

How to Speed-up Fault-Tolerant Clock Generation in VLSI Systems-on-Chip via Pipelining

Matthias Függer
TU Wien, ECS Group
Vienna (Austria)
fuegger@ecs.tuwien.ac.at

Andreas Dielacher
RUAG Space
Vienna (Austria)
andreas.dielacher@space.at

Ulrich Schmid
TU Wien, ECS Group
Vienna (Austria)
s@ecs.tuwien.ac.at

Abstract—Fault-tolerant clocking schemes become inevitable when it comes to highly-reliable chip designs. Because of the additional hardware overhead, existing solutions are considerably slower than their non-reliable counterparts. In this paper¹, we demonstrate that pipelining is a viable approach to speed up the distributed fault-tolerant DARTS clock generation approach introduced in (Függer, Schmid, Fuchs, Kempf, EDCC’06), where a distributed Byzantine fault-tolerant tick generation algorithm has been used to replace the traditional quartz oscillator and highly balanced clock tree in VLSI Systems-on-Chip (SoCs). We provide a pipelined version of the original DARTS algorithm, termed pDARTS, together with a novel modeling and analysis framework for hardware-implemented asynchronous fault-tolerant distributed algorithms, which is employed for rigorously analyzing its correctness & performance. Our results, which have also been confirmed by the experimental evaluation of an FPGA prototype implementation, reveal that pipelining indeed allows to entirely remove the adverse effect of large interconnect delays on the achievable clock frequency, and demonstrate again that methods and results from distributed algorithms research can successfully be applied in the VLSI context.

Keywords-Fault-tolerant distributed algorithms, VLSI, clock synchronization, pipelining, modeling approaches.

I. MOTIVATION

Modern *very-large scale integration* (VLSI) circuits, in particular, *systems-on-chip* (SoCs), have much in common with the loosely-coupled distributed systems that have been studied by the fault-tolerant distributed algorithms community for decades (see [2], [3] for an overview on commonalities). It is hence tempting to try and employ distributed algorithms results and methods in this new application domain. Recent work e.g. on scheduling of DRAM memory requests [4] and hardware-implemented transactional memory in multicores [5], fault-tolerant clock generation in SoCs [6], and self-stabilizing microprocessors [7] confirm that this is indeed feasible and quite promising. Conversely, results and methods from VLSI design have also been applied successfully in the distributed algorithms context. Examples are error-correcting codes, which allow to efficiently cope with Byzantine adversaries [8] and bear interesting relations to fault-tolerant consensus [9], and

pipelining, the most important paradigm for concurrency in VLSI design, which is also a well-known technique for speeding up synchronous distributed algorithms [10, Ch. 6.2.2], [11]–[13].

This paper extends and integrates techniques from both distributed algorithms and VLSI design for developing and proving correct a pipelined version of the DARTS fault-tolerant clock generation approach for SoCs introduced in [6]: Fault-tolerant clocking schemes become inevitable when it comes to highly-reliable SoC designs. In contrast to the classical non fault-tolerant approach, which uses a quartz oscillator and a clock tree for disseminating the clock signal throughout the chip, DARTS employs a Byzantine fault-tolerant distributed tick generation algorithm. The latter is a variant of Srikanth & Toueg’s consistent broadcasting primitive [14] introduced in [15], which has been adapted to the particular needs of a VLSI implementation [6], [16]. Clearly fault-tolerance does not come for free. Since the frequency of an ensemble of DARTS clocks is solely determined by the end-to-end delays along certain paths (which depend on the chip’s physical dimensions and hence cannot be made arbitrarily small), the maximum clock frequency is limited. For example, our first FPGA prototype implementation ran at about 24 MHz; our recent space-hardened 180 nm CMOS DARTS ASIC runs at about 55 MHz.

Fortunately, pipelining comes as a rescue for speeding-up the clock frequency here; first estimates predict a clock frequency of 100 MHz for our current 180 nm CMOS implementation.

The purpose of this paper is actually two-fold: First, the pDARTS algorithm demonstrates that pipelining is not only effective for speeding-up synchronous distributed algorithms, but also for fault-tolerant asynchronous ones in systems with large bandwidth \times delay products. Note that the latter shows a clear rising trend in modern distributed systems, since the bandwidth provided by state-of-the-art computer networks and processors—as well as by VLSI data paths and circuits—has tremendously increased, while the spatial distances between nodes—including inter-chip and on-chip communication—remain essentially the same. In the asynchronous context, pipelining exploits the fact that any FIFO data transmission/processing path in a dis-

¹A Brief Announcement [1] of this work was presented at PODC’09. This work is supported by the Austrian Science Foundation (FWF) projects P21694 (FATAL), P17757 (Theta) and P20529 (PSRTS).

tributed system, ranging from asynchronously (clocklessly) “computing” logic gates interconnected by simple wires to multiprocessors nodes (CPU, network processors, DMA controllers, etc.) interconnected via FIFO network links, has an inherently pipelined architecture. For example, a simple message channel [even a wire] with bandwidth 10^9 messages/s and delay 5 ns is able to “store” 5 messages. A fault-tolerant distributed algorithm may hence immediately start phase k (rather than wait for the acknowledgments of the previous data processing phase $k - 1$ in a “stop-and-go fashion”), provided that the acknowledgments for phase $k - X - 1$ (for some integer $X > 0$) have already been received from sufficiently many correct processes.

And second, the new modeling framework used for the correctness proof and performance analysis of pDARTS shows how to cope with the quite special situation of fault-tolerant distributed algorithms developed for VLSI circuits: Such algorithms are made up of a typically large number of simple building blocks (a multiplication is already non-trivial here) consisting of logic gates, interconnected by simple wires carrying boolean signals, which “compute” asynchronously, continuously and concurrently. Our modeling framework, which is based on continuous time and FIFO channels with delay, facilitates “switching” between different — but consistent — views (state, transition and counting view) of the same signal. In sharp contrast to existing modeling frameworks capable of expressing timed executions,² these features allow to express the properties of fault-tolerant distributed algorithms designed for a direct implementation in “partially synchronous”³ logic in a natural and simple way.

Detailed contributions: (i) We adapt the Byzantine fault-tolerant distributed tick generation algorithm introduced in [15] for pipelined execution, and make it suitable for a direct implementation in asynchronous digital logic. Like the original DARTS [6], the resulting pipelined pDARTS algorithm achieves this by enforcing certain atomic actions (“interlocking”) via relative timing assumptions, and by replacing multi-bit messages by up/down signal transitions only. (ii) We introduce the relevant parts of our novel modeling and analysis framework for fault-tolerant distributed algorithms designed for a direct implementation in VLSI. (iii) We present the cornerstones⁴ of the correctness proof of the pDARTS algorithm, and the worst case bounds for performance metrics like synchronization precision and minimum/maximum clock frequency. Since our “system-level proof” rests on fundamental properties of simple basic building blocks only, it effectively reduces the complex

²Our framework is substantially different from existing modeling frameworks for asynchronous VLSI circuits (“trace theory”), which are time-free and hence cannot deal with failures [17].

³By “partially synchronous” designs, we mean clockless designs that, in contrast to pure asynchronous designs, also depend on some additional (relative) timing constraints.

⁴Lacking space does not allow us to include the complete proof in the paper. The details are provided in the technical report [18], which also contains an overview of the related work on modeling approaches.

problem of guaranteeing system correctness to the simpler problem of assuring the correctness of the implementations of the basic blocks. (iv) We provide a glimpse of the results of the experimental evaluation of pDARTS in an FPGA prototype system, which confirm the feasibility and efficiency of our approach.

II. INFORMAL OVERVIEW

The basic idea of DARTS⁵ is to replace the common quartz oscillator and the clock tree that disseminates the clock in a SoC by a fully distributed GALS-like approach (globally asynchronous, locally synchronous [19]): Every functional unit Fu_i in the SoC has attached a dedicated fault-tolerant tick generation unit (TG-Alg), which generates Fu_i ’s local clock signal. To accomplish this, all TG-Algs communicate with each other over a simple “network” of clock signals (TG-Net). In contrast to GALS, however, DARTS ensures that the local clock signals of different Fu ’s are synchronized to each other to within a few clock cycles. In [20], it was proved that such loose synchrony suffices to implement metastability-free high-speed communication between different Fu ’s driven by fault-free clocks. Thus SoCs built atop of DARTS do not need asynchronous communication mechanisms, i.e., handshaking. Besides fault-tolerance, DARTS clocks (patented in [21]) provide a number of additional advantages, making them particularly promising for critical applications e.g. in the aerospace domain.

