Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2011, Article ID 936712, 23 pages
doi:10.1155/2011/936712

Research Article

VLSI Implementation of a Distributed Algorithm for Fault-Tolerant Clock Generation

Gottfried Fuchs and Andreas Steininger

Embedded Computing Systems Group (E182/2), Technische Universität Wien, Treitlstraße 3, 1040 Vienna, Austria

Correspondence should be addressed to Andreas Steininger, steininger@ecs.tuwien.ac.at

Received 15 June 2011; Accepted 12 August 2011

Academic Editor: Jae-Yoon Sim

We present a novel approach for the on-chip generation of a fault-tolerant clock. Our method is based on the hardware implementation of a tick synchronization algorithm from the distributed systems community. We discuss the selection of an appropriate algorithm, present the refinement steps necessary to facilitate its efficient mapping to hardware, and elaborate on the key challenges we had to overcome in our actual ASIC implementation. Our measurement results confirm that the approach is indeed capable of creating a globally synchronized clock in a distributed fashion that is tolerant to a (configurable) number of arbitrary faults. This property facilitates eliminating the clock as a single point of failure. Our solution is based on purely asynchronous design, obviating the need for crystal oscillators. It is capable of adapting to parameter variations as well as changes in temperature and power supply–properties that are considered highly desirable for future technology nodes.

1. Introduction

Throughout the last decades progress in VLSI technology has constantly fueled an incredible advancement in complexity, speed, functionality, and power efficiency of digital circuits [1]. This trend has always created new opportunities, but at the same time has been accompanied by various challenges for the design of these circuits [2]. Contemporary chip design seems to be dominated by the following issues.

(i) *Fault Tolerance*. It is commonly agreed that technology nodes smaller than 65 nm tend to become increasingly vulnerable to single-event upsets, due to their small critical charges and the low voltage swing [3–5]. As a consequence the need for fault tolerance emerges, even for non-safety-critical applications.

(ii) *Power Efficiency*. With a growing number of transistors per unit area, the power density is increasing, even in spite of technological progress. This leads to problems with power distribution and with heat dissipation.

(iii) *Variation Tolerance*. The fabrication tolerances of new technology nodes lead to uncertainties in the timing behavior, power consumption, and so forth, where traditional corner-case design is too pessimistic [6]. Therefore design techniques are sought that are capable of sustaining reliable operation even under these variations.

In the light of these substantial challenges even one of the foundations of digital design is being questioned, namely, the globally synchronous paradigm. While the abstraction of the chip being a perfect isochronous region facilitates an efficient design, retaining a reasonable synchrony all over a large and complex chip with a sub-nanosecond precision has become extremely cumbersome. To minimize the skew within the clock network, not only sophisticated geometries are applied, but in addition large numbers of clock buffers and deskewing units have to be placed at carefully chosen positions [7–9]. As a result an appreciable share of the power budget goes into the clock distribution network [10, 11]. This stands in contrast with the above stated requirement for power efficiency. Recently, parallelism is being introduced to increase processing power while maintaining clock speed. This trend is accompanied by new communication schemes, so called networks on chip. As a recent example, the Godson-3B [12, 13] comprises two distinct groups of four tightly coupled cores per group.

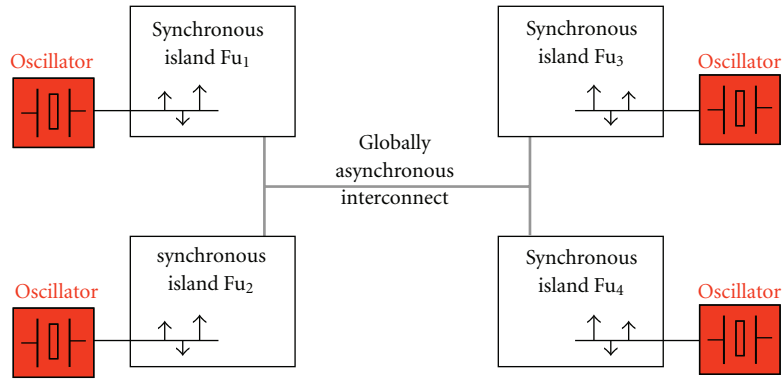


Figure 1: Globally asynchronous locally synchronous (GALS) architecture.

Item 3 on the above list, namely the increasing fabrication tolerances, unfortunately forms another obstacle for maintaining an isochronous region all over the chip: The synchronous design paradigm rests upon the intimate knowledge of the circuit timing that becomes blurred for newer technologies.

And finally the globally synchronous clocking approach proves to be problematic with respect to fault tolerance as well. Although synchrony is an important foundation for many fault-tolerance schemes (such as TMR or duplication and comparison), the single, central clock source forms a single point of failure even in such a replicated architecture. This issue has long been neglected, as the clock network is considered robust due to its strong drivers and its relatively high capacitance. Recently, however, concerns have been raised about the vulnerability of the clock nets, and specifically clock repeaters, as well [14].

In addition to all these challenges, however, newer technologies also introduce new possibilities as well. The architectures found in systems on chip have very much in common with traditional distributed architectures. In the latter a globally synchronous clock source has hardly ever been employed—their components are rather loosely coupled, employing several local clock sources that are then synchronized on a higher level of abstraction by distributed algorithms, if desired. In this paper we will review options for generating a fault-tolerant clocking scheme that is feasible for modern technologies and architectures, and we will present a novel scheme for fault-tolerant clock generation that is based on a distributed algorithm and thus exploits the typical architecture found in systems on chip (SoC). In addition to its superior fault tolerance the proposed scheme is extremely robust against process variations.

2. Related Work

As motivated above traditional, globally synchronous design may not be able to meet all upcoming challenges to future computer architectures. Subsequently, several promising alternatives will be surveyed that have been proposed in the literature.

2.1. Globally Asynchronous, Locally Synchronous. The Globally Asynchronous Locally Synchronous (GALS) approach [15] is based on the generic architecture depicted in Figure 1. Small (local) synchronous islands implement functions (subtasks) of the whole system. Each local island's function is executed using the traditional synchronous design style, whereas global interaction follows an asynchronous communication style. Each island is provided with its own oscillator as clock source for the locally synchronous computations. Compared to the high effort for global clocking of a purely synchronous system, in local synchronous islands skew optimization of the clock signal is much easier to attain. Although GALS simplifies the clock distribution to some extent, some other issues are still left. The need for a dedicated oscillator for each synchronous island adds additional components to the system, which clearly decreases reliability. The often used quartz oscillators are sensitive to, for example, vibration, temperature, shock, and so forth, while on-chip RC oscillators are known for their strong dependence on operating conditions like temperature and supply voltage, which leads to frequency changes in the range of 10 to 30%.

Beyond that, if compared to a synchronous system, the GALS concept has two major fundamental disadvantages. Firstly, a GALS design does not implicitly provide the convenient systemwide notion of time which most hardware designers are used to and design tools are made for. All clock sources are free running—the local clocks may drift arbitrarily apart from each other. Communication leaving a local island's clock domain introduces the need for synchronization. The fact that the interface between globally asynchronous communication and locally synchronous data processing has to incorporate some sort of synchronizer circuits poses the second, probably the most severe, disadvantage of GALS. Unfortunately, synchronizing clock domains with arbitrary, possibly changing, relation to each other, cannot be solved in a safe way. Metastability issues might even upset the synchronizer circuits [16] and can only be made more unlikely by adding further synchronizer stages. Taking parameter variations and clock jitter into

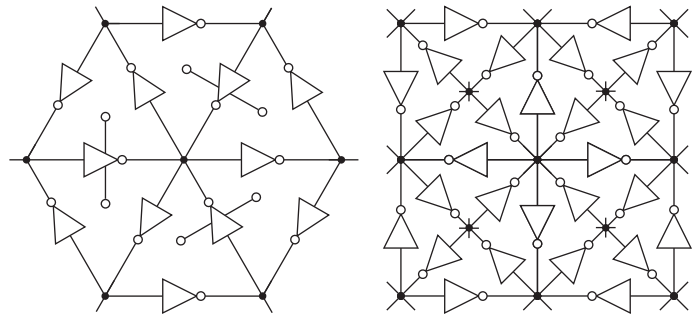


Figure 2: Interconnected ring oscillator architectures.

account synchronizers have to be designed very conservatively, thus introducing significant performance penalties into the asynchronous/synchronous interfaces.

Recent GALS implementations incorporate stoppable (plausible) and/or stretchable clocks [17, 18] to reduce performance loss at the clock domain interfaces. This, however, comes at the price of a reduction of clock accurracy and stability.

2.2. Interconnected Rings and Oscillators. This concept proposed by Maza and Aranda in [19, 20] presents an alternative approach for generating and distributing GHz clocks. The design relies on the self-oscillation property when interconnecting an odd number of inverters in a ring topology (shown in Figure 2) and achieves high clock frequencies due to its simplicity. Inverter and buffer placement of the proposed architecture determines wiring costs (in terms of wire length), speed, and skew of the generated clocks. The design especially fits as on-chip clocking scheme for the previously introduced GALS systems. It can be seen as a refinement of the GALS RC-oscillator clocking. Due to the fact that all inverters of the clock generation scheme are interconnected directly (locally) or indirectly (globally, through some additional inverter stages) with each other, the local islands of a GALS system cannot arbitrarily desynchronize (at least in the fault-free case). This property severely eases synchronization within the GALS design since the synchronizers can take advantage of the fact that the local clocks are not entirely unrelated.

2.3. Distributed Clock Generator. The scheme of a distributed clock generator (DCG) introduced by Fairbanks and Moore [21, 22] represents a special form of asynchronous FIFO implementation for the purpose of on-chip generation and distribution of a synchronized clock. Similarly to the approach by Maza and Aranda, interconnected clock generation hardware is distributed in a grid all over the chip, but the locally generated clocks are generated at approximately the same instant having only small skew. Every DCG instance is interconnected with its four neighbors, and half of the DCG units are initialized with a clock token. Due to the asynchronous FIFO implementation of each DCG the so-called Charlie effect [22] ensures that clock tokens are passed over to neighboring nodes in a synchronous way, generating a chip wide synchronized clock signal (the Charlie effect describes the force that slows down a subsequent token within a FIFO if it is closing in on a previous one).

