tau^+_{GR} + \tau^+_{TH} + T^+$, where $\varepsilon = \tau^+_{rem} - \tau^-_{rem}$.*

Proof. By generalization of [24, 27, 29]. See Appendix A.2.1 for the detailed proofs. □

Lemma 4.8. (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 4.9 relates the progress of the ticks generated by the fastest process and real-time.

Lemma 4.9. (Maximum Increase of b^{max}). *Let $b^{max}(t)$ be the maximum of $b_p(t)$ over all correct processes p, and t_k^{first} be the time when b^{max} was increased to k, i.e. when the first process sent tick k. Define the indicator function $I_{usync}(t) = I_{t\notin\{t_0^{first},t_1^{first},t_2^{first},\dots\}}$ to be 1 if the bracketed condition is fulfilled, and 0 otherwise. Then, for any $t_1 \le t_2$, it holds that $b^{max}(t_2) - b^{max}(t_1) \le \left\lfloor \frac{t_2-t_1}{T^-_{first}} \right\rfloor + I_{usync}(t_1)$.*

Proof. For $t_1 \in \{t_0^{first}, t_1^{first}, t_2^{first}, \dots\}$, Lemma 4.8 (Fastest Progress) can be applied, which reveals that $b^{max}(t_2) - b^{max}(t_1) \le \left\lfloor \frac{t_2-t_1}{T^-_{first}} \right\rfloor$ 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 $b^{max}(t_2)-b^{max}(t_1) \le \left\lfloor \frac{t_2-t_k}{T^-_{first}} \right\rfloor + 1 \le \left\lfloor \frac{t_2-t_1}{T^-_{first}} \right\rfloor + 1$ by monotonicity of $\lfloor x \rfloor$. □

Lemma 4.10. (Slowest Progress). *Assume that some process p sends tick k at time t. Then all correct processes must send tick $k' > k$ by time $t + (k'-k)T^+ + T_{sim}$.*

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

Theorem 4.11 gives the precision π of our algorithm, which guarantees $\forall t : |b_q(t) - b_p(t)| \le \pi$ for every pair of correct processes p and q.

Theorem 4.11. (Precision). *The precision $\pi \ge |b_q(t) - b_p(t)|$ of our algorithm is bounded by $\pi \le \left\lfloor \frac{T_{sim}}{T^-_{first}} \right\rfloor + 1$.*

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

$$\begin{aligned} b^{max}(t') &\le b_p(t') + \pi \text{ and} \\ b^{max}(t_{k+1}^p) &\le b_p(t_{k+1}^p) + \pi - 1 \end{aligned} \tag{5}$$

Assume that process p sends tick $k+1$ at time t_{k+1}^p. Because of (S), $t_{k+1}^p \le t_{k+1}^{first} + T_{sim}$ must hold, and because of monotonicity, $b^{max}(t_{k+1}^p - T_{sim}) \le b^{max}(t_{k+1}^{first}) = k+1$ must hold. According to Lemma 4.9, $b^{max}(t_{k+1}^p) \le \left\lfloor \frac{T_{sim}}{T^-_{first}} \right\rfloor + I_{usync}(t_{k+1}^p - T_{sim}) + b^{max}(t_{k+1}^p - T_{sim})$. 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 $b^{max}(t_{k+1}^p) \le \left\lfloor \frac{T_{sim}}{T^-_{first}} \right\rfloor + k + 1$. Now, for $t_k^p \le t' < t_{k+1}^p$, we obtain $b_p(t') = k$ and $b^{max}(t') \le b^{max}(t_{k+1}^p)$ and hence $b^{max}(t') - b_p(t') \le \left\lfloor \frac{T_{sim}}{T^-_{first}} \right\rfloor + 1 = \pi$. For t_{k+1}^p, obviously $b_p(t_{k+1}^p) = k+1$ and hence $b^{max}(t_{k+1}^p) - b_p(t_{k+1}^p) \le \left\lfloor \frac{T_{sim}}{T^-_{first}} \right\rfloor = \pi - 1$.

The inequalities (5) can be generalized to hold at any time t, by finding an appropriate k for which $t_k^p \le t < t_{k+1}^p$, or $t = t_{k+1}^p$ holds at p and applying (5) within these bounds. Since obviously $\forall t : b_p(t) \le b^{max}(t)$, any two correct processes p and q must fulfill $\forall t : |b_q(t) - b_p(t)| \le \pi$. □

Theorem 4.12 (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 is within $[1/T^+, 1/T^-_{first}]$.