Like DARTS, the pipelined pDARTS TG-Alg developed and analyzed in this paper derives from a simple distributed algorithm, namely, a synchronizer for the Θ -Model [22], [23] introduced in [15]. Its pseudo-code description is given in Figure 1; X is a system parameter determined by the inherent pipeline depth of the system: If X is chosen well (see Section V for dimensioning issues), then pDARTS will generate clock ticks with a frequency that is independent of the TG-Net delay. Note that, at this level of description, pDARTS for $X = 0$ is the same as the original DARTS.

The algorithm assumes a message-driven system (where nodes make atomic receive-compute-send steps whenever they receive a message) of $n = 3f + 1$ nodes (= TG-Alg instances), at most f of which may behave arbitrarily faulty, i.e., Byzantine. The nodes are connected by a reliable⁶ point-to-point message-passing network (= TG-Net): No spurious messages are ever generated by the network, no messages are lost or altered, and all messages sent at time t are received within the interval $t + [\tau^-, \tau^+]$, where τ^- (resp. τ^+) denotes the (possibly unknown) lower (resp. upper) bound on the end-to-end delay of messages exchanged between correct nodes. Let $\varepsilon = \tau^+ - \tau^-$ be the maximum uncertainty of the message delay, and $\Theta = \tau^+ / \tau^-$ the maximum delay ratio.

The algorithm of Figure 1 works as follows: Initially, every node broadcasts $\text{tick}(-X), \dots, \text{tick}(0)$. Note that all

⁵See our project web page ti.uwien.ac.at/darts for further details.

⁶This reliable network assumption is not unduly restrictive, since communication failures can be mapped to failures of the sending node.

```

1: VAR k: integer /* Local clock value */
2: send tick(-X) ... tick(0) to all /* At booting time */
3: k:=0
4: if received tick( $\ell$ ) from at least  $f + 1$  distinct nodes with  $\ell > k$  then
5:   send tick( $k + 1$ ) ... tick( $\ell$ ) to all [once]
6:    $k := \ell$ 
7: if received tick( $\ell$ ) from at least  $2f + 1$  distinct nodes, with  $\ell \geq k - X$ 
   then
8:   send tick( $k + 1$ ) to all [once]
9:    $k := k + 1$ 

```

Figure 1. TG-Alg for pDARTS

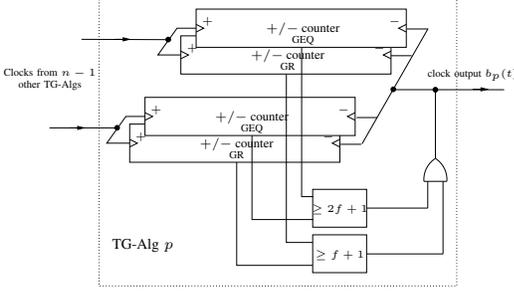


Figure 2. Basic architecture of a pDARTS TG-Alg.

correct nodes are assumed to initialize (almost) at the same time, in the sense that they don't lose any message due to late booting. If node p receives $f + 1$ tick(ℓ) messages (line 4), it can be sure that at least one of those was broadcast by a correct node. Therefore, p can safely catch up and send tick($k + 1$), ..., tick(ℓ). If some correct node p receives $2f + 1$ tick($k - X$) messages (line 7) and thus broadcasts tick($k + 1$), one can be sure that all messages among the $2f + 1$ ones that were broadcast by correct nodes, i.e., at least $f + 1$, will be received within ε by every other node. Hence, every correct node will execute line 4 and send tick($k + 1$) and tick($k - X$) occur quite close (within ε) to each other. This property is called Quasi-Simultaneity (QS), and is a key property in proving the correctness of the algorithm.

Our detailed analysis will reveal that this indeed suffices to prove that correct nodes generate a sequence of consecutive messages tick(k), $k \geq 1$, in a synchronized way (see Section III-D): If $b_p(t)$ denotes the number of tick messages broadcast so far by the TG-Alg at node p , which is equal to the value of the variable k (cf. Figure 1), it turns out that $(t_2 - t_1)\alpha_{\min} \leq b_p(t_2) - b_p(t_1) \leq (t_2 - t_1)\alpha_{\max}$ for any correct p and $t_2 - t_1$ sufficiently large (the property is called “accuracy”); the constants α_{\min} and α_{\max} depend on τ^- , τ^+ and X . Moreover, every two correct nodes p, q maintain $|b_p(t) - b_q(t)| \leq \pi$ (the property is called “precision”), for a small constant π that depends on Θ and X only.

Comparison with the original DARTS algorithm [6] reveals that only the progress rule (line 7) was changed in order to implement pipelining:⁷ Rather than just waiting for tick(ℓ) with $\ell \geq k$, as in the original DARTS algorithm, the pDARTS algorithm waits for $\ell \geq k - X$. Our first intuition

⁷Note that our “top-down approach” for incorporating pipelining is quite different from the way it is usually done in VLSI design.

was to incorporate X also in the catchup rule (line 4), which would have considerably reduced the complexity of the low-level hardware implementation of pDARTS. It turned out, however, that doing this causes the algorithm to fail.

Since the pDARTS algorithm in Figure 1 looks very simple, it is tempting to conclude that it is easily implemented in hardware: Node p 's TG-Alg just needs to drive a boolean-valued clock signal, where it outputs the k -th signal transition when it sends its tick(k) message; the TG-Net is formed by feeding all clock signals to all TG-Algs (e.g., by a bus). In [6], however, it turned out that several challenging design issues must be solved in order to arrive at a low-level version of the pDARTS TG-Alg as depicted in Figure 2. The major building blocks of a single TG-Alg are $2(n - 1)$ custom $+/-$ counters, two for each of the $n - 1$ other TG-Algs q in the system. The two $+/-$ counters for remote TG-Alg q at local TG-Alg p are responsible for maintaining $\ell_q - k_q > 0$ (resp. $\ell_q - k_q \geq -X$), which are required to implement line 4 (resp. line 7) in Figure 1. Herein, ℓ_q denotes the number of tick-messages seen from q at the $+/-$ counter so far, whereas k_q denotes the number of already perceived tick-messages from the own TG-Alg p . The status signals GR (resp. GEQ) signal when the corresponding inequality holds. In addition, a “ $\geq f + 1$ ” (resp. “ $\geq 2f + 1$ ”) threshold circuit implements the rules in line 4 (resp. line 7). Finally, there is a device (shown as an OR-gate in Figure 2), responsible for generating every local clock tick exactly once from the outputs of the threshold gates. These local ticks are not only broadcast to all $n - 1$ remote TG-Algs via remote links, but are also fed back to each of the $2(n - 1)$ $+/-$ counters locally.

In order to circumvent the circuit from producing glitches, a commonly used modification is applied: The circuit is made to operate in two alternating phases (even, odd). The signal GR , for example, was split into two signals GR^e and GR^o , tied to even and odd ticks, respectively: GR^e is true iff the inequality $\ell_q - k_q > 0$ holds and $k_q \in \mathbb{N}_{\text{even}} := 2\mathbb{N}$, whereas GR^o is true iff the inequality $\ell_q - k_q > 0$ holds and $k_q \in \mathbb{N}_{\text{odd}} := 2\mathbb{N} + 1$. The associated threshold circuit is also duplicated, providing one for the GR^e signals and one for the GR^o signals. The same splitting is done for the signals GEQ and their threshold circuits. Finally, all their outputs are combined to generate p 's odd and even ticks.