2.4. Purely Asynchronous Design. Asynchronous design styles [23] are considered a viable alternative for synchronous design in the future, specifically for application fields like low power [24] or high performance [25, 26]. With asynchronous design the burden of clock distribution can be entirely eliminated and the clock tree be substituted by far less timing critical local handshake signals. Parameter variations are much less problematic in the context of, for example, quasi delay-insensitive circuits [27] since, due to the indication principle, only performance but not the correct function is influenced by variations. Furthermore, the inherent robustness of asynchronous design styles allows to address the issue of increased failure rates in future VLSI technology [28, 29] to some extent. On the downside, the variety of existing asynchronous design styles and delay models distracts not only designers who are not expert in the field, but also EDA companies whose design and verification tools are crucial enablers for a general acceptance of the asynchronous design paradigm. Nonnegligible area overhead, higher design complexity, and the intricate circuit testing issues add to these problems. Even though a good robustness is inherent to asynchronous designs, the "wait for all" paradigm implied by the indication principle prevents established system-level fault-tolerance techniques, like triple modular redundancy (TMR), from being directly applied to asynchronous systems [30].

2.5. Discussion. In the presented approaches the incorporation of fault tolerance as well as the robustness required for coping with unexpected faults as well as parameter variations is mostly lacking. GALS in general has issues with interfacing multiple uncorrelated clock domains, and its lack of a global time severely complicates the design process (which is also the case for purely asynchronous approaches). The interconnected rings and oscillators as well as the distributed clock generator approach are not able to cope with failures. To be able to tolerate arbitrary failures in a clock synchronization process, theory shows that almost

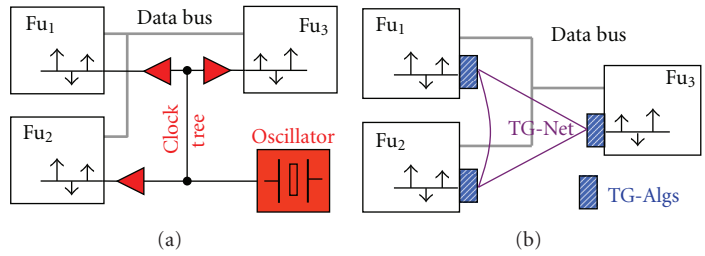


Figure 3: Replacing synchronous clocking by fault-tolerant distributed tick generation.

fully connected networks are needed [31] which is clearly not fulfilled by those two approaches. Therefore, a transient fault might lead to major clock deviation, overclocking phenomena or could even stop the whole clock generation process.

The work described in this paper focuses on the development of a robust clocking scheme for future dependable systems. Especially in safety- and mission-critical environments like in the automotive and aerospace domain robustness against arbitrary faults is of utmost importance. Similar to GALS, our approach provides strong local synchrony. In contrast to GALS, however, a fault-tolerant time base is maintained on the global level as well (albeit with slightly relaxed synchrony assumptions).

3. The DARTS Concept

Figure 3 illustrates the key principle of our approach: we replace the central clock source (crystal oscillator) by a set of tick generation units. Each of these units implements an instance of the same distributed algorithm in hardware (therefore we call them *TG-Algs* further on). This algorithm is based on communication between the individual TG-Algs, through which the TG-Algs mutually stimulate each other, thus creating an oscillation that can be viewed as a globally synchronized clock. Through the choice of an appropriate algorithm and by means of a careful implementation, the TG-Algs' local perceptions of this global clock remain within a bounded precision, even if the communication network, called *TG-Net* in Figure 3, introduces considerable delays and skew. Each of the TG-Algs is attached to one or more functional units of the SoC for which it provides a local clock that is in synchrony with all other local clocks generated by the other TG-Algs for their respective functional units (Fu_i). Based on these local clocks the functional units can internally be operated according to the traditional synchronous design paradigm, which is desirable and unproblematic as long as their local extent is limited.

From a high-level perspective distributed algorithms suitable for tolerating multiple Byzantine faults can be incorporated to get a robust clocking scheme. If we use such an algorithm for our TG-Algs, our clock generation will stay operational even if some of the TG-Algs and/or links of the TG-Net should fail arbitrarily. Clearly, the functional units connected to the failed TG-Algs will no more be supplied with a proper clock, but this can be compensated by their appropriate replication along with connection of the replica to different TG-Algs. Note that this solves a notorious problem in classical fault-tolerant VLSI systems: fault tolerance is considerably easier to implement under the assumption of global synchrony, while the establishment of global synchrony by means of a central clock introduces a single point of failure. The key advantage of our approach is to provide a globally synchronized clock that is at the same time fault tolerant. On this foundation it is straightforward to build a fault-tolerant architecture on the level of functional modules. Furthermore, the global synchrony allows metastability-free communication between the functional units without the need for synchronizers [32]. Another advantage of our approach is its insensitivity to delays in the communication links as well as to the TG-Alg's propagation delays; as we will show later, even considerable tolerances, drift and jitter, can be accommodated. This relieves the designer from using strong drivers, which in turn increases power efficiency.

We have designed, formally proven, simulated, implemented, and evaluated the proposed concept in the course of the research project *Distributed Algorithms for Robust Tick-Synchronization (DARTS)*. While an in-depth analysis of the formal aspects of the DARTS approach can be found in [33], this paper will be more concerned with the implementation-related issues of DARTS. In particular, throughout the remainder of this section we will investigate the foundations for our concept, namely, the selection of a suitable algorithm and the constraints that have to be considered when implementing that abstract algorithm as VLSI chip design.

3.1. Finding a Suitable Distributed Algorithm. Distributed computing research provides the required algorithms for fault-tolerant generation of synchronized clock ticks. The class of distributed algorithms considered in the DARTS approach is based on message passing, with a set of particular properties to meet the requirements for tick generation. In short these characteristics of tick-generation algorithms are as follows.

(i) The algorithm consists of a set of rules which are evaluated whenever a message arrives at a node. These rules conditionally update the respective node's local memory and trigger the transmission of messages to other nodes.

```
1: variables
2:    k: integer :=0
3: end variables
4: if C^{k-1}(t) = kP then //ready to start C^k
5:    → broadcast(TICK(k))
6: end if
7: if accepted the message(TICK(k)) then //according to a selection/voting function
8:    → C^k(t) := kP + α
9: end if
```

Algorithm 1: Nonauthenticated algorithm for clock synchronization at node p [36].

(ii) To implement a tick generation approach, the class of distributed algorithms is restricted to those that send only messages containing ascending natural numbers, that is, it is demanded that every node p sends messages $\langle 0\rangle, \langle 1\rangle, \langle 2\rangle, \ldots$ in the given order during its executions. When mapping tick generation to hardware the natural numbers of messages TICK(k) mod 2 can be seen as discrete *up* and *down* transitions of a hardware clock.

(iii) To achieve synchronization among all nonfaulty nodes of a distributed system, a tick generation algorithm has to solve the synchronization problem following Lamport's definition [34, 35] if synchronization precision π and accuracy shall hold.

(iv) Furthermore, the algorithm is called fault-tolerant if it maintains the conditions described above even in the presence of faults.

An algorithm presented by Srikanth and Toueg in [36] fulfils all these criteria. It comprises two parts. According to Algorithm 1, the so-called nonauthenticated clock synchronization part, node p broadcasts message TICK(k) as soon as its clock counter $C^{k-1}(t)$ reaches a threshold value indicating that the next tick has to be issued. The internal clock counter is directly driven by a local oscillator. The round-based threshold value is given by kP, where P denotes the predefined resynchronization interval.

When reaching the threshold count node p is ready to change from round $k-1$ to round k. However, in order to keep the system synchronized, this transition must be coordinated with the other nodes. This is accomplished by resynchronizing the clock counter $C^k(t)$ at the instant an "accepted TICK(k)" message is received from another node (actually, $C^k(t)$ is adjusted to $kP+\alpha$, where α denotes a constant ensuring that the clock always steps forward in time, see Figure 4).

The function responsible for generating these "accept TICK(k)" messages forms the second part of the algorithm and is shown in Algorithm 2. It comprises three parallel rules for message processing. In response to the reception of particular TICK(k) messages from at least $f+1$ distinct nodes, either via *init* or *echo* messages, each node relays an *echo* TICK(k) message to all other nodes (Relay rules). The actual generation of an "acceptance event" for advancing the clock, however, requires the reception of at least $2f+1$ distinct

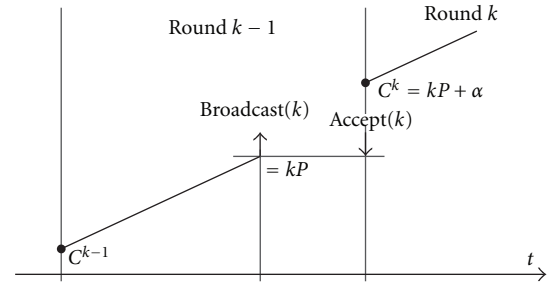


Figure 4: Nonauthenticated broadcast execution at node p.

echo TICK(k) messages (Accept Rule). It has been shown by Srikanth and Toueg that in a system of $n \geq 3f+1$ nodes Algorithm 1 in cooperation with the consistent broadcast primitive of Algorithm 2 solves the clock synchronization problem, even in the presence of up to f Byzantine faulty nodes if the conditions hold that

(i) the local clocks' maximum drift rate is known and bounded by ρ,

(ii) message end-to-end delays are within a certain known bound of $[d, d+\varepsilon]$,

(iii) two specific timing assumptions are ensured by properly chosen values for P and α.

Let us reconsider the initial motivation for taking a closer look at tick generation algorithms, namely, to get synchronized clocks without the need for local clock sources. The nonauthenticated algorithm for clock synchronization, presented above in Algorithm 1, still requires a local clock source at each node to supply the local clock counter. Fortunately, some modifications of Algorithm 1 yield a solution which no longer requires a local counter and also removes the distinction of *init* and *echo* events, which largely eases message handling. This algorithm is shown below (Algorithm 3).