Theorem 4.12. (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 $\max\left\{0, \frac{\Delta - T_{sim} - T^+}{T^+}\right\} \le |b_p(t_2) - b_p(t_1)| \le \left\lceil \frac{\Delta}{T^-_{first}} \right\rceil + \min\left\{\pi + 1, \left\lceil \frac{\Delta}{D^-} - \frac{\Delta}{T^-_{first}} \right\rceil\right\}$.*

Proof. We will first show the upper bound for the accuracy. We know that $\forall t : b_p(t) \ge b^{max}(t) - \pi + (1 - I_{usync}(t))$ and $\forall t : b_p(t) \le b^{max}(t)$ by Lemma 4.11 and Lemma 4.9. Thus $b_p(t_2) - b_p(t_1) \le b^{max}(t_2) - b^{max}(t_1) + \pi - (1 - I_{usync}(t_1))$. By applying Lemma 4.9, $b_p(t_2) - b_p(t_1) \le \left\lfloor \frac{t_2-t_1}{T^-_{first}} \right\rfloor + 2I_{usync}(t_1) - 1 + \pi \le \left\lfloor \frac{t_2-t_1}{T^-_{first}} \right\rfloor + \pi + 1 \le \left\lceil \frac{t_2-t_1}{T^-_{first}} \right\rceil + \pi + 1$. Moreover, from Lemma 4.6 it follows that $b_p(t_2) - b_p(t_1) \le \lceil \frac{t_2-t_1}{D^-} \rceil$. Hence, $b_p(t_2) - b_p(t_1) \le \min\left\{ \left\lceil \frac{\Delta}{T^-_{first}} \right\rceil + \pi + 1, \lceil \frac{\Delta}{D^-} \rceil \right\} \le \left\lceil \frac{\Delta}{T^-_{first}} \right\rceil + \min\left\{\pi + 1, \lceil \frac{\Delta}{D^-} \rceil - \left\lceil \frac{\Delta}{T^-_{first}} \right\rceil\right\} \le \left\lceil \frac{\Delta}{T^-_{first}} \right\rceil + \min\left\{\pi + 1, \left\lceil \frac{\Delta}{D^-} - \frac{\Delta}{T^-_{first}} \right\rceil\right\}$ since $\lceil x + y \rceil \le \lceil x \rceil + \lceil y \rceil$.

To prove the lower bound, we first define $b_1 = b_p(t_1)$, $b_2 = b_p(t_2)$ and $t_{b_1}^p \le t_2$, $t_{b_2}^p \le t_2$ as the points in time when p sends tick b_1 and b_2. Clearly $t_{b_2+1}^p > t_2$, and by Lemma 4.10 it follows $t_2 - t_1 \le t_{b_2+1}^p - t_{b_1}^p \le T_{sim} + (b_2 - b_1 + 1)T^+$. Hence, $\max\left\{0, \frac{\Delta - T_{sim} - T^+}{T^+}\right\} \le b_p(t_2) - b_p(t_1)$. □

This final theorem establishes that pipeline size is indeed bounded, provided that the additional condition $\tau^+_{Diff} \leq T^-_{first}$ holds.

Theorem 4.13. (Pipeline size). *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* $S = \left\lfloor \frac{T_{sim}+\max(\tau^+_{rem},\tau^+_{loc})+\tau^+_{Diff}}{T^-_{first}} \right\rfloor + 1$.

The proof is by bounding the maximum time a tick messages can exist before it is eliminated by the Diff-Gate.

The following lemma establishes 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 4.14. *If $\tau^+_{Diff} \leq T_{first}-$, 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 $T_k = t_k^{last} + \max(\tau^+_{rem},\tau^+_{loc}) = t_k^{first} + T_{sim} + \max(\tau^+_{rem},\tau^+_{loc})$. We show by induction that tick $k-1$ is removed at most τ^+_{Diff} later.

We can start the induction with $k=0$, where 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. Now assume that tick $k-1$ has been removed by time $T_k + \tau^+_{Diff}$ 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 4.9, we find $T_{k+1} \geq T_k + T^-_{first} \geq T_k + \tau^+_{Diff}$. 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 $T_{k+1} + \tau^+_{Diff}$ as asserted. □

Theorem 4.15. (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* $S = \left\lfloor \frac{T_{sim}+\max(\tau^+_{rem},\tau^+_{loc})+\tau^+_{Diff}}{T^-_{first}} \right\rfloor + 1$.