III. FORMALIZATION

It has been highlighted in the previous section that even the simple distributed algorithm presented in Figure 1 makes use of design elements that are not available or too costly at the hardware implementation level. Correctness proofs and performance analyses using standard distributed computing models would hence suffer from a substantial “proof gap” w.r.t. the actual implementation, which considerably diminishes their value: Although the high-level algorithm has been proved correct, it is likely that the implementation does not match the algorithm. Consequently, we base our

formal analysis on a more low-level model for specifying the building blocks and analyzing the execution of fault-tolerant distributed algorithms that are implemented directly in asynchronous digital logic. Note that our framework is substantially different from existing modeling frameworks for asynchronous VLSI circuits like trace theory, which are time-free and hence cannot deal with failures [17], and is also different from the framework used for the correctness proof and performance analysis [6] of the original DARTS.

VLSI distributed algorithms typically have a hierarchical structure: For example, the top-level of pDARTS is made up of n TG-Algs interconnected via the signal wires making up the TG-Net. Every TG-Alg can be further partitioned into several building blocks (like the $+/-$ counters), which are interconnected in some non-regular way. Our model abstracts from these internals by considering *modules*, which possess input and output *ports* (boolean signals). A module's *behavior* specifies how the signals on the input and output ports are related. Modules differ from timed automata [24] primarily in that they *continuously* compute their outputs, based on the history of their inputs.

Compound modules consist of multiple sub-modules and their interconnect, which specifies how sub-module ports are connected to each other and to the module's input/output ports. The interconnect specification itself assumes zero delays; modeling non-zero interconnect delays, e.g., for real wires, requires intermediate channels: A channel possesses a single input port and a single output port, and its behavior specifies delayed FIFO delivery of input port signal transitions at the output port. Modules that are not further refined are called *basic modules*. Basic modules are zero-delay boolean functions (AND, OR, ...) and channels.

Clearly, the behavior of a (non-faulty) composite module is determined by the behavior of its constituent sub-modules; the behavior of a basic module must be given *a priori*. An execution of a system is specified by the behaviors of each of its signals, and is typically modeled as a set of event traces (see below). Correctness proofs establish properties of the behaviors of higher-level modules, based on the assumption that (1) the system and failure model holds, and (2) that the implementations of the basic modules indeed satisfy their behavioral specification.

A. Signals and zero-bit message channels

Since we target implementations using asynchronous circuits, our formal framework will be based on a continuous notion of real-time $t \in \mathbb{R}_0^+$. We assume that the system initialization (reset) occurs at time $t = 0$.

A *signal* S may be either represented (i) by its event trace, (ii) by its status, or (iii) by its counting function. These representations are consistent, in a well-defined way, and can hence be used interchangeably.

(i) Event trace: The representation of S by an event trace, denoted by \widehat{S} , is specified by a relation $\widehat{S} \subseteq \mathbb{R}_0^+ \times \{0, 1\}$, where event $(t, 1) \in \widehat{S}$ (resp. $(t, 0) \in \widehat{S}$) means that S takes on value 1 (resp. 0) at real-time t . In order to enforce a

unique initial value, we require either $(0, 1) \in \widehat{S}$ or $(0, 0) \in \widehat{S}$. We further demand non-simultaneity of contradicting events for any single signal, i.e., $((t, x) \in \widehat{S}) \wedge ((t, y) \in \widehat{S}) \Rightarrow (x = y)$. Let $\text{pre}(\widehat{S}, t) := \{(t', v') \in \widehat{S} \mid t' \leq t\}$ denote the prefix of \widehat{S} until time t . In this paper, we will restrict our attention to event traces where only finitely many alternating events (i.e., with different value) can occur in any finite time interval. Due to this restriction,

$$\begin{aligned} \text{last-val}(\widehat{S}, t) &:= v' \text{ s.t. } \exists (t', v') \in \text{pre}(\widehat{S}, t) : \\ &\forall (t'', v'') \in \text{pre}(\widehat{S}, t) : (t'' \geq t') \Rightarrow (v'' = v') \end{aligned}$$

is always well-defined. This definition still allows arbitrarily many idempotent events to occur in a prefix.

(ii) Status: Since idempotent events do not change the state of a signal, they are often irrelevant. Idempotent events can be abstracted away by considering the signal S 's status representation, denoted by \widetilde{S} , which is a function $\widetilde{S} : \mathbb{R}_0^+ \rightarrow \{0, 1\}$ from time t to the boolean value of S at time t . Obviously, signals may be composed out of already defined signals by using arbitrary boolean predicates, e.g., $\widetilde{A} := \widetilde{B} \wedge \widetilde{C}$, with signals $\widetilde{B}, \widetilde{C}$ defined as $\widetilde{A}(t) := \widetilde{B}(t) \wedge \widetilde{C}(t)$.

One can easily switch between the two representations of S : If given the event trace \widehat{S} of S , the equivalent status representation \widetilde{S} of S can be obtained by $\widetilde{S}(t) := \text{last-val}(\widehat{S}, t)$. Given \widetilde{S} , one cannot regenerate the "original" event trace \widehat{S} , since all idempotent events have been lost. Still, most of the time, it suffices to obtain *some* event trace \widehat{S}' that has the same status representation as \widehat{S} . Such an event trace can simply be obtained from \widetilde{S} by $\widehat{S}' := \{(t, \widetilde{S}(t)) \mid t \geq 0\}$; here \widehat{S}' contains continuum-many events.

(iii) Counting function: Finally, signal S can be represented by the number of non-idempotent events that occur during $]0, t]$, denoted as the counting function $S(t)$. For example, if $\widehat{S} = \{(0, 0), (1, 1), (1.5, 1), (2, 0)\}$, then $S(0) = 0$ and $S(2) = 2$. Sometimes, we will also employ generalized counting functions $S'(t)$ that have an initial value other than 0: We define $S'(t) := S(t) + S_0$, where $S(t)$ is the standard counting function of S and S_0 an arbitrary offset.

It is again easy to switch between the counting function representation of S and the other representations: $S(t)$ can be obtained from \widehat{S} by just counting the non-idempotent events in \widehat{S} (excluding the initial event), and getting \widehat{S}' from $S(t)$ [for some given initial event $(0, I)$, $I \in \{0, 1\}$] is accomplished via $\widehat{S}' := \{(t, [I + S(t)] \bmod 2) \mid t \geq 0\}$. Switching between the status representation and the counting function can be done transitively via the corresponding event trace representation, or directly via $\widetilde{S}(0) := 0$, $\widetilde{S}(t) := [I + S(t)] \bmod 2$ for $t > 0$.

A threefold representation of signals may look counter-productive in a modeling approach, which typically aims at unification (i.e., deriving properties from as few concepts as possible). However, the benefit of equipping the system model with three different representations of a signal becomes clear when considering behavioral specifications,

correctness proofs and performance property proofs of real-world designs. Consider a Muller C-Element [25], for example, a component extensively used in asynchronous designs, with input signals a, b and output y . It can be specified by using both status and event representations (whereas a specification in e.g., status only would be lengthy):

$$(0, 0) \in \widehat{y} \text{ and} \\ (t, 1) \in \widehat{y} \Leftrightarrow \widetilde{a}(t) \wedge \widetilde{b}(t), \quad (t, 0) \in \widehat{y} \Leftrightarrow \neg \widetilde{a}(t) \wedge \neg \widetilde{b}(t).$$

A combinatorial element, like an AND with inputs a, b and output y can easily be specified by status representations:

$$\widetilde{y}(t) = \widetilde{a}(t) \wedge \widetilde{b}(t).$$

A specification via events would be lengthy and distracting. Finally, the counting function representation is useful when specifying the behaviour of queueing systems (consult the Diff-gate specification in Section III-C for an example).