It is derived from the algorithm originally proposed by Widder and Schmid [37], however, with the following important change: in order to facilitate an implementation in hardware (see Section 3.2) it no longer relies on infinite TICK(k) numbers. Under the additional constraint that every TICK(k) message is only sent [**once**] this simplification can be accomplished without spoiling the analysis of [37].

```
1: variables
2:    k: integer :=0
3: end variables
4: for each correct process do
5:    if received(init,TICK(k)) from at least f + 1 distinct nodes then //Init Relay Rule
6:       → send(echo,TICK(k)) to all
7:    end if
8:    if received(echo,TICK(k)) from at least f + 1 distinct nodes then //Echo Relay
9:       → send(echo,TICK(k)) to all
10:   end if
11:  if received(echo,TICK(k)) from at least 2f + 1 distinct nodes then //Accept Rule
12:     → accept(TICK(k))
13:  end if
14: end for
```

Algorithm 2: Acceptance function selecting valid clock ticks [36].

```
1: variables
2:    k: integer :=0
3: end variables
4: initially send TICK(0) to all [once]
5: if received TICK(ℓ) from at least f + 1 rem. processes with ℓ ≥ k then //Relay Rule
6:    send TICK(k),..., TICK(ℓ) to all [once]
      K := ℓ
7: end if
8: if received TICK(k) from at least 2f + 1 remote processes then //Increment Rule
9:    send TICK(k + 1) to all [once]
      K := k + 1
10: end if
```

Algorithm 3: Byzantine-tolerant tick generation suitable for bounded tick numbers.

With this algorithm the previously used assumption on message delays $[d, d+\varepsilon]$ can be weakened to the one that for any two messages in transit m_1, m_2 it has to hold that

$$\frac{\delta(m_1)}{\delta(m_2)} \leq \Theta \tag{1}$$

with $\delta(m_1)$ and $\delta(m_2)$ being the respective message delays of m_1 and m_2, and Θ being constant. The analyses in [37] show that despite the presented substantial simplifications, Algorithm 3 still solves the clock synchronization problem—in this particular case, this is maintaining precision π as well as accuracy even in the presence of Byzantine faults. Algorithm 3 processes like this: the "Relay Rule" of a correct node fires as soon as TICK(ℓ) messages from at least $f + 1$ distinct nodes have been received—given that f is the maximum number of faults the system is supposed to tolerate, this ensures that at least one of these TICK(ℓ) messages has been issued by a correct node. Notice that in the case of triggering the "Relay Rule" the node does not immediately set its local clock k to ℓ since this would lead to skipping some values of k if the respective node is lagging more than one tick behind. The strategy followed in Algorithm 3 explicitly ensures that all messages TICK(k),..., TICK(ℓ) are issued when catching up with faster nodes, resulting in a continuous progression of the clock without potentially troublesome leaping effects. Especially when recalling the targeted application of clocking synchronous circuits, skipped clock ticks might result in inconsistent state progression over different functional modules.

As a result we now have a distributed algorithm available that is able to generate an ascending sequence of clock ticks k in a fault-tolerant manner without relying on a local clock source.

3.2. Hardware Implementation Challenges. So far our approach has been to use a distributed tick generation algorithm for generating an ascending sequence of tick numbers, and the mapping to rising and falling edges in the hardware implementation simply implies a mod 2 operation. The above algorithm provides all required features; however, substantial implementation-related problems originate in the fact that this algorithm (like virtually all other distributed algorithms) has been designed on a very high level of abstraction, at best with a software implementation in mind.

In general, a tick generation algorithm operates on unbounded natural numbers, whereas a hardware clock signal simply toggles between the two logic values *high* and *low* (Figure 5). For our ultimate purpose of clocking our

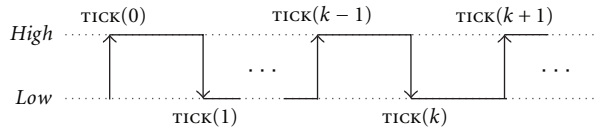


FIGURE 5: Hardware clock signal versus tick numbers.

functional units we do not need the history information contained in the individual tick numbers, and we cannot afford to convey it. With clock frequencies ranging into hundreds of MHz or even some GHz the value of k rapidly reaches gigantic dimensions, and in fact its unbounded nature prohibits any concrete implementation. Moreover, each value has to be repeatedly transferred at this high frequency as part of the tick generation process, which obviously causes an excessive data rate.

At the same time we cannot completely get rid of k, since the abstract operational principle of tick generation algorithms relies on counting up this integer tick number. In order to facilitate a hardware implementation, we have to change the algorithm such that it can accommodate bounded values for the TICK(k) numbers and the resulting wrap-around effects in their numerical representation, that is, after sending the largest value of k in the chosen integer representation, the smallest one, for example, TICK(*0*) follows. From the hardware point of view the bound on k should be as low as possible. In practice, however, this minimization of k is limited by the boundary conditions that

> Alg-R1: it must be ensured that no TICK(k) messages of different wrap-around phases can interfere with each other, and a bound on the maximum offset of any two clocks holds;
>
> Alg-R2: the two parallel rules (Increment- and Relay-Rule) executed on a node p never generate and sequentially transmit the same TICK(k) message.

To accommodate for bounded values of k, the algorithm presented by Widder and Schmid [37] was augmented by the requirement that every TICK(k) message is only sent [**once**] regardless of the fact that multiple rules might be eligible to generate this particular TICK(k) message. This change is already reflected in Algorithm 3.

Based on the refinements of our original algorithm presented in this chapter, and considering the boundary conditions we have identified, we can concentrate our efforts on finding a suitable mapping of algorithmic statements to hardware building blocks in the next chapter.

4. Hardware Implemented Fault-Tolerant Tick-Generation

4.1. Hardware-Related Requirements. Even with the modified tick synchronization algorithm by Widder and Schmid (Algorithm 3) there are still several difficulties when attempting to map the software-based (high-level) tick generation algorithm to the restrictions of hardware design. Most of them are not due to algorithm-related requirements, but rather originate from the need to find a fast and area-efficient projection of the algorithm to hardware, since these issues are by no means considered in the high-level description of the algorithm so far. In the following we will give a list of these challenges.

> HW-R1: *Tick Generation Network*. Integer TICK(k) messages have to be conveyed in a way to keep the clock network as simple as possible without compromising clock speed. Therefore, strategies having more than a single wire per clock signal are assumed too costly.
>
> HW-R2: *Tick Messages*: Simple TICK(k) messages have to be used to enable highest possible speed, while still operating on a single rail per clock signal—clock transitions (*up/down*) on a single signal rail, as depicted in Figure 5, seem to be the only viable implementation option.
>
> HW-R3: *Unique Sender Identification*: The algorithm is based on the assumption that the receiver of a TICK(k) message can uniquely identify the respective sender. In the light of HW-R2, appending a sender ID is not feasible, hence we need a point-to-point connection between any two nodes, that is, a fully meshed network. This in turn confirms the claim for a lightweight interconnection posed in HW-R1.
>
> HW-R4: *Asynchronous Design*: Since the TG-Algs' task is to generate a clock, they do not have a clock available for their own operation in the first place. (In principle, the provision of a local clock to each TG-Alg would be possible, but it would counteract the original intention of the approach, namely generating a clock, and it would suffer metastability problems at the clock domain boundaries that would inevitably emerge then.) As a consequence their implementation needs to follow an asynchronous design paradigm. From the available approaches the (quasi) delay insensitive one is most attractive, since it does not make assumptions on the individual path delays, thus increasing the desired robustness.
>
> HW-R5: *Atomicity of Actions*: For all distributed computing models that the authors are aware of, atomic computing steps at the level of a single node are assumed. However, this abstraction cannot be adopted when implementing an algorithm directly in (asynchronous) hardware, where computations are performed by numerous concurrently operating digital logic gates. The most challenging part in our case is given by the parallel processing of the two algorithmic rules ("Relay Rule" and "Increment Rule" of Algorithm 3) in conjunction with concurrently arriving TICK() messages. To handle requirement Alg-R1, explicit synchronization of local computations is needed.
>
> HW-R6: *Fast Operation*: The theoretical analysis of the algorithm [38] confirms the intuition that the

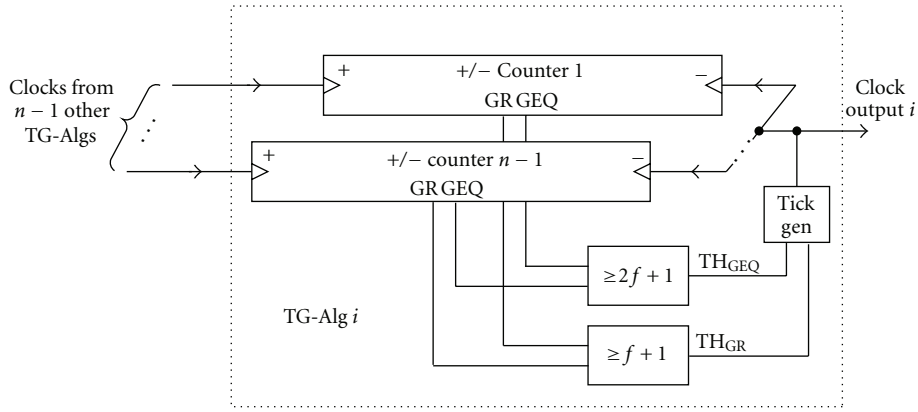


Figure 6: Tick generation architecture handling relative tick numbers.

oscillation frequency delivered by the DARTS scheme as well as the attainable precision is determined essentially by the round-trip times of the transitions, that is, the time from the generation of a particular transition until the generation of the next one as a result of its reception by all other nodes. Consequently, in order to obtain fast operation, the hardware implementation shall minimize the number of gate delays in these paths.