Proof. From Lemma 4.14, we know that tick $k-1$ is completely removed by time $T'_k = t_k^{first} + T_{sim} + \max(\tau^+_{rem},\tau^+_{loc}) + \tau^+_{Diff}$. Hence, the maximum size of any pipe is determined by the maximum number of ticks any process can generate in the interval $T_{sim}+\max(\tau^+_{rem},\tau^+_{loc})+\tau^+_{Diff}$. According to Lemma 4.9, those are at most $S = \left\lfloor \frac{T_{sim}+\max(\tau^+_{rem},\tau^+_{loc})+\tau^+_{Diff}}{T^-_{first}} \right\rfloor + 1$. □

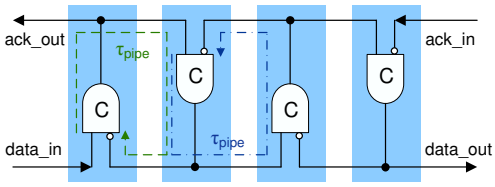

Figure 7. Elastic pipeline consisting of Muller C-Elements and inverters.

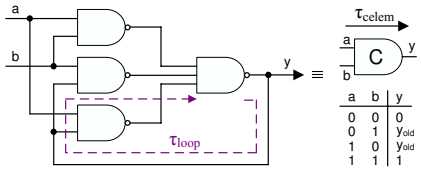

Figure 8. Muller C-Element at gate-level.

5 Hardware Implementation

Now we will present implementation details of the low-level building blocks of our TS-Alg implementation, which have been introduced at higher levels of abstraction in Fig. 3 and 4. We will also derive timing conditions an implementation has to satisfy in order to fulfill the behavioral specification of Section 3.2. It turns out that the conditions can be enforced easily by design tools, if not a priori true.

Fig. 6 shows an overview of the complete architecture of a single TS-Alg that refines Fig. 4.

5.1 Implementation of $+/-$ Counters

Every TS-Alg consists of $n-1$ $+/-$ counters, one for every remote TS-Alg. Every $+/-$ counter consists of (i) two elastic pipelines, which are used as FIFO-buffers for remote resp. local ticks, (ii) a custom Diff-Gate responsible for removing matching ticks from the end of both pipelines, and (iii) a pipe compare signal generation (PCSG) unit that actually generates the pivotal signals GEQ^e, GR^e, GEQ^o and GR^o. We will describe those elements in some detail below.

Elastic Pipeline: An elastic pipeline can be seen as a FIFO buffer for signal transitions, based on Sutherland's micropipelines [28]. As shown in Fig. 7, it consists of a chain of Muller C-Elements [21] (+ inverters), each of which can store a single low-to-high/high-to-low transition.

In order to get an idea of the internal operation of the elastic pipeline in Fig. 7, consider the gate-level implementation of a C-Element given in Fig. 8. It is apparent that it changes the value of its output y only if both inputs have the same value – it hence implements an AND for signal transitions. Note that a C-Element must incorporate a feedback loop, since y must stay at the current logic level when either a or b (but not both) changes.

The correct operation of the pipeline depends upon some timing constraints regarding (i) the feedback loop involved

in the Muller C-Elements and (ii) the loop formed by two consecutive stages in the pipeline. Actually, the delays of those feedback loops impose some minimum delay between successive input transitions, which must be maintained in order not to jeopardize correct behavior. In standard delay-insensitive applications of elastic pipelines, this is ensured by triggering a new `data_in` transition only when the `ack_out` transition that acknowledges the previous `data_in` transition occurs. Since we obviously cannot use `ack_out` in our application (we cannot acknowledge every generated clock tick), the required timing constraints must be externally maintained.

We will first analyze the timing conditions of the Muller C-Element: To allow the gates on the feedback path in Fig. 8 to settle, it has to be ensured that `a` and/or `b` do not change state in opposite directions (high-to-low, low-to-high) within τ^+_{loop} of each other if potentially causing a state change. The condition $\tau_{min} \geq \tau^+_{loop}$ has to be obeyed for both consecutive transitions on a single input as well as opposite transitions on different inputs if these transitions would lead to a state change (e.g. `a` high, whereas `b` performs low-to-high and high-to-low within τ^+_{loop}). The bound τ^+_{loop} can be obtained by analyzing the netlist of the hardware generated by a synthesis resp. place&route tool.

The basic timing constraints for the elastic pipeline, which originate from the feedback loops formed by any two consecutive pipeline stages, as highlighted in Fig. 7, can be derived in a similar manner: As long as input signal `data_in` does not change faster than any stage's τ^+_{pipe}, i.e., $\tau_{min} \geq \tau^+_{pipe}$, no transition can be lost and the elastic pipeline behaves as expected. Since activation of a C-Element's output takes place after the propagation delay τ^+_{celem}, cp. Fig. 8, we have $\tau^+_{pipe} = 2\tau^+_{celem} + \tau^+_{inverter} + \tau^+_{wires}$. If τ^+_{celem} is considered to be the same for all C-Elements for simplicity, it follows that the minimum timing constraint $\tau_{min} \geq \tau^+_{loop}$ for any single C-Element implicitly holds when $\tau_{min} \geq \tau^+_{pipe}$ and $\tau^+_{pipe} \geq \tau^+_{loop}$ (place&route constraint).