A *channel* models a reliable FIFO channel for signal transitions with finite delay. Since signal transitions must be alternating, only the occurrence time but no data can be conveyed over a single channel. Formally, the semantics of a channel \mathcal{C} is as follows: Let \mathcal{C}^s be the channel's single input port [which will be connected to an output port of a single sender module], and \mathcal{C}^r be its single output port [which will be connected to the input ports of some receiver module(s)]. There exists a continuous and strongly monotonically increasing *delivery function* $f : \mathbb{R}_0^+ \rightarrow \mathbb{R}_0^+$ for \mathcal{C} , which maps sending time t to delivery time $f(t)$. We assume that the channel delay is within $[\tau_{\mathcal{C}}^-, \tau_{\mathcal{C}}^+]$, i.e., $f(t) - t \in [\tau_{\mathcal{C}}^-, \tau_{\mathcal{C}}^+]$. From the properties of f , it follows immediately that f is a bijection from \mathbb{R}_0^+ to its codomain $f(\mathbb{R}_0^+)$. More specifically, f maps every closed interval $[t_1, t_2]$ bijectively to the closed interval $[f(t_1), f(t_2)]$. Clearly, the inverse function f^{-1} of f also exists and has the same properties. In addition, we will assume that the channel output has some well-defined initial state (is initialized to) $I \in \{0, 1\}$, which is $I = 0$ if not specified otherwise. Given f , the channel's behavior in terms of event traces is

$$(0, I) \in \widehat{\mathcal{C}}^r \wedge \nexists (t, v) \in \widehat{\mathcal{C}}^r \text{ with } v \in \{0, 1\}, t \in [0, f(0)[\\ \text{and } (f(t), x) \in \widehat{\mathcal{C}}^r \Leftrightarrow (t, x) \in \widehat{\mathcal{C}}^s.$$

Since f carries over the total order of the events (t, x) in $\widehat{\mathcal{C}}^s$ to the events $(f(t), x)$ in $\widehat{\mathcal{C}}^r$ [called *matching* events in the sequel], it follows that $\widehat{\mathcal{C}}^r$ is an event trace. In terms of states, the channel behavior can be defined as

$$\forall t \in [0, f(0)[: \widetilde{\mathcal{C}}^r(t) := \widetilde{\mathcal{C}}^r(0) = I \text{ and} \\ \forall t \geq 0 : \widetilde{\mathcal{C}}^r(f(t)) := \widetilde{\mathcal{C}}^s(t).$$

B. pDARTS System Model

The pDARTS system consists of a set P of $n \in \mathbb{N}$ top-level modules. These top-level modules will interchangeably be called node/TG-Alg and are usually denoted by letters p, q etc. Every node p has exactly one output port with the counting function $b_p(t)$ (the number of broadcast messages),

and one input port per remote node $q \in P \setminus \{p\}$ with the counting function $r_{p,q}^{rem}(t)$ (the number of *received* messages). We assume a fully connected system, i.e., from every node p to every node $q \in P \setminus \{p\}$, there is a channel $\langle REM, p, q \rangle$ with input $b_p(t)$, output $r_{q,p}^{rem}(t)$, and delay in $[\tau_{rem}^-, \tau_{rem}^+]$. Figure 3 shows the resulting outbound channels of node p . Let $t_{p,boot}$ denote the time when correct node p completes booting (starts executing its TG-Alg). We require that $t_{p,boot} \in [0, B]$ for some constant B , where $B \leq \tau_{rem}^-$. Due to this assumption, messages sent by p may not get lost at any correct node q because of late booting.

The following notation will be used throughout the paper: For any $k \geq 1$, we say that node p sends tick k , at time $t_{p,k}$, if the k^{th} event (without counting idempotent events) occurs at $t_{p,k}$. The time when the first (resp. the last) correct node sends tick k is denoted by $t_{first,k}$ (resp. $t_{last,k}$). Analogously, we say that p receives tick k from q at time t , if $r_{p,q}^{rem}(t) = r_{p,q}^{rem}(t^-) + 1 = k$, where t^- is the time immediately before the reception takes place.

We partition our system into multiple *fault-containment regions* (FCRs), i.e., sets of modules that are potentially affected by a single fault like a particle hit and thus cannot be assumed to fail independently. More specifically, we define FCR p to consist of the single node p together with all its outgoing channels, as depicted in Figure 3. If FCR p is correct, then each of its sub-modules behaves as specified in Section III-C. If FCR p is faulty, any of its sub-modules may behave arbitrarily (Byzantine).⁸ Since every FCR is associated with exactly one node, we will use these terms interchangeably as well. Throughout the paper, let C be the set of correct FCRs, and F , with $f := |F|$, the set of faulty FCRs. Clearly $P = C \cup F$ and $C \cap F = \emptyset$, i.e., C and F partition P . We will prove in Section IV that correct nodes behave as specified in Section III-D in the presence of up to f Byzantine faulty FCRs, provided that the total number of nodes is $n \geq 3f + 2$. This is slightly more than the required lower bound of $n \geq 3f + 1$ for clock synchronization [32], but facilitates a considerably better precision and accuracy.⁹

C. Specifications of TG-Alg basic modules

The internal architecture of a single TG-Alg, as described in Section II, is obtained by expressing and refining Figure 2 in terms of our formal model. Due to space limitations, we

⁸Since hardware faults easily lead to Byzantine failures [26], we assume this failure semantics here: The adverse power of Byzantine failures in our context lies in the ability of faulty nodes to generate wrong clock ticks (early or even spurious) that are perceived inconsistently at different remote nodes. Such failures can be the consequence of manufacturing defects or electrostatic breakdown [27], particle hits [28], or electromagnetic noise [29], which may affect any module in a TG-Alg. Due to different wire lengths and signal-level detection thresholds, such faults typically propagate differently to different receivers. We allow faulty nodes to create even metastability [30], but must assume that metastability cannot propagate beyond FCRs; we have already convincing evidence [31] that this is ensured by the elastic pipelines in the $+/-$ counters with high probability.

⁹This follows from counting only remote messages when calculating the $f + 1$ and $2f + 1$ thresholds; including self-reception would lead to $\tau_{rem}^- = \tau_{loc}^-$ in Theorem 2, which spoils the achievable worst-case precision considerably.

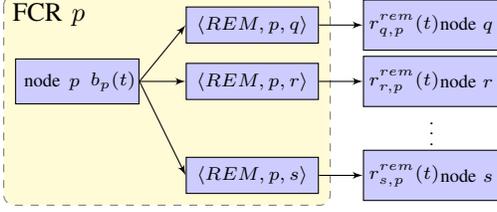


Figure 3. Fault-containment region FCR p .

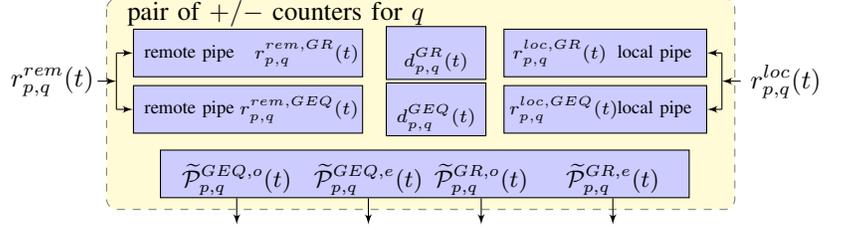


Figure 4. Architecture of a pair of $+/-$ counters at p corresponding to q

will only present the formalization of the $+/-$ counters here, which turned out to be the most intricate component.

As shown in Figure 4, the implementation of every pair of $+/-$ counters comprises two $+/-$ counters. Each $+/-$ counter consists of two elastic pipelines [25], which can be seen as shift registers/FIFO buffers 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*, which removes “matching” transitions as soon as they traveled through the pipelines. The status signals GEQ^o , GEQ^e , GR^o , and GR^e are provided by the *pipe compare signal generator (PCSG)* circuits, which monitor the last few stages of both pipes. For simplicity, we did not include the channels arising in the sub-modules’ interconnect in Figure 4.

Pairs of elastic pipes: Every node p incorporates two pairs of elastic pipelines for every remote TG-Alg $q \in P \setminus \{p\}$. The pipepair responsible for the GEQ -rule is denoted $(p, q)_{GEQ}$, the one for the GR -rule $(p, q)_{GR}$. Every pair consists of a remote pipeline that can store up to $S_{rem,geq}$ (resp. $S_{rem,gr}$) ticks sent by q , and a local pipeline that can hold up to $S_{loc,geq}$ (resp. $S_{loc,gr}$) ticks generated by p locally. The numbers $S_{rem,geq}$, $S_{rem,gr}$, $S_{loc,geq}$ and $S_{loc,gr}$ are implementation parameters that have to be chosen in accordance with the bounds of Section III-D; in the specifications of this section, they are just assumed to be unbounded (arbitrary large).