The most demanding design requirement is certainly HW-R4, the need for an entirely asynchronous hardware implementation: this restriction completely rules out the employment of the well-established synchronous design methods. Systematic asynchronous design styles (e.g., [39]) follow a handshake-based flow control to ensure that no old data interferes with new one—this provides *interlocking* between subsequent data waves. In this context it is important that a transition at a gate input causes an effect at the gate output, before the next input transition is allowed to occur. This so-called *indication principle* is crucial for handling concurrency (without having to introduce timing assumptions), and it can be maintained by handshaking, as long as the function of the gate is such that every input transition actually causes an output transition, that is, no input is being masked or disregarded.

However, in the context of fault-tolerant design a fundamental problem arises with this indication principle: if a module is waiting until *all* REQs have properly arrived before issuing an ACK and proceeding with its operation (as it is implied by the handshake procedure), this would allow a single faulty unit to inhibit any further processing. Hence, a strategy has to be followed where processing is halted only until an algorithm-dependent *threshold* of REQs has been reached. While such an approach now enables the incorporation of fault tolerance, it necessarily breaks the implementation's REQ/ACK feedback loops for the slowest paths, thus inhibiting their indication. Without additional measures or constraints such open loops tend to run out of sync, endangering the correct operation of the whole system. In order to obtain a fault-tolerant asynchronous design the traditional closed REQ/ACK control loops have to be augmented by explicit timing constraints, this way supporting an interlocking scheme for consecutive data waves for all, even the slowest paths. We will elaborate on these timing constraints later on. For the moment we will just summarize this insight in a further requirement.

HW-R7: *Timing Requirements*: In the absence of a global clock we are forced to apply the principles of asynchronous design, according to which all activities must be involved in a REQ/ACK handshake cycle. Due to requirement HW-R1 we do not want to extend this handshaking principle to the TG-Net. Furthermore, fault tolerance techniques like a threshold function essentially contradict the "wait-for all" paradigm implied by the handshaking. For these reasons handshaking shall not be employed in a completely consequent fashion, and appropriate timing conditions shall be elaborated to constrain the implementation in such a way that proper operation is still ensured where the handshake loops are broken.

4.2. Block Diagram of the Implementation. Figure 6 shows the basic architecture of a single TG-Alg's hardware design resulting from the above described specifications. The most notable peculiarity of this design is the dissemination strategy for TICK(k) messages. In accordance with HW-R2 no explicit tick numbers are transmitted over the TG-Net. Anonymous *up* and *down* signal transitions (zero-bit messages) are used instead of conveying integer values. From an abstract point of view this means that the sender just conveys "differential" information in the shape of a plain transition, while each receiver is equipped with a counter to integrate this information into the actual tick number k. In this way the message size can be reduced to the absolute minimum, while, however, every node now requires one separate counter per incoming link (i.e., $n-1$), further called "remote counter" (RC), in addition to the single one for maintaining its local count (LC).

A closer look at Algorithm 3 reveals that only a *relative* comparison between LC and RC is carried out, while their absolute value is not relevant: the relay rule checks whether the remote count l is greater than the local count k, while

the increment rule checks whether RC $\geq$ LC. Therefore, as shown in Figure 6, we can employ up/down counters, further referred to as +/− Counters to just maintain the *difference* between LC and RC rather than their absolute values. Note that this difference never becomes larger than the precision—this is guaranteed by the function of Algorithm 3. With this knowledge we can safely minimize the size of the difference counters without having to consider a potential wrap-around further on.

In accordance with Algorithm 3, in the next stage we join the results of the "greater" (GR) comparisons and those of the "greater or equal" (GEQ) comparisons from all inputs of a node and check whether the number of positive comparison results reaches the threshold of $f + 1$ implied by the relay rule or $2f + 1$ from the increment rule, respectively. This is done by the units called "threshold gates," in Figure 6 marked by their respective threshold values. The task of an m-of-n-threshold gate is simply to activate its output as soon as m or more of its n inputs are at logic HI.

At this point it is interesting to note that, although implied by Algorithm 3, we do not consider a self-reception in our hardware implementation, that is, a node only receives the tick messages sent by *all other* nodes, not the one produced by itself. The reason is that in practice the self-reception path is very likely to be much faster than all the other message delays. Now recall from Section 3.1 that the attainable precision largely depends on a constant Θ which is derived from the ratio of the fastest and slowest feedback delays within the tick generation scheme. Hence the imbalance caused by an extraordinarily fast self-reception path would unnecessarily increase Θ, thus degrading the attainable precision. As a consequence of omitting the self-reception path, the presented tick generation system has to comprise at least $3f + 2$ TG-Algs instead of the usually applied (lower bound of) at least $3f + 1$ nodes to attain the targeted degree of Byzantine fault tolerance.

The activation of a threshold gate's output corresponds to the firing of a rule in Algorithm 3, and as there are two concurrent rules in the algorithm, we have two threshold gates operating in parallel, with the appropriate thresholds of $f + 1$ and $2f + 1$. Finally, a *tick generation unit* ("Tick Gen" in Figure 6) takes care of properly combining these outputs into a tick that can be conveyed over the TG-Net. This is the place where the "once" statement from Algorithm 3 and requirement HW-R5 become important: care must be taken to issue only one tick per k value in spite of the concurrent operation of the two threshold gates.

The block diagram developed so far has been sufficiently accurate to allow for a thorough formal analysis of the TG-Alg design [33], yielding several implementation constraints. Nevertheless, from a hardware designer's point of view the abstraction of the TG-Alg design has to undergo further steps of detailing to enable a successful mapping to an ASIC manufacturing process. This will be the subject of the next section. (Our previous work [40] treated an FPGA prototype, whereas the main focus of this paper is on the presentation of the ASIC implementation and the evaluation of a DARTS cluster based on the manufactured ASICs.)

5. DARTS Implementation

In this section the TG-Alg's hardware implementation is presented in more detail. To this end Figure 7 presents a more accurate architecture of a single TG-Alg.

In the top part of the figure the +/− Counter module is shown, decomposed into a "Local Pipeline," a "Remote Pipeline," a "Diff Module," and a "Pipe Compare Signal Generator." Note that this counter module just corresponds to one remote input; overall $n - 1$ of these modules are required per node to handle all incoming links from the TG-Net. This fact is illustrated by the further counter modules shown in the bottom left part of the figure. The modules termed "Threshold Modules," on the bottom right in the figure, comprise the threshold gates and the "Tick Generation" unit.

To understand the proposed structure of the $\pm$ Counter recall from Section 4 that message transmission is differential, that is, we use *transitions* to convey the TICK(k) messages. As we do not know (and actually do not want to assume) any phase relationship between the local ticks and the remote ticks, TICK($\uparrow$) and TICK($\downarrow$) transitions may occur arbitrarily close to each other, hence introducing the potential for metastability. Instead of building a flip-flop-based counter, as one would usually do in synchronous logic, we decided to go for a consequent implementation in *transition logic*. In transition logic the expressiveness is limited to the causal ordering of events in a basically time-free system. However, to retain its delay insensitiveness the class of allowed circuit elements is fairly restricted. Permitted elementary units are for instance Muller C-Elements, inverters, XOR gates, and a few rather complicated and quite exotic building blocks like the toggle unit [41–43]. Even simple logic operations have to be treated in a different way in the scope of transition signaling. The behavior of a conventional OR gate that is generating an output as soon as the first (rising) input event has occurred would, for example, destroy the causality relation of the late input with the issued output transition. Its use is therefore not permitted in transition logic; the same is true for the AND gate.

Given this limited set of available functions, our approach to implement the +/− Counter is as follows: we provide a buffer for incoming transitions, both from the local and from the remote side. An implementation based on the well-known transition signaling elastic pipeline approach made famous by Sutherland [43] can be employed here. These are the modules termed "Remote Pipeline" and "Local Pipeline" in Figure 7. The "Diff Module" is connected to the outputs of both these pipelines and removes pairs of "matching" transitions (i.e., such with matching (virtual) k values) from both sides. In the static case one pipeline is always empty while the other one contains the difference of ticks. The nonempty pipeline is the one that has experienced a higher number of ticks (indicating the sign of the difference), while the number of pipeline entries equals the absolute value of the difference. As a result, the two elastic pipelines together with the Diff Module form the desired +/− Counter for the differential ticks. Note that, while due to requirement HW-R1 no acknowledge is given