It is important to mention, however, that the simple considerations above apply only to the case where `data_in` and `ack_in` do not change simultaneously. The latter case, which could obviously cause metastability problems if there were no additional constraints, is treated in conjunction with the Diff-Gate below.

Difference Gate: The Diff-Gate is responsible for removing matching transitions from the ends of the remote and local pipe, see Fig. 9. Our goal is to implement a $+/-$ counter, which keeps track of the difference of the number of ticks seen so far. Still, every pipe must have at least $S = \left\lfloor \frac{T_{sim} + \max(\tau^+_{rem}, \tau^+_{loc}) + \tau^+_{Diff}}{T^-_{first}} \right\rfloor + 1$ stages to avoid an overflow due to the fact that remote and local clock are not perfectly synchronized, cp. Theorem 4.13. Note that S is the only implementation-technology-dependent parameter that is compiled into the TS-Algs. However, since it depends on the ratio of maximum and minimum delays only, like in the Theta-Model [16], rather than on the absolute delays, we are convinced that it is fairly independent of things like temperature and implementation technology.

Our Diff-Gate implementation, cp. Fig. 9, is by a simple asynchronous state machine which, in case of matching values of `req_ext` and `req_int`, removes the last transition of the remote (left) pipeline before it removes the matching last transition of the local (right) one. There are two reasons why this left-before-right removal is mandatory: The first one is related to guaranteeing the required properties of the signals GEQ^e, GR^e, GEQ^o and GR^o and is explained in the context of the PCSG below. The second reason is the avoidance of metastability problems at C_1 in Fig. 9, which connects two only loosely coupled clock domains. To cause metastability problems, the inputs of the C-Element C_1 have to change in a manner (introduced earlier) where its internal τ^+_{loop} constraint is violated. We have shown before, however, that this cannot happen in the standard case of a single input change if the pipeline constraint regarding $\tau_{min} \geq \tau^+_{pipe}$ is maintained. Hence, we are left with the case where both inputs of C_1 change in opposite directions within τ^+_{loop} of each other. However, our Diff-Gate removes matching transitions only, i.e., acts only when `req_ext` and `req_int` have the same value. Hence, after a matching 0, for example, any of the inputs, say, `req_ext` of C_1 can go to 1 only when `ack_ext` = 0 has been generated by C_1 and processed by C_2. Hence, `req_ext` can go to 1 not earlier than $2\tau^-_{celem} + \tau^-_{inverter} + \tau^-_{wires} > \tau^+_{loop}$ (place&route constraint on τ^-_{celem} of C_1, C_2) after the last of `req_ext` and `req_int` went to 0, and `req_int` can not go to 1 before another $2\tau^-_{celem} + 2\tau^-_{inverter} + \tau^-_{wires}$ (τ^-_{celem} of C_3, C_4). Consequently, if the above constraint is maintained and if local and remote clock ticks are not generated faster than they can be removed by the Diff-Gate, i.e., faster than $\tau^-_{Diff} = 4\tau^-_{celem} + 3\tau^-_{inverter} + \tau^-_{wires} \geq \tau^+_{pipe}$ (place&route constraint), we can ensure its correct operation. Since τ^-_{Diff} is part of D^-, cp. (4), which in turn determines the minimum time τ_{min} between successive clock ticks according to Lemma 4.6, we can be sure that $\tau_{min} > \tau^-_{Diff} > \tau^+_{pipe} > \tau^+_{loop}$. Hence, our $+/-$ counters indeed work as expected.

Pipe Compare Signal Generators: The PCSG has to detect whether or not the remote pipeline holds the same or

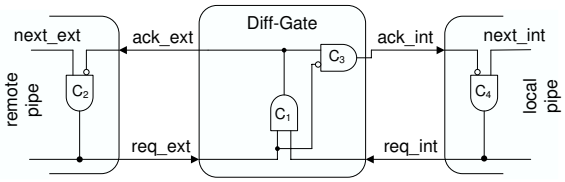

Figure 9. Diff-Gate in combination with remote and local pipeline.

a greater number of transitions and must activate the corresponding status signals GEQ^e, GR^e, GEQ^o and GR^o. Unlike all other building blocks described so far, the PCSG is not a transition signaling logic but rather a conventional asynchronous state logic. However, it must be able to handle transitions, which arrive at random instants, without producing glitches or wrongly activate its output signals. Since this is impossible in general, we have separated GEQ^e, GR^e from GEQ^o and GR^o: As already explained in Section 2, the former signals are only relevant if the next tick to be generated is odd, whereas the latter status signals are only relevant when it is even. When the next tick to be generated is the other one, we do allow glitches and other incorrect signal changes.