The local pipe of $(p, q)_{GEQ}$ has a single input port represented by the counting function $r_{p,q}^{loc}(t)$, which is fed by TG-Alg p ’s local clock ticks $b_p(t)$ supplied via a (local) channel $\langle LOC, p, q \rangle$, and a single output port represented by the counting function $r_{p,q}^{loc,GEQ}(t)$, which denotes the number of ticks that arrived at the output of the local pipe by time t . Similarly, the remote pipe of $(p, q)_{GEQ}$ has a single input port represented by the counting function $r_{p,q}^{rem}(t)$, which is fed by TG-Alg q ’s clock ticks supplied via the (remote) channel $\langle REM, q, p \rangle$, cp. Figure 3, and a single output port represented by the counting function $r_{p,q}^{rem,GEQ}(t)$, which denotes the number of ticks that arrived at the output of the remote pipe by time t . The same description applies to the pipepair $(p, q)_{GR}$, except that $r_{p,q}^{rem,GEQ}(t)$ and $r_{p,q}^{loc,GEQ}(t)$ is replaced by $r_{p,q}^{rem,GR}(t)$ and $r_{p,q}^{loc,GR}(t)$, respectively. Upon initialization, the remote pipe of every $(p, q)_{GEQ}$ is prefilled with “virtual” ticks $-X, \dots, 0$ (expressed by a counting function with offset

X), while the other pipes are initialized with “virtual” tick 0; we call these ticks “virtual”, since they were never sent.

Behavioral Description: Every pipeline has the behavior of a zero-delay¹⁰ channel: $r_{p,q}^{rem,GR}(t) := r_{p,q}^{rem}(t)$, $r_{p,q}^{rem,GEQ}(t) := r_{p,q}^{rem}(t) + X$, $r_{p,q}^{loc,GR}(t) = r_{p,q}^{loc,GEQ}(t) := r_{p,q}^{loc}(t)$, where $X \geq 0$ is the pipeline depth parameter. The initial values are $r_{p,q}^{rem,GEQ}(t_{p,boot}) = X$ (to account for the prefilling) and $r_{p,q}^{rem,GR}(t_{p,boot}) = r_{p,q}^{loc,GR}(t_{p,boot}) = r_{p,q}^{loc,GEQ}(t_{p,boot}) := 0$.

Diff-Gate: To avoid pipes with infinite capacity, each pair of pipes is equipped with a special Diff-Gate circuit that removes matching clock ticks from their outputs, i.e., clock ticks contained in both pipes: The Diff-Gate for $(p, q)_{GEQ}$ has two input ports connected to $r_{p,q}^{rem,GEQ}(t)$ and $r_{p,q}^{loc,GEQ}(t)$, and a single output port represented by the counting function $d_{p,q}^{GEQ}(t)$, which gives the largest tick number that has been removed from *both* the remote and local pipe of $(p, q)_{GEQ}$ by time t . [The description of the Diff-Gate of $(p, q)_{GR}$ is very similar.]

Behavioral Description: Let $t_{rmv,k}^{GEQ}$, $k \geq 0$, be the time when tick k is removed from both the remote output $r_{p,q}^{rem,GEQ}(t)$ and the local output $r_{p,q}^{loc,GEQ}(t)$ of $(p, q)_{GEQ}$, i.e., $d_{p,q}^{GEQ}(t_{rmv,k}^{GEQ}) = k$. Tick k stored in the remote pipeline actually is tick $k+X$ showing up at the output $r_{p,q}^{rem,GEQ}(t)$. (i) By convention, the remote pipe initially is prefilled with ticks $-X, \dots, 0$. Thus ticks $0, \dots, X$ show up at the output $r_{p,q}^{rem,GEQ}(t_{rem,k}^{GEQ})$ at time $t_{rem,k}^{GEQ} := t_{p,boot}$. Further, by convention, tick -1 is removed at time $t_{rmv,-1}^{GEQ} := t_{p,boot}$, i.e., $d_{p,q}^{GEQ}(t_{p,boot}) = -1$. (ii) For $k \geq 0$, if tick $k+1$ shows up at $r_{p,q}^{rem,GEQ}(t_{rem,k+1}^{GEQ})$ at time $t_{rem,k+1}^{GEQ}$, and tick $k+1$ shows up at $r_{p,q}^{loc,GEQ}(t_{loc,k+1}^{GEQ})$ at time $t_{loc,k+1}^{GEQ}$, and tick $k-1$ is removed at time $t_{rmv,k-1}^{GEQ}$, then tick k is removed at some $t_{rmv,k}^{GEQ} \in \max\{t_{rem,k+1}^{GEQ}, t_{loc,k+1}^{GEQ}, t_{rmv,k-1}^{GEQ}\} + [\tau_{Diff}^-, \tau_{Diff}^+]$.

On top of the above defined signals $r_{p,q}^{loc}(t)$ and $d_{p,q}^{GEQ}(t)$, the size of the local pipe of $(p, q)_{GEQ}$ at time t is defined as $s_{p,q}^{loc,geq}(t) := r_{p,q}^{loc}(t) - d_{p,q}^{GEQ}(t)$. The other pipes’ sizes $s_{p,q}^{rem,geq}(t)$, $s_{p,q}^{loc,gr}(t)$ and $s_{p,q}^{rem,gr}(t)$ are defined similarly.

Pipe Compare Signal Generators (PCSGs): The signals provided by the pipepairs and their Diff-Gates are connected to the PCSG, which generates four status signals

¹⁰Actually, the pipe delays are accounted for in the delays of the other channels in the signal path, namely, $\langle REM, q, p \rangle$ and $\langle LOC, p, q \rangle$.

$\widetilde{\mathcal{P}}_{p,q}^{GEQ,o}(t)$, $\widetilde{\mathcal{P}}_{p,q}^{GEQ,e}(t)$, $\widetilde{\mathcal{P}}_{p,q}^{GR,o}(t)$ and $\widetilde{\mathcal{P}}_{p,q}^{GR,e}(t)$ that characterize the difference of the number of clock ticks stored in the remote and local pipes by time t . Different signals are provided for odd and even clock ticks. For example, $\widetilde{\mathcal{P}}_{p,q}^{GEQ,o}(t)$ signals when the number of remote clock ticks is greater than or equal to the number of local clock ticks, provided that the last clock tick that entered the local pipe was odd. All these signals are fed, via dedicated channels that add some delay, to the threshold modules of the TG-Alg p . Note that we need $\widetilde{\mathcal{P}}_{p,q}^{GEQ,o}(t)$ and $\widetilde{\mathcal{P}}_{p,q}^{GR,o}(t)$ to be valid only if the local pipes contain exactly one tick. These signals are fed into dedicated channels, all of which are initialized to 0: $\langle \mathcal{P}^{GEQ}toGEQ, o, p, q \rangle$ with input $\widetilde{\mathcal{P}}_{p,q}^{GEQ,o}(t)$, output $\widetilde{GEQ}_{p,q}(t)$ and delay $[\tau_{GR}^-, \tau_{GEQ}^+]$, and $\langle \mathcal{P}^{GR}toGR, o, p, q \rangle$ with input $\widetilde{\mathcal{P}}_{p,q}^{GR,o}(t)$, output $\widetilde{GR}_{p,q}(t)$ and delay $[\tau_{GR}^-, \tau_{GR}^+]$. [Analogously, $\widetilde{\mathcal{P}}_{p,q}^{GEQ,e}(t)$ and $\widetilde{\mathcal{P}}_{p,q}^{GR,e}(t)$ and their channels are defined by substituting \mathbb{N}_{odd} with \mathbb{N}_{even} .]

Threshold modules, tick generation module and interconnect: The GR (resp. GEQ) signals from the PCSGs are fed into the threshold modules, which signal within delay $[\tau_{TH}^-, \tau_{TH}^+]$ whether the $f+1$ (resp. $2f+1$) threshold has been reached. The thresholds' outputs are finally combined by the tick generation module, which actually produces every tick exactly once, and broadcasts it via the channels $\langle REM, p, q \rangle$ (within delay $[\tau_{rem}^-, \tau_{rem}^+]$) to all other TG-Algs, and to its own $+/-$ counters via the channels $\langle LOC, p, q \rangle$ (within delay $[\tau_{loc}^-, \tau_{loc}^+]$).