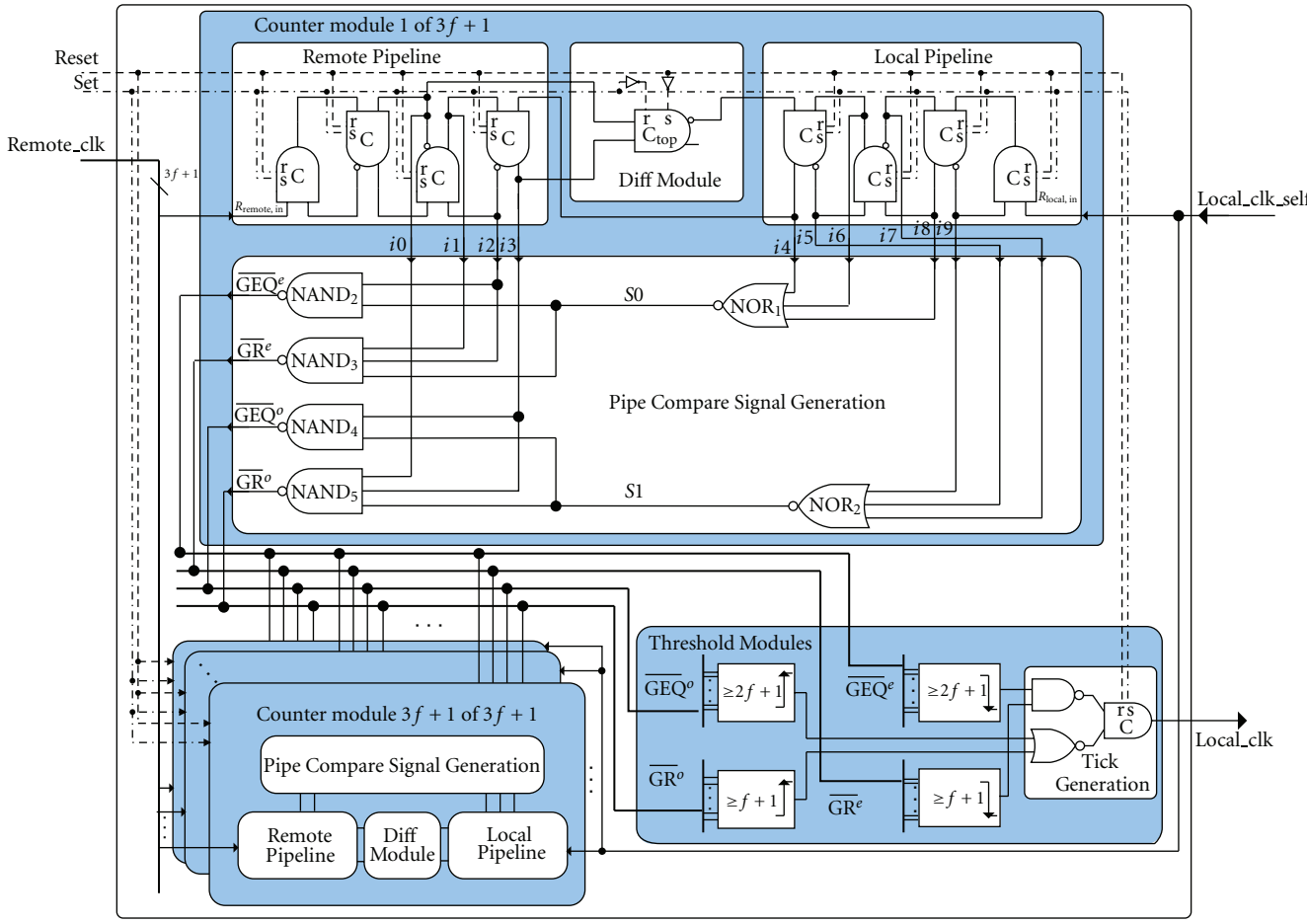


Figure 7: TG-Alg ASIC design architecture.

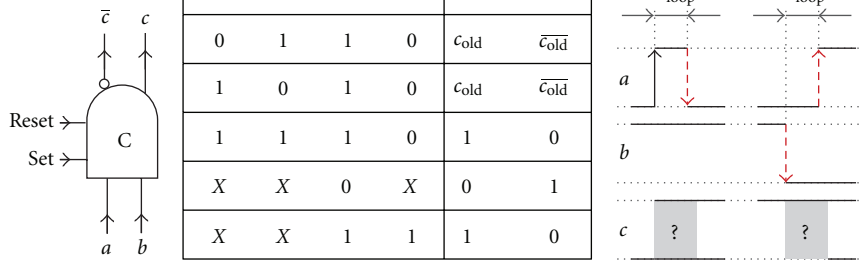
for incoming ticks, the pipelines internally operate fully via handshake, and also the interface to the Diff Module employs handshaking, thus yielding very robust operation. Details on the implementation of these modules will be presented in Sections 5.1 and 5.2.

There it will also become clear that the function of this counter is fully symmetric for TICK(↑) and TICK(↓) transitions. This allows us to use no-return-to-zero (NRZ) encoding for our tick messages, that is, each transition represents a tick, no matter whether it is a rising or a falling one, which perfectly supports our desire for efficient communication (cf. HW-R2). Therefore we are willing to accept the higher implementation efforts usually associated with NRZ coding. By exploiting this symmetry we can furthermore ensure that both half periods of our DARTS clock (HI and LO) have undergone the same treatment through our algorithm, and hence the duty cycle will be very close to 50%. Note that, although in essence the ticks need not be distinguishable, we can separate "odd" and "even" ticks by their polarity (rising corresponds to odd, falling to even). This will become important later on in the context of interlocking.

The "Pipe Compare Signal Generation (PCSG)" module performs the "greater" and "greater or equal" comparisons of Algorithm 3 by inspecting the fill levels of the remote and the local pipelines. In essence this is a conversion from transition signaling to state logic. This move to state logic is inevitable, since a counter value essentially represents a state and not an event, and so does the comparison result between two counter values. Within the state logic implementation we operate without handshaking and rather rely on timing assumptions. This is inevitable for the subsequent stage, the threshold gate, anyway, since, as already outlined, we cannot operate an m-out-of-n threshold in a fully handshake-based manner. In the conversion from transition logic to state logic care must be taken not to incidentally interpret transient states; this issue will be treated in more detail below.

Notice that the PCSG provides two outputs for the "greater" comparison, namely, GR^o and GR^e, and another two for the "greater or equal" comparison, namely, GEQ^o and GEQ^e. This is because the PCSG's operation is *not symmetric* for rising and falling transitions. In order to preserve the distinction between odd and even ticks we therefore generate separate output signals for these.

5.1. Queueing Ticks. As mentioned before elastic pipelines can be viewed as FIFO buffers for transitions. The better part of an elastic pipeline consists of Muller C-Elements



a	b	Reset	Set	c	$\overline{c}$
0	0	1	0	0	1
0	1	1	0	c_{old}	$\overline{c_{old}}$
1	0	1	0	c_{old}	$\overline{c_{old}}$
1	1	1	0	1	0
X	X	0	X	0	1
X	X	1	1	1	0

Figure 8: Basic function of a Muller C-Element.

(cf. Figure 7). The functionality of a two-input Muller C-Element can informally be described as follows: the output `c` is assigned the same logic value as the inputs `a` and `b` whenever both inputs are equal (`c=a=b=0` or `c=a=b=1`), and `c` retains its previous value otherwise `c=c_old`. As a Boolean function this can be expressed as

$$c = c_{old} \cdot (a + b) + a \cdot b = c_{old} \cdot a + c_{old} \cdot b + a \cdot b. \quad (2)$$

The Muller C-Element's ability of retaining the old value of output c clearly demands some sort of storage loop. For this storage loop to operate properly the inputs a and b have to stay stable for at least t_{loop}. This delay is defined by the propagation delay through the logic gates plus some additional wiring delays. More specifically, a Muller C-Element's correct behavior rests upon the assumptions that

(i) a single input does not toggle faster than t_{loop} if the initial transition would cause output c to change its value. For example, if input $a = 0$, $b = 1$, and output $c = 1$, input b is not allowed to toggle faster than t_{loop} since the output preserving feedback loop needs time to settle the new value of $c = 0$ (see left part of Figure 8(c));

(ii) input a and b never change their logic level to the opposite value too close to each other. For instance, again starting with $a = 0$ and $b = 1$ both inputs must not change to the opposite polarity $a = 1$ and $b = 0$ within an interval smaller than t_{loop} (right part of Figure 8(c)).

A storage loop with respective timing restrictions is common to all Muller C-Element designs. In an asynchronous design fully relying on handshaking the REQ/ACK control loop ensures that a further input transition is not applied to the Muller C-Gate before the output transition caused by the previous input transition has been acknowledged. In this way the above timing conditions are always met, at least in the fault-free case. However, in our case we have to be aware of the involved timing constraints.

5.1.1. Elastic Pipeline. Figure 9 shows a four-stage implementation of an elastic pipeline (our theoretical analysis predicted a precision of three ticks, so we considered a pipeline

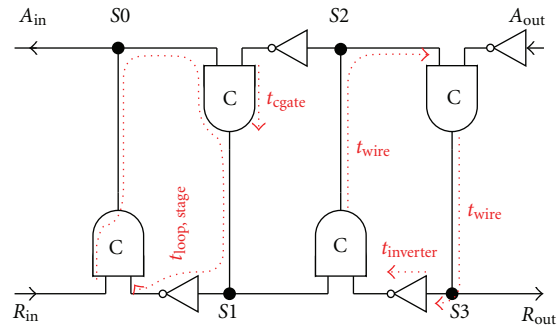


Figure 9: Elastic pipeline design.

depth of four safe) whose regular structure allows for effortless configuration of the FIFO's buffer depth. It is capable of storing up to four transitions applied at R_{in}. The rightmost entry is consumed by a transition on A_{out}, resulting in a right shift of all remaining ones.

In general, the elastic pipeline's way of transition processing provides a very elegant flow control and buffering mechanism as long as some basic timing constraints are maintained. The involved timing paths are depicted in Figure 9. Similar to the Muller C-Element itself, the feedback loops of the elastic pipeline introduce an additional timing condition restricting the input sequence. The path delay $t_{loop,\,stage}$ limits the minimum distance between two subsequent input transitions on R_{in}. The Muller C-Element's input constraint which is characterized by t_{loop} obviously is the less restricitve factor at this point since $t_{loop,\,stage} \approx 2t_{cgate} + 2t_{wire} + t_{inverter}$ with $t_{loop,\,stage} \gg t_{cgate} \approx t_{loop}$.

It is evident that speed, robustness, and area efficiency of a TG-Alg implementation are largely determined by the quality of the available library cell for the Muller C-Element. Therefore we decided to use a customized, transistor level implementation in our ASIC design. It is based on the circuit proposed by van Berkel [44], however, enhanced with several extensions. First of all, dedicated *reset* and *set* inputs (cf. Figure 8(a) for the symbol and Figure 8(b) for the enhanced function) allow to properly initialize the element's state. In the case of our TG-Alg an empty pipeline is used as starting

point. Furthermore, for improved performance two output signals, c and its inverted equivalent $\bar{c}$, are provided, thus saving the extra inverter in the feedback path of the elastic pipeline (cf. Figure 7).

Note that in the TG-Alg's elastic pipeline the output A_{in} is not connected. Again a closer look at Figure 7 reveals that R_{in} corresponds to the clock input signal (remote or local). In turn the feedback output A_{in} would correspond to an acknowledge signal for the incoming clock signal transitions which we omit in accordance with requirements HW-R1 and HW-R7.