To ensure correct behavior of our algorithm, the relevant status signals for the next tick to be generated, say, GEQ^e, must be activated only if $r_{p,q}(t) > r^s_{p,q}(t) \wedge s_{p,q}(t) = 1$ holds for sure. This requires a left-before-right removal of matching transitions by the Diff-Gate: GEQ^e must never become true, not even in a glitch, if the number of remote ticks seen is smaller than the number of local ticks. If we allowed the Diff-Gate to remove the right transition before the left one, however, the PCSG unit might see a larger number of remaining transitions in the remote pipe than in the local pipe, and would hence produce an erroneous output on GEQ^e. Further, it must be ensured that any state signal can become true only if the next tick to be generated is not already available in the local pipe. This can happen for the pipelines corresponding to a slow remote node, where the faster ones have already achieved the required $2f + 1$ or $f + 1$ threshold. Our PCSG implementation ensures this by comparing the last stages of the local and remote pipeline, subject to the condition that the three last stages of the local pipeline contain the same value. It is easy to see from the operation of an elastic pipeline that the latter condition signals an (almost) empty pipe, which is sufficient to infer the required "freshness" of the status outputs.

5.2 Other low-level building blocks

Apart from the $n - 1$ $+/-$ counters, a TS-Alg also consists of standard asynchronous $f + 1$ and $2f + 1$ threshold gates and a clock generation unit (represented by the OR-gate in Fig. 3).

Threshold gates: The threshold logic is responsible for implementing the if-clauses depicted in Fig. 5. The clock generation is based on four separate threshold gates: Two are active, waiting for $\geq f + 1$ GEQ^e resp. $\geq 2f + 1$ GR^e, when the next tick to be generated is odd, while the remaining two, waiting for $\geq f + 1$ GR^o resp. $\geq 2f + 1$ GEQ^o, are inactive, in the sense that they can behave arbitrarily.

Clock generation: According to the algorithm in Fig. 5, either the active $f + 1$ or the active $2f + 1$ threshold gate may produce the next clock tick. Our implementation employs a suitable combination of active-high and active-low signals on the inputs and outputs of the threshold gates, which allows us to use a simple logical AND resp. OR for combining the two corresponding threshold gate outputs for generating even resp. odd ticks. The output of the odd and even tick generation is joined by a final Muller C-Element, which ensures that the local clock output remains stable until the next valid clock tick is generated. Note that this element can be seen as the back-transition from the state-based logic (PCSG and threshold gates) to transition signaling logic ($+/-$ counters).

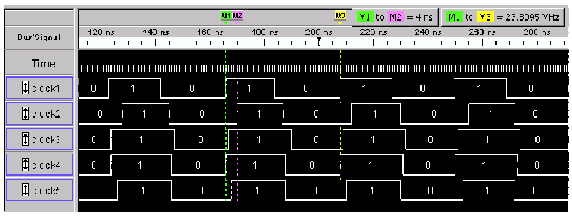

Figure 10. DARTS approach with 5 TS-Alg running on an FPGA

5.3 Experimental Evaluation

The low-level building blocks described above have been implemented using VHDL and assembled in a complete TS-Alg. We have also prototyped a complete system of $n = 5$ TS-Algs, which has been synthesized and put into operation on an Altera APEX EP20K1000 FPGA. The clock signals of all five TS-Algs in a sample run are depicted in Fig. 10. The clock frequency is approximately 24MHz, and the maximum skew of the clock ticks is 4ns. Of course, this FPGA implementation is just a proof of concept and shall demonstrate the principal feasibility of the DARTS approach. We are convinced that there is a huge potential for optimizations, e.g., to increase the clock frequency: FPGAs are not particularly suitable for asynchronous designs due to the structure of the lookup tables and registers. Furthermore, the performance of the TS-Alg clock depends upon the ratio of the routing delays of the longest and the shortest clock signal paths ($\frac{\tau+}{\tau-}$). Due to the static interconnect structure of an FPGA, these signal delays may vary within a wide range. We are hence convinced that much better performance, at much lower cost, can be obtained by the standard cell CMOS ASIC implementation ultimately targeted by our DARTS project.

6 Conclusions

We proposed a novel clock generation approach for VLSI chips that has been derived from a well-known distributed fault-tolerant tick generation algorithm. It provides

a set of local clock signals, to be fed to the subsystems of a SoC, for example, which are synchronized within a bounded precision to each other. Our approach does not need any external quartz oscillator or the like, and generates a clock frequency that automatically adapts to the current operating conditions.