D. System-level properties

Correct TG-Algs are required to guarantee the properties:

(P) Precision (see Theorem 2): There is a constant π , such that for every pair of correct nodes $p, q \in C$:

$$\forall t : |b_q(t) - b_p(t)| \leq \pi. \quad (1)$$

(A) Accuracy (see Theorem 3): There are constants $R^-, O^-, R^+, O^+ > 0$, such that for every correct node $p \in C$:

$$O^-(t_2 - t_1) - R^- \leq b_p(t_2) - b_p(t_1) \leq O^+(t_2 - t_1) + R^+. \quad (2)$$

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

$$s_{p,q}^{loc,gr}(t) \leq S_{loc,gr}, s_{p,q}^{rem,gr}(t) \leq S_{rem,gr}, \\ s_{p,q}^{loc,geq}(t) \leq S_{loc,geq}, \text{ and } s_{p,q}^{rem,geq}(t) \leq S_{rem,geq}.$$

In the following Section IV, we will sketch the cornerstones of our proofs, which show that the TG-Algs at correct nodes indeed satisfy the above properties in all executions complying to the system and failure model, provided that (a) the implementations of correct basic modules specified in Section III-C indeed fulfill their specifications, and (b) additional ‘‘global’’ Constraints 1–3, which are relative timing constraints, hold. Our Theorems 2 and 3 will also establish numerical values for precision and accuracy, which only depend on X and the delay parameters introduced in the specifications of the TG-Alg sub-modules in Section III-C.

IV. CORRECTNESS PROOFS

The first cornerstone of our proofs¹¹ is the ‘‘Interlocking Lemma’’, which states that an ‘‘old’’ tick $k-2, k-4, \dots$ is never mixed up with a ‘‘recent’’ tick k when generating tick $k+1$. This property does not come for free, however, but can be guaranteed to hold only if the following timing constraint is satisfied (which is easy to enforce via a suitably defined constraint during place-and-route of a VLSI circuit):

Constraint 1: (Interlocking Constraint). With the abbreviations $T_{max} := \tau_{TH}^+ + \max(\tau_{GR}^+, \tau_{GEQ}^+) + \tau_{loc}^+$, $T_{min} := \tau_{TH}^- + \min(\tau_{GR}^-, \tau_{GEQ}^-) + \tau_{loc}^- + \tau_{Diff}^-$ and $T_{min,dis} := \tau_{TH}^- + \min(\tau_{GR}^-, \tau_{GEQ}^-) + \tau_{loc}^-$, it must hold that $T_{max} \leq T_{min} + T_{min,dis}$.

Lemma 1 (Interlocking): If, for some correct node p and $k' = k+1 \geq 2$, $b_p(t) = k+1$, then

(i) either there exists a set Q of size $|Q| \geq 2f+1$ s.t. for $t' := t - \tau_{TH}^- - \tau_{GEQ}^-$:

$$k \in \mathbb{N}_{even} \Rightarrow \forall q \in Q : \exists t_q \leq t' : \widetilde{\mathcal{P}}_{p,q}^{GEQ,e}(t_q) \wedge r_{p,q}^{loc}(t_q) \geq k \\ k \in \mathbb{N}_{odd} \Rightarrow \forall q \in Q : \exists t_q \leq t' : \widetilde{\mathcal{P}}_{p,q}^{GEQ,o}(t_q) \wedge r_{p,q}^{loc}(t_q) \geq k,$$

(ii) or there exists a set Q of size $|Q| \geq f+1$ s.t. for $t' := t - \tau_{TH}^- - \tau_{GR}^-$:

$$k \in \mathbb{N}_{even} \Rightarrow \forall q \in Q : \exists t_q \leq t' : \widetilde{\mathcal{P}}_{p,q}^{GR,e}(t_q) \wedge r_{p,q}^{loc}(t_q) \geq k \\ k \in \mathbb{N}_{odd} \Rightarrow \forall q \in Q : \exists t_q \leq t' : \widetilde{\mathcal{P}}_{p,q}^{GR,o}(t_q) \wedge r_{p,q}^{loc}(t_q) \geq k.$$

This result enables us to prove a minimum duration between two successive ticks generated by a correct node:

Lemma 2: If correct node p sends tick $k \geq 1$ at time $t_{p,k}$, it cannot send tick $k+1$ before $t_{p,k} + T_{min}$.

The following Lemma 3 in conjunction with an additional Constraint 2 allows us to exclude the possibility of queuing effects in a pipepair $(p, q)_{GR}$ corresponding to correct node q in a correct TG-Alg p . [An analogous lemma exists for $(p, q)_{GEQ}$.]

Constraint 2: $\tau_{Diff}^+ \leq T_{min}$.

Lemma 3: For any pair of distinct correct nodes p, q and $k \geq 1$: If correct node p sent tick k at $t_{p,k}$ and q sent tick k at $t_{q,k}$, then tick $k-1$ is removed from the local and remote pipe of pipepair $(p, q)_{GR}$ by time $\max\{t_{k,p} + \tau_{loc}^+, t_{k,q} + \tau_{rem}^+\} + \tau_{Diff}^+$, if Constraint 2 holds.

The following pivotal Theorem 1 and its proof are a generalization of well-known properties of consistent broadcasting, cp. [14], [15], [33]. To hold true, the following additional timing constraint must be satisfied:

Constraint 3: $T_{first}^- \geq (X+1)(\tau_{loc}^+ + \max\{\tau_{Diff}^+, \tau_{GR}^+, \tau_{GEQ}^+\} + \tau_{TH}^+)$.

Theorem 1: (Synchronization Properties). pDARTS satisfies the properties Unforgeability (U), Progress (P), Quasi-Simultaneity (QS) and Booting-Simultaneity (BS), if Constraints 1, 2, 3 and $n \geq 3f+2$ hold.

¹¹Lacking space does not allow us to present the complete correctness proof and performance analysis here, which can be found in the technical report [18].

$$\begin{aligned}
\pi &:= \max \left\{ \left\lfloor \frac{T_{QS}}{T_{first}^-} \right\rfloor (X+1) + \left\lfloor \frac{T_{QS} - \left\lfloor \frac{T_{QS}}{T_{first}^-} \right\rfloor T_{first}^-}{T_{min}} \right\rfloor + X+1, \left\lfloor \frac{T_{QS}}{T_{first}^-} \right\rfloor (X+1) + \left\lfloor \frac{T_{QS} - \left(\left\lfloor \frac{T_{QS}}{T_{first}^-} \right\rfloor - 1 \right) T_{first}^-}{T_{min}} \right\rfloor \right\} \\
L(t_2 - t_1) &:= \max \left\{ 0, \left\lfloor \frac{t_2 - t_1 - \max \left\{ \frac{T_{BS}(2X+1), \min \{T_{QS} + (X+1)T_P, T_{BS}(k) \mid k \geq 2X+2\}} \right\}}{T_P} \right\rfloor + 1 \right\} \\
U(t_2 - t_1) &:= \left\lfloor \frac{t_2 - t_1}{T_{first}^-} \right\rfloor (X+1) - \left\lfloor \frac{t_2 - t_1 - \left\lfloor \frac{t_2 - t_1}{T_{first}^-} \right\rfloor T_{first}^-}{T_{min}} \right\rfloor + 1 + \pi
\end{aligned}$$

Figure 5. Precision and (lower and upper) accuracy bounds of pDARTS system.

(U) Unforgeability. If no correct node sends tick $k \geq 1$ by time t , then no correct node sends tick $k + X + 1$ by time $t + T_{first}^-$ or earlier, with $T_{first}^- := \tau_{rem}^- + \tau_{Diff}^- + \tau_{GEQ}^- + \tau_{TH}^-$.