In contrast, the far-end interconnection to the Difference Module includes the entire pipeline interface R_{out} and A_{out}. As long $R_{\text{out}} = A_{\text{out}}$, the pipeline is empty and waiting for input transitions and no tick can be removed by the Difference Module. However, as soon as $R_{\text{out}} \neq A_{\text{out}}$ the pipeline holds at least one clock tick which can be consumed by altering A_{out} to the value of R_{out}.

5.2. Counting Ticks. Each of the elastic pipelines presented above manages to buffer incoming clock transitions—four clock transitions in the particular case of the proposed TG-Alg ASIC node design. This buffering scheme is essential because it decouples the time domains between local and remote ticks, thus allowing us to handle them according to a strict, predefined sequence (i.e., without having to consider concurrency), which in turn avoids metastability by design. As already outlined above, the actual +/− counting is implemented by combining a pair of such elastic pipelines, one for the remote ticks and one for the local ones and removing matching ticks from both sides. Note that this removal procedure requires us to have one dedicated instance of the local pipeline per counter.

5.2.1. Difference Module. This module is responsible for an orderly removal of matching ticks. In essence it resembles an asynchronous state machine that first removes a tick from the remote pipeline (as soon as one is available) and only after this removal has been acknowledged enables tick removal at the local side. This procedure ensures that the conditions *remote* ≥ *local* and *remote* > *local*, which directly translate to the fill-level signals $\text{GEQ}^e, \text{GR}^e$, and $\text{GEQ}^o, \text{GR}^o$ are never falsely activated. It turns out from the analysis of the desired behavior that this state machine can be implemented by a Muller C-Element as shown in Figure 10, that is, in contrast to the Muller C-Elements in the elastic pipeline, initialized to 1.

A hardware design optimized for the targeted ASIC manufacturing and implementing the whole +/− Counter Module is presented in Figure 10.

5.2.2. Pipeline Compare Signal Generation. The PCSG module is responsible for generating the comparison results GR^o, GEQ^o and GR^e, GEQ^e based on the fill levels of local and remote pipe. As already mentioned, its function can be partitioned in the processing of odd and even ticks. The PCSG part treating incoming even ticks ultimately triggers the generation of odd ticks by issuing $\text{GEQ}^e, \text{GR}^e$ signals. Similarly, the circuit concerned with odd ticks and controlling GEQ^o and GR^o is responsible for generating even clock ticks.

Generally all output signals of the Pipeline Compare Signal Generation Module ($\text{GEQ}^e, \text{GR}^e$, $\text{GEQ}^o, \text{GR}^o$) as well as all internal logic operations are active *low*. This allowed us to exclusively use inverting basic gates (NAND/NOR instead of AND/OR) within the PCSG design, which contributed to optimizing the speed of the ASIC implementation.

The PCSG performs the conversion from the transition logic used in the elastic pipelines and the Diff Module to state logic. At this point specific care must be taken that the comparison signals *never* switch to the active state before *remote* ≥ *local* and *remote* > *local* conditions, respectively, actually hold. Note that, however, the nature of the tick generation function allows them to stay active for some time even if the respective conditions are no longer fulfilled. From the perspective of the algorithm this means that the early or illegal firing of a rule is disastrous, while the late deactivation of a rule is less critical. In the design of the Diff Module we have already carefully avoided glitches that might be introduced by the removal process. Still, however, we may experience erroneous activations due to a dynamic state of one elastic pipeline, that is, when a transition ripples towards the output of the pipe. For this reason three taps of the local pipeline are combined to ensure that no dynamic effects during tick arrival or removal can compromise the fill level signal, although for the static case only two would be sufficient. In detail, the signals *SLocal*1, *SLocal*2, and *SLocal*3 in conjunction with the NOR_1 gate are used to indicate whether the pipeline holds a single even tick. The fill-level indicators on the remote side (*SRemote*2, *SRemote*3 and *SRemote*2_*N*, *SRemote*3_*N*) are responsible for checking if one or more clock ticks are currently stored in the pipeline. An appropriate combination of local and remote side fill-level signals allows to generate the output signals GEQ^e and GR^e, which represent the conditions *remote* ≥ *local* and *remote* > *local*, respectively. To attain an active fill-level signal GEQ^e, corresponding to *remote* ≥ *local* it has to hold that:

(i) at most one even (tick-↓) clock transition is stored inside the local pipeline which is indicated by NOR_1 (the distinction whether no or one single transition is in the pipeline depends on the state of the ACK, that is, whether the last transition has already been removed by the Diff Module, as this is not relevant for our comparison, involving the ACK signal was not necessary),

(ii) at least one even clock tick is present in the remote pipeline, indicated by signal *SRemote*3_*N*.

These two conditions are combined in a final step via NAND_2, generating the output GEQ^e. Similarly, for the activation of the low active signal GR^e implementing the condition *remote* > *local* the following constraints have to be fulfilled:

(i) again only one even (tick-↓) clock transition is allowed inside the local pipeline, which is assessed by the NOR_1 gate;

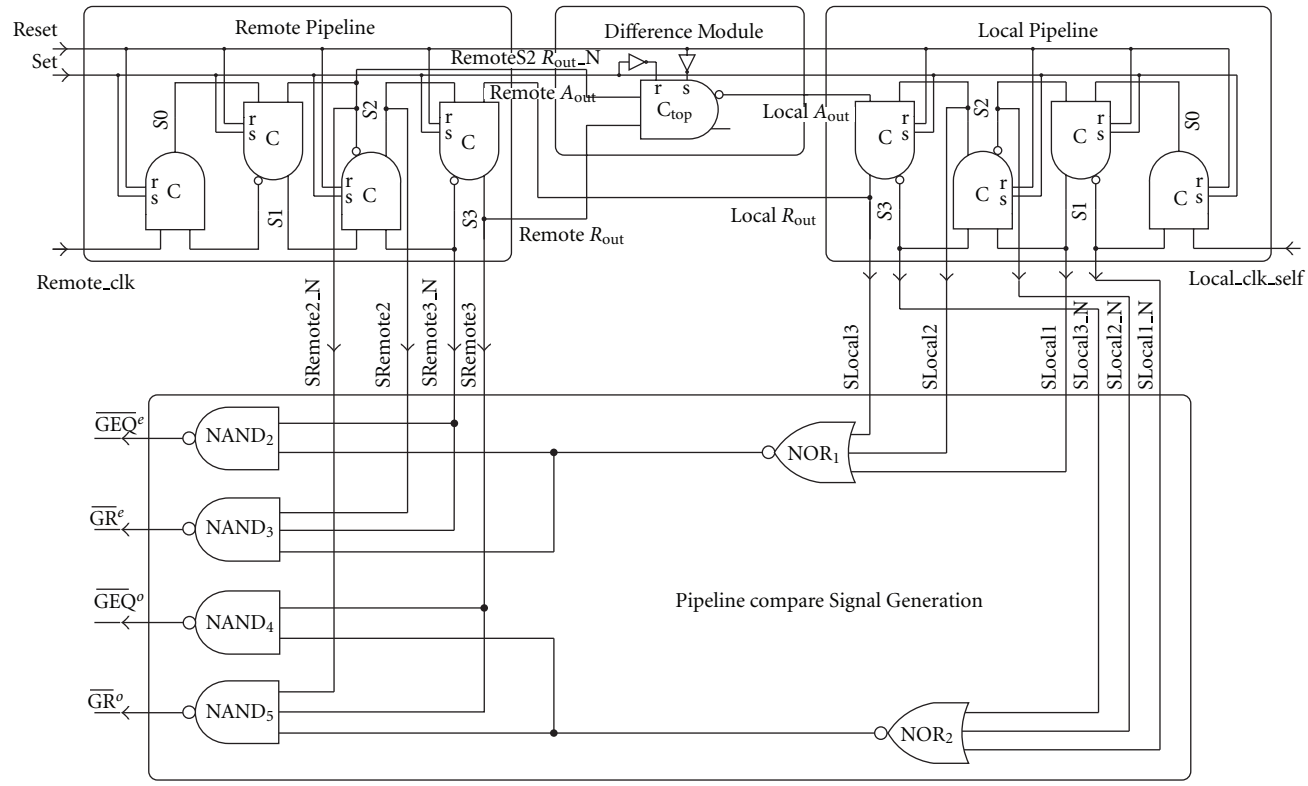


Figure 10: TG-Alg ASIC +/− Counter Module.

(ii) more than one clock tick has to be present in the remote pipeline, an even clock tick in pipeline stage $S3$ and additionally an odd tick in stage $S2$.

These conditions are evaluated by the gate $NAND_3$ via signals *SRemote2_N* and *SRemote3_N* in conjunction with the output of NOR_1. The activation of the signals GEQ^o and GR^o follows by analogous means, simply treating odd instead of even input signals.

We have carefully analyzed the dynamic behavior of the pipe to confirm that our solution can handle all possible dynamic effects caused by transitions rippling through the elastic pipelines.

5.3. Generating Ticks. The final processing step of every TG-Alg node is concerned with the evaluation of the counter fill levels and the generation of new clock ticks according to the "Relay Rule" and "Increment Rule" from Algorithm 3. Here a move back from state logic to transition logic needs to be performed, which requires the careful consideration of possible glitches. After all, in transition signaling every signal change is treated as meaningful data.

5.3.1. Threshold Modules. Four distinct threshold circuits allow to separately evaluate all output signals of a node's $(3f + 1) + /-$ Counter Modules. As depicted in Figure 11, two threshold circuits are responsible for processing the fill-level signals GEQ^e and GR^e for even ticks. This way they implement the tick generation algorithm's "Relay Rule" and "Increment Rule" for falling transitions by virtue of threshold gates with threshold $f + 1$ and $2f + 1$, respectively. In the same way their odd counterparts treat the signals GEQ^o and GR^o.

For the implementation of a single threshold function several possibilities exist. We have evaluated them according to the following criteria.

(i) Low Propagation Jitter: As outlined in Section 3.1 the algorithm's correctness and performance ultimately rely on the ratio Θ of different timing path delays within the TG-Alg design. Therefore we do not want different paths to have substantially different propagation delays (e.g., comprise a different number of logic stages).