Major modifications had to be applied to the original algorithm in order to adapt to the inherent fine-grain parallelism and limited resources of VLSI hardware implementations. Our algorithm depends upon the implementation technology only via the required number S of stages of the elastic pipelines. Since S depends upon the ratio of max. and min. clock signal delays only, rather than on the delay values itself, we are convinced that it is actually reasonably independent from implementation technology.

We also provided a correctness proof which shows that a system incorporating $n \geq 3f+2$ clock generation units can cope with up to f Byzantine faulty nodes and links. This allows our algorithm to work correctly in presence of up to f units that produce spurious clock transients perceived inconsistently by the other units. Our proof rests upon simple properties of some locally available signals only, which can be verified by digital design tools. Hence, the correctness of a system of any size n can be guaranteed if the low-level building blocks providing the required signals are implemented correctly. Note that a comparable result cannot be established via model checking. Current work is devoted to speed optimization and stabilization, both within the scope of the algorithm and the hardware implementation. First simulations of our ASIC implementation reached $\geq$ 200 MHz clock frequency. Systematic experimental evaluations are also planned in the near future.

Acknowledgements The contributions of Johann Vilanek (Diff-Gate, preliminary simulations and experiments), Markus Ferringer (FPGA implementation) and Thomas Handl (tools and library setup) are gratefully acknowledged. Valuable feedback on the design and implementation of our TS-Algs was provided by Josef Widder and Andreas Steininger, who also helped us to improve the presentation in this paper.

References

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

[2] R. Bhamidipati, A. Zaidi, S. Makineni, K. Low, R. Chen, K.-Y. Liu, and J. Dalgrehn. Challenges and methodologies for implementing high-performance network processors. *Intel Technology Journal*, 6(3):83–92, Aug. 2002.

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

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

[5] 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.

[6] 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.

[7] E. M. C. (ed.). Editorial: Distributed computing issue in hardware design. *Distributed Computing*, 1:185–186, 1986.

[8] 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.

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

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

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

[12] International technology roadmap for semiconductors, 2001.

[13] 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.

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

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

[16] 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.

[17] A. Maheshwari, I. Koren, and W. Burleson. Accurate estimation of Soft Error Rate (SER) in VLSI circuits. In *Proceedings of the 2004 IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 377–385, Oct. 2004.

[18] A. J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1:226–234, 1986.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] U. Schmid, J. Klasek, T. Mandl, H. Nachtnebel, G. R. Cadek, and N. Kerö. A Network Time Interface M-Module for distributing GPS-time over LANs. *J. Real-Time Systems*, 18(1):24–57, Jan. 2000.

[26] 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).

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

[28] I. E. Sutherland. Micropipelines. *Communications of the ACM, Turing Award*, 32(6):720–738, June 1989. ISSN:0001-0782.

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

A APPENDIX

A.1 Informal Overview of TS-Alg Design Problems

- *How to reconcile transition signaling logic and fault-tolerance?*

  The probably most elegant paradigm for asynchronous logic is transition signaling, where information is conveyed exclusively via signal transitions, rather than via signal states as in conventional logic [10]. Any reasonable delay-insensitive asynchronous circuit can be composed from a small set of elementary elements, which e.g. includes the Muller C-Element (see Section 5) forming the equivalent of the logical AND of two input signals, see [3, 18, 28] for details.

  The "expressive" power of transition signaling is solely restricted to time-free systems, however, where causality is the only meaningful relation between events. Consequently, there is no full equivalent to the logical OR of two input signals [10]: If only one of the two inputs can provide a transition, the EXOR (exclusive OR) gate can safely be used for this purpose However, there is no meaningful transition signaling OR if both inputs could—but need not—provide a (somehow related) transition. In fact, generating an OR output transition when the first input transition arrives would destroy the causality relation of the second input transition and the generated output transition.

  Unfortunately, incorporating fault-tolerance, as instantiated by our threshold gates, requires exactly this semantics: The clock signal of some process must not depend causally upon the clocks of faulty processes, since this may lead to blocking. Hence, there is no way but to switch from transition signaling to state signaling and vice versa here.

- *How to manage a clean switch between transition signaling and state signaling?*

  In our implementation of Fig. 3, some transition signaling logic is used for processing clock ticks ($+/-$ counters) and generating exactly one state transition *tick(*k*)* for any k (OR-gate). The threshold circuits for processing the status signals *GR* and *GEQ*, on the other hand, are ordinary asynchronous state logic devices, with their well-known tendency to produce glitches.