(P) Progress. If all correct nodes send tick $k \geq X + 1$ by time t , then every correct node sends at least tick $k + 1$ by time $t + T_P$, with

$$T_P := \max \left\{ \begin{array}{l} \tau_{loc}^+ + \tau_{Diff}^+ + \max\{\tau_{GEQ}^+, \tau_{GR}^+\}, \\ \tau_{loc}^+ + \tau_{GR}^+, \\ \tau_{rem}^+ + \tau_{Diff}^+ + \tau_{GEQ}^+ - XT_{min} \end{array} \right\} + \tau_{TH}^+ \quad (3)$$

(QS) Quasi-Simultaneity. If some correct node p sends tick $k \geq X + 2$ by time t , then every correct node (including p) sends at least tick $k - X - 1$ by time $t + T_{QS}$, with

$$\begin{aligned}
T_{QS}^1 &:= \tau_{rem}^+ + \tau_{GR}^+ + \tau_{TH}^+ + X(\tau_{loc}^+ \\
&\quad \max\{\tau_{Diff}^+ + \tau_{GR}^+, \tau_{GEQ}^+\} + \tau_{TH}^+ - T_{min}) - T_{first}^- \\
T_{QS}^2 &:= B + (X+1)(\tau_{loc}^+ + \max\{\tau_{GEQ}^+, \tau_{GR}^+\} + \tau_{TH}^+ - \\
&\quad T_{min}) - (\tau_{loc}^+ - \tau_{loc}^-) - T_{first}^- \\
T_{QS} &:= \max\{T_{QS}^1, T_{QS}^2\}.
\end{aligned}$$

(BS) Booting-Simultaneity. If some correct node sends tick $k \geq X + 1$ by time t , then every correct node sends at least tick k by time $t + T_{BS}(k)$, with $T_{BS}(k) := B + \max\{\tau_{GEQ}^+, \tau_{GR}^+\} - \min\{\tau_{GEQ}^-, \tau_{GR}^-\} + \tau_{TH}^+ - \tau_{TH}^- + (k-1)(T_P - T_{min})$.

The (QS) bound comprises two bounds, T_{QS}^2 that holds right after booting, and T_{QS}^1 that holds for larger tick numbers. T_{QS}^1 itself contains three parts (depicted as shaded boxes): The 1st (resp. the 3rd) correspond to the longest (resp. shortest) remote delay; their difference is the uncertainty ε mentioned in the proof idea outlined in Section II. The 2nd term results from queueing effects in pDARTS.

Theorem 1 eventually leads to our major results:

Theorem 2: (Precision). The pDARTS algorithm ensures $\forall t: |b_p(t) - b_q(t)| \leq \pi$ for all correct p, q , with π defined in Figure 5.

Note that π depends on the *ratio* of certain timing parameters only, which typically does not change much when, e.g., one migrates the algorithm to a faster VLSI technology.

The following Theorem 3 allows to relate clock time intervals to real-time intervals, and to make statements about the local clock frequency. For example, it reveals that the long-term frequency is within $\left[1/T_P, (X+1)/T_{first}^-\right]$.

Theorem 3: (Accuracy). Given t_1 and t_2 with $t_2 > t_1 \geq t_{p,X+1}$, the accuracy $b_p(t_2) - b_p(t_1)$ of any correct node p is bounded by the lower and upper bounds $L(t_2 - t_1) \leq b_p(t_2) - b_p(t_1) \leq U(t_2 - t_1)$, defined in Figure 5.

The term $\min\{T_{QS} + (X+1)T_P, T_{BS}(k)\}$ for $k \geq 2X+2$ in $L(t_2 - t_1)$ accounts for the fact that correct nodes may be synchronized very tightly (within $T_{BS}(k)$) after booting, such that $T_{QS} + (X+1)T_P$ would be too conservative. However, when $T_{min} < T_P$, typically being the case in SoCs, the initial synchrony from booting cannot be maintained, i.e., the constant bound from (QS) will be tighter.

Finally, we have proven that the pipeline sizes $S_{rem,geq}$, $S_{rem,gr}$, $S_{loc,geq}$ and $S_{loc,gr}$ at correct TG-Algs are indeed bounded by some constants, which again depend on the ratio of certain timing parameters only.¹²

V. PROTOTYPE IMPLEMENTATION AND MEASUREMENT RESULTS

Since pDARTS uses the same basic blocks as the original DARTS VHDL implementation [6], [16], it was reasonably easy to build an FPGA prototype implementation of pDARTS: Recall that the only substantial change was the doubling of the $+/-$ counters, and the need to initialize all GEQ- $+/-$ counters to X ; the latter was accomplished by putting X ticks, which correspond to the virtual, received ticks $-X, \dots, 0$, into their remote pipes upon reset.

Similar to the first prototype of DARTS, we have synthesized a complete system of 5 pDARTS TG-Algs on an Altera APEX EP20K1000 FPGA. Although FPGAs are not particularly suitable for asynchronous designs due to the fixed structure of the lookup tables and registers, this prototype nevertheless provides a proof of concept and clearly demonstrates the feasibility and efficiency of the pipelined approach.

A comparison of pDARTS with DARTS in terms of area reveals that, despite the doubling of the $+/-$ counters in pDARTS, it scales similar to DARTS: In [34], we have

¹²Detailed results and proofs can be found in the technical report [18].

shown that the threshold modules, which are identical in DARTS and pDARTS, dominate area consumption.

A. Parameter choices

Recall from Section I and II that the pipelining parameter X is related to the inherent pipeline depth of the end-to-end delay paths in the system. Formally, this is expressed in Constraint 3, which requires $T_{first}^- \geq (X + 1)(\tau_{loc}^+ + \max\{\tau_{Diff}^+ + \tau_{GR}^+, \tau_{GEQ}^+\} + \tau_{TH}^+)$. It effectively limits X to a value that allows it to “store” at least X ticks within the fastest end-to-end delay path, even if the X ticks are generated as slowly as possible.

On the other hand, from Theorem 3, it follows that the clock frequency lower bound is $\Omega(1/T_P)$, with T_P given by (3) in Theorem 1. From the term $\tau_{rem}^+ + \tau_{Diff}^+ + \tau_{GEQ}^+ - XT_{min}$ appearing in T_P , it is obvious that, by choosing X sufficiently large, the dependency on a large remote delay bound τ_{rem}^+ can be dropped entirely.

From Theorem 2, it is apparent that the worst case precision π increases when increasing X . It is important to note, however, that this is a matter of a “scaling transformation” and *not* a sign of reduced real-time synchronization quality: Since the precision gives the maximum number of *ticks* two clocks can be off at the same real-time, the increase of π is outweighed by the decrease of the tick duration. What indeed increases when increasing X is the maximum size of the elastic pipelines, however: In order to be able to experiment with different values of $X \in \{0, 2, 4\}$, we had to choose the conservative size of 8 for every elastic pipeline.

B. Measurement results

The above choice of parameters in our FPGA implementation resulted in a local-loop delay (T_{min}) of about 25 ns, which amounts to a maximum local clock frequency of about 20 MHz.¹³ In order to be able to demonstrate the benefits of pipelining in this setting, we enforced a remote delay of $\tau_{rem}^- = \tau_{rem}^+ \approx 125$ ns in the TG-Net. The inherent pipeline depth is hence about $125/25 = 5$. As revealed by the measurement results for $X = 0$, DARTS achieves a clock frequency of about 4 MHz in this case.

Figure 6 shows the case $X = 2$. The repetitive pattern consisting of a burst of 3 ticks followed by some idle time in every clock signal is an artefact of our simple initialization approach, in conjunction with an inherent pipeline depth larger than 3: On reset, the TG-Alg generates $X+1$ ticks and waits until it receives the first tick from sufficiently many remote TG-Alg to generate the next tick. Further increasing X fills up the 125ns clock half period with additional ticks. Figure 7 finally shows the case with the maximum feasible $X = 4$; the next possible value $X = 6$ would already violate Constraint 3. For $X = 4$ we can observe that the clocks run at a frequency of about 20 MHz, which is the maximum frequency of the local loop. This speed-up by a factor of $X + 1 = 5$ confirms that pipelining is able to (completely)

¹³Our FPGA DARTS prototype achieved about 24 MHz, which is due to the fact that the sizes of the elastic pipelines were 4 instead of 8.

hide the large remote delay in the system. Interestingly the initially bursty clock cycles tend to spread out evenly after some time, cp. our comment after Theorem 3.

VI. CONCLUSIONS AND FUTURE WORK

We demonstrated that pipelining is effective for increasing the clock frequency of a fault-tolerant distributed clocking approach in VLSI circuits with large bandwidth \times delay products. We further introduced a new modeling framework for low-level distributed algorithms, which was used to formally prove the algorithms correctness and analyze its performance. The validity of our results was confirmed by measurement results obtained in an FPGA prototype system.