(ii) Low Propagation Delay: The threshold module's propagation delay directly adds to the TG-Alg's round-trip time and thus impacts performance.

(iii) Robustness: Since we are heading for a fault-tolerant solution we do not allow a module to compromise the overall robustness. This rules out solutions based on summing up currents or charges in the analogue domain.

(iv) Area Overhead

(v) CMOS Technology: As the approach is targeted for digital CMOS circuits, the threshold gate should as well be implementable in CMOS technology, preferably with standard cells.

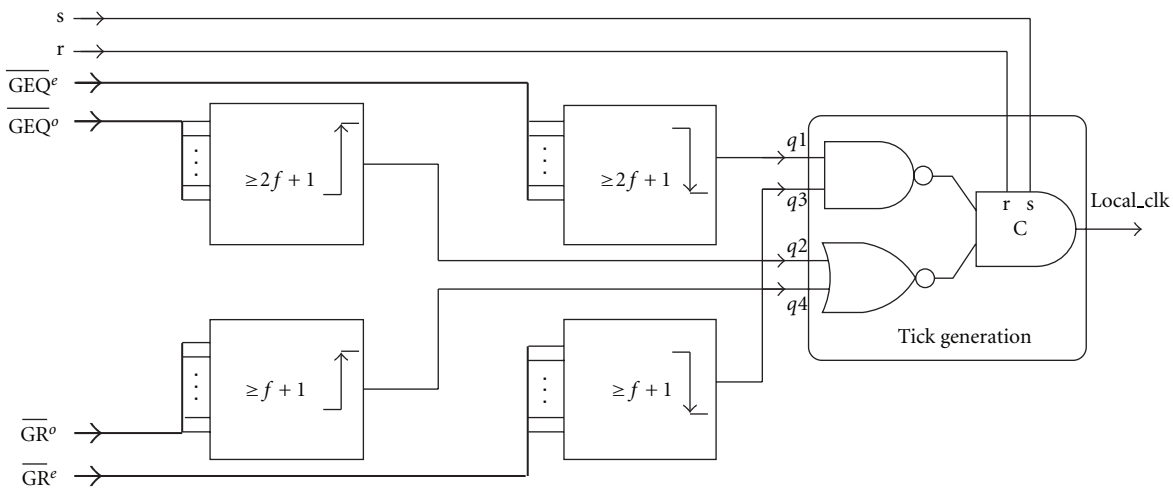


Figure 11: TG-Alg ASIC Threshold Modules and Tick Generation.

In [45] we have made an elaborate comparison of the available approaches and identified the sum of products scheme as the one that best matches these constraints, although its area overhead is quite substantial.

For the needed implementation of an m-out-of-k design $\binom{k}{m}$ product terms have to be computed and then summed up. The threshold circuits of the ASIC TG-Alg have been designed for 11 input signals. The resulting 4-out-of-11 implementation of the $f + 1$ threshold circuit yields 330 product terms. These product terms have to be summed up in a treelike cascaded architecture since no elementary gates with a fan-in of 330 are available in the ASIC target technology. A notable peculiarity of the sum of products implementation is the fact that for a given configuration of $n - 1$ inputs with $n - 1 = 3f + 1$, the required threshold functions $f+1$ and $2f+1$ can be converted into each other by simply inverting all input and output signals, therefore only one function has to be designed.

Although the chosen sum of products implementation is beneficial in many respects, it cannot be implemented such as to operate completely glitch-free. This is not an implementation deficiency, but a general behavior of asynchronous state logic that tends to produce glitches on the outputs while processing its inputs as long as no strict restriction on the input sequence is ensured [46]. At this point we can take advantage of the separation between odd and even ticks. Having the circuit blocks for odd and even ticks operating in alternation, we can determine a period of inactivity for every threshold gate, during which it may produce glitches without jeopardizing the proper overall operation of the TG-Alg, provided these glitches are orderly masked. It will be the task of the Tick Generation Module (see Section 5.3.2) to provide this masking, and we will have to consider timing constraints for the allowed duration of glitches.

5.3.2. Tick Generation Module. The task of the Tick Generation Module (see Figure 11) is to actually generate and broadcast the next tick, as soon as the rules implemented in the threshold module indicate it is time to do so. It is specifically here where the conversion from state logic back to transition logic takes place.

In the Tick Generation Module the four threshold circuit outputs q_1, q_2, q_3, and q_4 are combined by simple logic gates in a way such that only valid clock ticks are generated, in essence it handles the concurrency of the two rules of the algorithm. Furthermore, the Tick Generation Module has to ensure that after generating a clock tick the TG-Alg's clock output remains stable despite the fact that the outputs of the threshold circuits might toggle due to glitches. This retention of the clock output is enabled mainly by the Muller C-Element at the output which only issues a new tick if *both* inputs indicate to do so. However, since the storage loop of the Muller C-Element needs stable inputs during its settling time the outputs of the threshold circuits have to be stable for a small time interval before and after a new tick is generated. This safety window must be ensured by the timing constraints. Assuming that all implementation constraints are fulfilled and taking the above-mentioned considerations into account, a new tick is generated only if

(i) the threshold circuits responsible for the generation of the last tick issued (by providing enabled input signals GEQ, GR) have become inactive again;

(ii) at least one of the two threshold circuits responsible for the generation of the new tick gets activated.

Note that by these rules the generation of an odd tick $k + 1 \in \mathbb{N}_{\text{odd}} := 2\mathbb{N} + 1$ is triggered only if the last tick generated was even ($k \in \mathbb{N}_{\text{even}} := 2\mathbb{N}$). A thorough analysis of the described tick generation process, conducted in the context of the DARTS project and published in [33], shows that the presented approach is sufficient to avoid that old and new instances of GR^o and GEQ^o get mixed up. The main message of this formal analysis—the derived timing constraints for the hardware implementations—will be given subsequently.

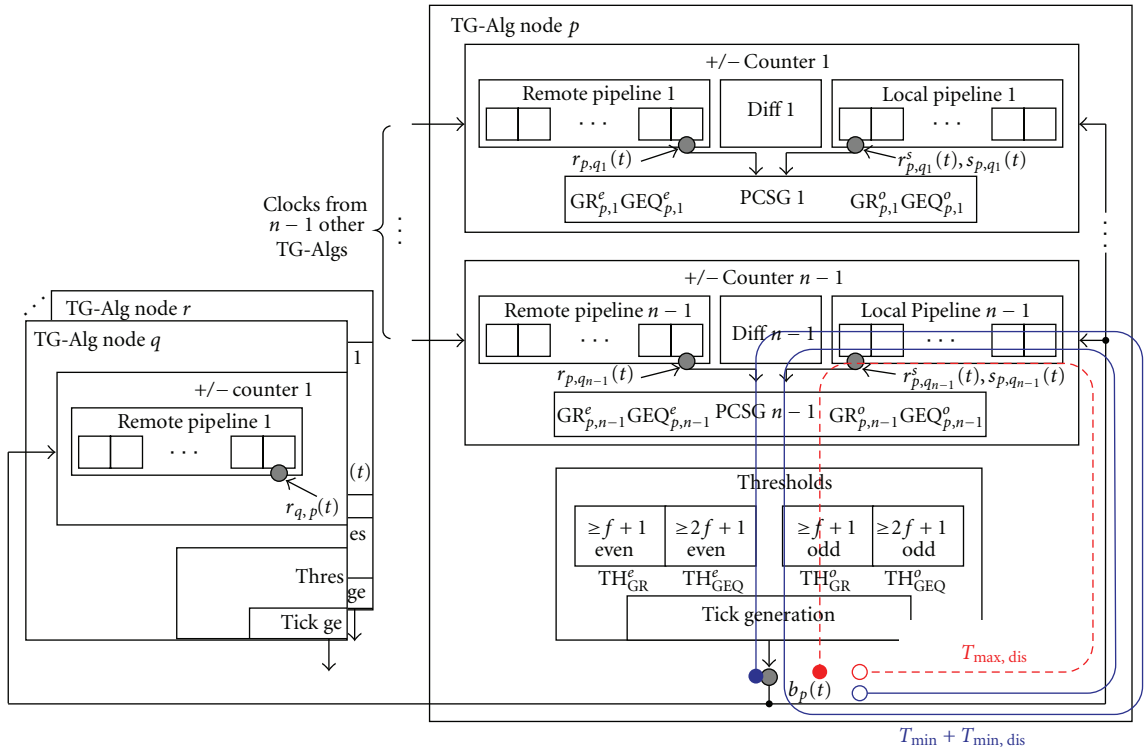


FIGURE 12: Timing paths of the interlocking constraint.

5.4. Timing Constraints. The correct behavior of the tick generation approach of Algorithm 3 relies on some particular timing constraints. In fact, implementation-specific constraints on path delays have to hold. The most important one is given by the *Interlocking Constraint* that ensures that TICK(k) and TICK($k+2$) messages do not interfere with each other.

Constraint 1 (interlocking). $T_{\max,\,\mathrm{dis}} \leq T_{\min} + T_{\min,\,\mathrm{dis}}$ must hold.

With the delay paths.

$$\begin{aligned} T_{\max,\mathrm{dis}} &:= \tau^+_{\mathrm{TH}} + \max\left(\tau^+_{\mathrm{GR}}, \tau^+_{\mathrm{GEQ}}\right) + \tau^+_{\mathrm{loc}} \\ T_{\min} &:= \tau^-_{\mathrm{TH}} + \min\left(\tau^-_{\mathrm{GR}}, \tau^-_{\mathrm{GEQ}}\right) + \tau^-_{\mathrm{loc}} + \tau^-_{\mathrm{Diff}} \\ T_{\min,\mathrm{dis}} &:= \tau^-_{\mathrm{TH}} + \min\left(\tau^-_{\mathrm{GR}}, \tau^-_{\mathrm{GEQ}}\right) + \tau^-_{\mathrm{loc}}, \end{aligned} \tag{3}$$

$T_{\max,\mathrm{dis}}$ represents the slowest disabling path starting and ending at the tick generation output of the respective node. $T_{\min}$ corresponds to the fastest path for generating a clock tick, whereas $T_{\min,\,\mathrm{dis}}$, analogously to $T_{\max,\,\mathrm{dis}}$, accounts for the minimum deactivation time of the previous clock tick which in turn enables the generation of the next clock tick. Figure 12 graphically presents TG-Alg node's opposing interlocking delay paths. Note that the involved paths only include design units at a local node. This locality of Constraint 1 considerably facilitates designing the path delays accordingly.