  The problem of avoiding such glitches, which are catastrophic for transition signaling, is solved by serializing the generation of ticks with odd and even number. More specifically, since transitions of a binary-valued signal must strictly alternate between low-to-high (= odd clock tick) and high-to-low (= even clock

tick), it is obvious that the next tick to be generated after an even tick must be odd, and vice versa.

Hence, in our solution, we just duplicate the signals *GR*, *GEQ* and the threshold circuits and use the following rules:

- Generate an odd tick $k{+}1 \in \mathbb{N}_{odd} := 2\mathbb{N}{+}1$ if the last tick generated was even ($k \in \mathbb{N}_{even} := 2\mathbb{N}$) and either (a) the same or a greater number of ticks have been seen from $\geq 2f+1$ TS-Algs (*GEQ*^{e} true), or (b) a greater number of ticks have been seen from $\geq f+1$ TS-Algs (*GR*^{e} true). Note that condition (a) ensures that the even tick k has been seen from $\geq 2f+1$ ones, whereas (b) guarantees that the odd tick $k{+}1$ has already been seen from $\geq f+1$ ones.
- Generate an even tick $k+1 \in \mathbb{N}_{even}$ if the last tick generated was odd ($k \in \mathbb{N}_{odd}$) and either (a) the same or a greater number of ticks have been seen from $\geq 2f+1$ TS-Algs (*GEQ*^{o} true), or (b) a greater number of ticks have been seen from $\geq f+1$ TS-Algs (*GR*^{o} true). Again, condition (a) ensures that the odd tick k has been seen from $\geq 2f+1$ ones, whereas (b) guarantees that the even tick $k{+}1$ has already been seen from $\geq f{+}1$ ones.

Glitches due to the non-simultaneous arrival of clock signal transitions from the peers are masked by simply ignoring the output of the threshold circuits (say TH^o_{GR} and TH^o_{GEQ}) that generated the even tick k when the next tick $k+1$ to be generated is odd[5]. This "gap" thus allows *GR*^{o} and *GEQ*^{o} (which were responsible for activating TH^o_{GR} or TH^o_{GEQ}) for tick k to first become inactive and then become active again for the tick $k+2$. As revealed by Lemma 4.5, this is sufficient to avoid mixing up old and new instances of *GR*^{o} and *GEQ*^{o}.

- *How to implement the $+/-$ counters?*

  This is the most delicate part of the hardware design work. First of all, implementing an asynchronous up/down counter is inherently difficult due to the fact that the up-clock and the down-clock transitions can occur arbitrarily close to each other. Nevertheless, we will provide an implementation that circumvents the resulting metastability problems [28] by design.

  Another problem is how to correctly generate the status signals *GR*^{o} and *GEQ*^{o} (resp. *GR*^{e} and *GEQ*^{e}). They should truly reflect the current counter value, at least during times when they are used. We will specify the detailed properties that must be maintained by those signals in Section 3. The correctness proof and performance analysis will only rely upon those properties, i.e., hold for every implementation of the local $+/-$ counters that fulfills the required properties.

  In Section 5, we will introduce a suitable implementation of the $+/-$ counters: Every counter consists of two elastic pipelines [28], which can be seen as shift registers for signal transitions. One is attached to the remote clock signal, the other one is fed by the local clock signal. They are fitted together at their ends via a special Diff-Gate that removes "matching" transitions as soon as they traveled through the pipelines. The status signals *GR*^{o}, *GEQ*^{o}, *GR*^{e} and *GEQ*^{e} are derived from monitoring the last few stages of both pipes.

All details are provided in Sections 3 and 5.

[5] Of course, glitches are forbidden as soon as this next tick has occurred.

A.2 Proofs

A.2.1 Synchronization Properties Proof (Theorem 4.7)

Lemma A.1. *The first correct process that sends tick $k+1 > 0$ 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, for 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 4.5, there exists a time $t'_{q_i} < t_{k+1}$ where $\mathcal{P}^{GR,k}_{p,q_i}(t'_{q_i})$ 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. □

Proof. We will show the properties Progress (P), Unforgeability (U) and Simultaneity (S) 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 $r_{p,q_i}(t') \geq b_{q_i}(t) \geq k$ and $r^s_{p,q_i}(t') \geq b_p(t)$. Now assume that $r^s_{p,q_i}(t') > k$. 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 $r^s_{p,q_i}(t') = k$. Then, $r_{p,q_i}(t') \geq r^s_{p,q_i}(t') = k$ must hold for at least $2f+1$ correct processes q_i. Therefore, $\mathcal{P}^{GEQ,k}_{p,q_i}(t'')$ must hold at time $t'' \leq t' + \tau^+_{Diff}$ if still $r^s_{p,q_i}(t'') = k$ (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 $t''' \leq t' + \tau^+_{Diff} + \tau^+_{GEQ} + \tau^+_{TH} \leq t + T^+$. 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 $TH^o_{GEQ}(t_{k+1})$ or $TH^o_{GR}(t_{k+1})$ is active.