ACKNOWLEDGEMENTS

We are grateful to Andreas Steininger and Gottfried Fuchs for many helpful discussions, and to Markus Ferriinger for providing us with the original DARTS FPGA implementation.

REFERENCES

- [1] A. Dielacher, M. Fuegger, and U. Schmid, “Brief announcement: How to speed-up fault-tolerant clock generation in vlsi systems-on-chip via pipelining,” in *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing (PODC’09)*. ACM Press, Aug. 2008, p. 423.
- [2] B. Charron-Bost, S. Dolev, J. Ebergen, and U. Schmid, “08371 summary – fault-tolerant distributed algorithms on vlsi chips,” in *Fault-Tolerant Distributed Algorithms on VLSI Chips*, ser. Dagstuhl Seminar Proceedings, B. Charron-Bost, S. Dolev, J. Ebergen, and U. Schmid, Eds., no. 08371. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009. [Online]. Available: <http://drops.dagstuhl.de/opus/volltexte/2009/1927>
- [3] U. Schmid, “Keynote: Distributed algorithms and VLSI,” in *Proceedings of the 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS’08)*, ser. Lecture Notes in Computer Science, vol. 5340. Detroit, USA: Springer Verlag, Nov. 2008, p. 3. [Online]. Available: <http://www.vmars.tuwien.ac.at/documents/extern/2467/sss08.pdf>
- [4] T. Moscibroda and O. Mutlu, “Distributed order scheduling and its application to multi-core dram controllers,” in *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC’08)*, 2008, pp. 365–374.
- [5] C. Ferri, T. Moreshet, R. I. Bahar, L. Benini, and M. Herlihy, “A hardware/software framework for supporting transactional memory in a mpsoc environment,” *SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 47–54, 2007.
- [6] M. Fuegger, U. Schmid, G. Fuchs, and G. Kempf, “Fault-Tolerant Distributed Clock Generation in VLSI Systems-on-Chip,” in *Proceedings of the Sixth European Dependable Computing Conference (EDCC-6)*. IEEE Computer Society Press, Oct. 2006, pp. 87–96.
- [7] S. Dolev and Y. Haviv, “Self-stabilizing microprocessors, analyzing and overcoming soft-errors,” *IEEE Transactions on Computers*, vol. 55, no. 4, pp. 385–399, Apr. 2006.

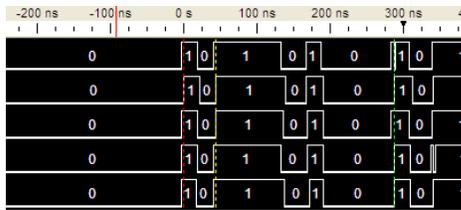


Figure 6. pDARTS with $X = 2$ (immediately after reset)

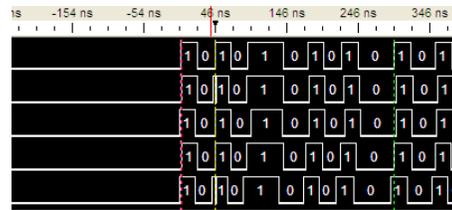


Figure 7. pDARTS with $X = 4$ (immediately after reset)

- [8] S. Dolev and N. Tzachar, “Brief announcement: Corruption resilient fountain codes,” in *DISC*, 2008, pp. 502–503.
- [9] R. Friedman, A. Mostefaoui, S. Rajsbaum, and M. Raynal, “Asynchronous agreement and its relation with error-correcting codes,” *IEEE Trans. Comput.*, vol. 56, no. 7, pp. 865–875, 2007.
- [10] N. Santoro, *Design and Analysis of Distributed Algorithms*, ser. Wiley Series on Parallel and Distributed Computing. John Wiley & Sons, 2007.
- [11] J. A. Garay, S. Kutten, and D. Peleg, “A sublinear time distributed algorithm for minimum-weight spanning trees,” *SIAM J. Comput.*, vol. 27, no. 1, pp. 302–316, 1998.
- [12] B. Awerbuch, I. Cidon, and S. Kutten, “Optimal maintenance of a spanning tree,” *J. ACM*, vol. 55, no. 4, pp. 1–45, 2008.
- [13] A. Bar-Noy, J. Naor, and M. Naor, “One-bit algorithms,” *Distributed Computing*, vol. 4, pp. 3–8, 1990.
- [14] T. K. Srikanth and S. Toueg, “Optimal clock synchronization,” *Journal of the ACM*, vol. 34, no. 3, pp. 626–645, Jul. 1987.
- [15] J. Widder and U. Schmid, “The Theta-Model: Achieving synchrony without clocks,” *Distributed Computing*, vol. 22, no. 1, pp. 29–47, Apr. 2009.
- [16] M. Ferringer, G. Fuchs, A. Steininger, and G. Kempf, “VLSI Implementation of a Fault-Tolerant Distributed Clock Generation,” *IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, pp. 563–571, Oct. 2006.
- [17] M. J. Fischer, N. A. Lynch, and M. S. Paterson, “Impossibility of distributed consensus with one faulty process,” *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.
- [18] A. Dielacher, M. Fuegger, and U. Schmid, “How to speed-up fault-tolerant clock generation in vlsi systems-on-chip via pipelining,” Technische Universität Wien, Institut für Technische Informatik, Research Report 15/2009, 2009, www.vmars.tuwien.ac.at/documents/extern/2571/techreport.pdf.
- [19] D. M. Chapiro, “Globally-Asynchronous Locally-Synchronous Systems,” Ph.D. dissertation, Stanford University, Oct. 1984.
- [20] T. Polzer, T. Handl, and A. Steininger, “A metastability-free multi-synchronous communication scheme for fault-tolerant soes,” Technische Universität Wien, Institut für Technische Informatik, Research Report 10/2009, 2009.
- [21] U. Schmid and A. Steininger, “Dezentrale Fehlertolerante Taktgenerierung in VLSI Chips,” Technische Universität Wien, Institut für Technische Informatik, Research Report 69/2004, 2004, (Österr. Patentanmeldung A 1223/2004).
- [22] G. L. Lann and U. Schmid, “How to implement a time-free perfect failure detector in partially synchronous systems,” Technische Universität Wien, Institut für Technische Informatik, Research Report 28/2005, 2005.
- [23] J. Widder, G. Le Lann, and U. Schmid, “Failure detection with booting in partially synchronous systems,” in *Proceedings of EDCC-5*, ser. LNCS, vol. 3463. Budapest, Hungary: Springer Verlag, Apr. 2005, pp. 20–37.
- [24] N. Lynch, *Distributed Algorithms*. San Francisco, USA: Morgan Kaufman Publishers, Inc., 1996.
- [25] I. E. Sutherland, “Micropipelines,” *Comm. of the ACM, Turing Award*, vol. 32, no. 6, pp. 720–738, Jun. 1989.
- [26] J. Grahl, T. Handl, and A. Steininger, “Exploring the usefulness of the gate-level stuck-at fault model for Muller C-elements,” in *Proceedings 20. TuZ’08*, Vienna, Austria, Feb. 2008, pp. 165–169.
- [27] I. Koren and Z. Koren, “Defect tolerance in VLSI circuits: Techniques and yield analysis,” *Proceedings of the IEEE*, vol. 86, no. 9, pp. 1819–1838, Sep 1998.
- [28] R. Baumann, “Soft errors in advanced computer systems,” *IEEE Design & Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005.
- [29] 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)*, 2001, pp. 773–776.
- [30] L. Lamport, “Buridan’s principle,” SRI International, Tech. Rep., 1984.
- [31] G. Fuchs, M. Fuegger, and A. Steininger, “On the threat of metastability in an asynchronous fault-tolerant clock generation scheme,” in *15th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC’09)*, May 2009.
- [32] D. Dolev, J. Y. Halpern, and H. R. Strong, “On the possibility and impossibility of achieving clock synchronization,” *JCSS*, vol. 32, pp. 230–250, 1986.
- [33] 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)*, Göteborg, Sweden, Jul. 2001, pp. 57–66.
- [34] G. Fuchs, M. Fuegger, U. Schmid, and A. Steininger, “Mapping a fault-tolerant distributed algorithm to systems on chip,” in *DSD*, Parma, Italy, September 2008, pp. 242–249.