1. Assume that tick $k+1$ was sent by process p because $TH^o_{GEQ}(t_{k+1})$ was active. Thus $TH^o_{GEQ} - \uparrow$ must have been sent at time $t' \leq t_{k+1} - \tau^-_{TH}$. Furthermore, Lemma 4.5 (Interlocking) states that $r_{p,q_i}(t''_{q_i}) \geq r^s_{p,q_i}(t''_{q_i}) = k \wedge s_{p,q_i}(t''_{q_i}) = 1$ 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 $t''' \leq t_{k+1} - \tau^-_{TH} - \tau^-_{GEQ} - \tau^-_{Diff} - \tau^-_{rem}$.
2. Assume tick $k+1$ was sent by process p because $TH^o_{GR}(t_{k+1})$ was active. By similar arguments as above, $\mathcal{P}^{GR,k}_{p,q_i}(t''_{q_i})$ 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 $t''' \leq t_{k+1} - \tau^-_{TH} - \tau^-_{GR} - \tau^-_{Diff} - \tau^-_{rem}$. 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 $t + \tau^-_{rem} + \tau^-_{Diff} + \tau^-_{GEQ} + \tau^-_{TH}$. 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 A.1 it must do so via TH^o_{GEQ}. Because of Lemma 4.5 (Interlocking), $\mathcal{P}^{GEQ,k-1}_{p,q_i}(t'_{q_i})$ 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 $t^{(1)}_{\bar{q}_i} \leq t_k - \tau^-_{TH} - \tau^-_{GEQ} - \tau^-_{Diff} - \tau^-_{rem}$.

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 $t^{(2)}_{q_j} \leq t_k - \tau^-_{TH} - \tau^-_{GEQ} - \tau^-_{Diff} + \varepsilon$. Thus, at all correct processes q_j, $r_{q_j,\bar{q}_i}(t^{(2)}_{q_j}) \geq k-1$ must be true for at least $f+1$ processes and all q_j will send tick $k-1$ by time $t^{(3)} \leq t_k - \tau^-_{TH} - \tau^-_{GEQ} - \tau^-_{Diff} + \varepsilon + \tau^+_{Diff} + \tau^+_{GR} + \tau^+_{TH}$ 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 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 $t^{(3),-} \leq t^- - \tau^-_{TH} - \tau^-_{GEQ} - \tau^-_{Diff} + \varepsilon + \tau^+_{Diff} + \tau^+_{GR} + \tau^+_{TH}$.

Finally, Progress (P) can be applied once, stating that all correct processes must send tick k by time $t^{(4)} \leq t_k - \tau^-_{TH} - \tau^-_{GEQ} - \tau^-_{Diff} + \varepsilon + \tau^+_{Diff} + \tau^+_{GR} + \tau^+_{TH} + T^+$. □

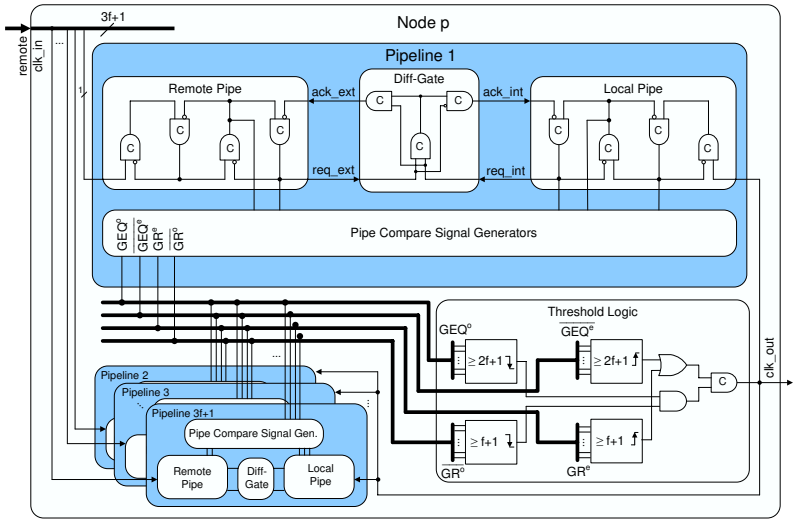

Figure 6. TS-Alg hardware implementation for a single node p