The interlocking constraint is not the only relevant timing bound. In short, it also has to hold that ticks in the local and remote pipelines get removed fast enough to inhibit excessive queuing of ticks. An additional timing constraint bounds the fastest remotely triggered generation of a new tick in relation to the slowest locally processed one. Moreover, the start-up (booting) of all correct nodes has to be in a certain time interval. A detailed description of these additional constraints can be found in [38].

6. Results

Based on the design presented in the previous section, we have implemented an ASIC prototype in a 0.18 μm technology. We have chosen this relatively large feature size, since it allows a radiation hard implementation, which is an important feature for the spaceborne applications we had in mind. The threshold gate is configurable for a system up to $f = 3$, that is, $n = 11$. This ASIC implementation allows us a first estimation of cost and performance of the DARTS concept in practice. This section reports on our experiences and measurement results.

6.1. TG-Alg Implementation Characteristics. The analysis of the whole TG-Alg design is conducted by putting together the characteristics of all subunits. For this purpose exact numbers for the hardware effort in terms of gate equivalents and die size will be given. The fully connected network topology clearly implies a quadratic growth of the TG-Net's

Table 1: Hardware effort of a single TG-Alg and its components.

	No. of basic gates	No. of C-Elements	area in [μm^2]	area in %
Remote Pipeline	—	4	944	0.20
Local Pipeline	—	4	944	0.20
Difference Module	2	1	270	0.06
PCSG	6	—	395	0.08
+/− Counter	8	9	2,553	0.05
11 +/− Counters	88	99	28,083	5.80
$f + 1$ circuit	550	—	51,641	10.67
$2f + 1$ circuit	1,013	—	176,083	36.39
Tick Generation	2	1	303	0.06
Threshold Modules	3,128	1	455,751	94.19
single TG-Alg	3,218	100	483,862	100.00

number of links with node count n and thus also with f, that is, $\mathcal{O}(f^2)$, and has direct impact on the complexity of a TG-Alg's implementation. Notice, however, that we do not advocate building a system with high n; even a value of $f = 3$, as it is available for our experiments, is much better than the "single fault assumption" usually applied for hardware.

Staying with the flow of the previously presented subblocks the tick queueing and tick counting mechanisms are treated first. The hardware effort for building a TG-Alg's queueing and counting blocks is for a considerable part determined by the amount of incorporated Muller C-Elements. Considering the remote and local elastic pipelines as well as the Difference Module, the Muller C-Element presents the only relevant building block, whereas the Pipeline Compare Signal Generation Module is assembled using a few basic gates with two and three inputs, respectively. Recall from Figure 7 that $n - 1 = 11$ individual +/− Counter Modules are required—one for each remote TG-Alg. The upper part of Table 1 presents numbers for gate count and silicon area (in the 0.18 μm ASIC target technology) treating submodules as well as the whole design of a +/− Counter. Furthermore, the hardware effort is added up to account for the 11 +/− Counters of the actual TG-Alg implementation. It can be observed that the elastic pipelines are the main contributors to the chip area of each +/− Counter. This is due to the relatively complex structure of the library cell employed for the Muller C-Element with its extra inputs for direct set and reset. Note, however, that we used custom cells designed on transistor level. Had we chosen a gate-level implementation (e.g., a design based on NAND gates), we would have experienced an even higher area overhead and a notable loss of performance.

In contrast to the queueing and counting blocks (+/− Counter) every TG-Alg holds only one Threshold Module—incorporating four threshold circuit units and the Tick Generation Module. As thoroughly described in Section 5.3.1, threshold circuits are purely combinatorial blocks following a sum of products implementation. Given an input width of $n - 1 = 11$, the presented complexity growth with the number of inputs yields 330 and 462 product terms for each of the $f + 1$ and $2f + 1$ threshold circuits, respectively. Therefore the exponential increase with approximately $\binom{3f+1}{f+1}$ is one of the prominent cost driving factors when scaling the tick generation system's resilience $f \leq \lfloor (n-2)/3 \rfloor$ and hence the number of nodes n. Due to the fact that basic standard cell gates like NAND and NOR, which are used in the sum of products implementation, are typically available only with two and three inputs, hardware effort is additionally increased with increasing number of n. This is true for the product terms as well as for the terminal sum term because increasing numbers n and m result in the need for cascading basic gates.

In contrast to the threshold circuits the Tick Generation Module does not suffer from scaling effects since it consists of two basic gates and a single Muller C-Element only. Similarly to the elastic pipelines it benefits from the transistor-level implementation of the Muller C-Element. The lower part of Table 1 lists gate count and area numbers for the involved design units and the Threshold Module block overall.

The comparison of TG-Alg's components in terms of hardware effort, shown in Table 1, reveals that the sum of product threshold circuit implementation accounts for a substantial part of the entire design. Almost 95% of a TG-Alg's chip area is devoted to the Threshold Modules. The enormous hardware effort reflects the threshold circuits' unfavorable scaling with f and n. In general, the Threshold Modules' predominance in hardware effort allows to give an estimate for the scaling of a TG-Alg's chip area following $\approx \binom{3f+1}{f+1}$. This scaling obviously only applies for the used sum of products approach and would be completely different for other implementation technologies. Analogously to the customized Muller C-Element, an applicable enhancement to reduce the sum of products area effort might be given by an optimized transistor-level implementation. Furthermore, the design alternatives presented in [45] might provide reasonable options for improvement.

6.2. Performance Assessment. The following assessment aims at thoroughly characterizing the properties of a running tick generation system. These evaluations give insight on tick generation under varying operating conditions and validate the fault tolerance properties (worst-case scenarios) of the DARTS approach. A tick generation system composed of

$n = 8$ (and $f = 2$) fully interconnected ASIC nodes ($U1$ to $U8$) was assembled for all evaluations of the cluster.

In the context of average-case experiments the assessment of implementation and operation characteristics is certainly of interest. In particular, stability considerations arise when recalling that the primary goal of the DARTS clocking scheme is to provide conventional synchronous circuits with a fault-tolerant clock. On the one hand the asynchronous nature of the TG-Alg implementation allows the design to adapt its operation to varying conditions, thus increasing its robustness. On the other hand this flexibility might be problematic from the synchronous unit's point of view since it is controlled by the adaptive, thus varying, TG-Alg clock. Due to the TG-Algs' asynchronous implementation a certain degree of operation parameter sensitivity can be expected.

6.2.1. Frequency. The attainable clock frequency solely relies on switching delays of the asynchronous circuits and interconnection delays of the remote and local clock lines. Using predictions from theory the tick generation scheme's frequency can be bounded by the synchronization property Progress (P) together with the tick generation path T^{-}_{first}:

$$f_{\text{average}} = \left[\frac{1}{2}T_P, \frac{1}{2}T^{-}_{\text{first}}\right]. \tag{4}$$

The path given by T_P denotes the slowest possible generation of a subsequent tick, while T^{-}_{first}, and $T_{\min}$ represent the fastest remotely, and locally triggered tick generation, respectively. The required delay parameters can be extracted from the ASIC design files. Together with delays for the chip interconnect this is sufficient to give a sound estimation of the achievable clock frequency. For the DARTS design, $T^{-}_{\text{first}} \approx 6\,\text{ns}$ and $T_{\min} \approx 6\,\text{ns}$ with an assumed interconnect delay of 1 ns lead to an expected $f_{\max} \approx 71\,\text{MHz}$.

Measurements of a similar path for T^{-}_{first} showed a delay of 7 ns. This path involves `remote_clk[·]` input pin, next 6 Muller C-Elements, the PCSG unit, the Threshold Modules including tick generation, and finally `local_clk` output pin. Analogously to the above examination the delay for $T_{\min}$ is measured as 7 ns. The path is also quite similar comprising the `local_clk_self` input pin followed by 5 Muller C-Elements, the PCSG unit, the Threshold Modules, and Tick Generation block, ending at the `local_clk` output pin.

The difference of 1 ns between measurements and design files is mainly due to the fact that the measurement setup does not—unlike the evaluation of the design files—use the shortest path through the threshold circuits. (This is due to the fact that a system of 8 nodes with $f = 2$ was used for the measurements; however, the paths considered in the design files use the faster paths of $n = 5$ and $f = 1$). From theory, however, it is clear that the rate based on the fastest paths (T^{-}_{first} and $T_{\min}$, resp.) can only be maintained for a short period, that is, either the remote or the local pipeline has to be full while ticks at the opposite side arrive at T^{-}_{first} or $T_{\min}$, respectively. As soon as the previously filled pipeline gets emptied the clock rate will notably slow down which leads to the observed average frequency of $f_{\text{average}} = 54\,\text{MHz}$.

Table 2: Cluster of 8 standard nodes: voltage scaling.

Core voltage in (V)	Avg. frequency in (MHz)	Current ASIC U6 in (mA)	Current all in (mA)
1.3	38	11.7	100
1.4	43	15.1	126
1.5	47	17.6	150
1.6	50	20.6	178
1.7	52	23.8	204
1.8	54	27.2	233

6.2.2. Operation Condition Dependency. As mentioned above, the switching speed of the circuits is likely to be a function of supply voltage. Moreover, digital CMOS circuits are also known to be sensitive to temperature variations. Both effects are typically encountered in normal operation modes. The mentioned voltage dependence of a CMOS circuit can be approximated by deriving the delay times for a single gate and essentially boils down to

$$t_{\text{gate}} \approx \frac{C_L}{\beta V_{\text{DD}}}, \tag{5}$$

with C_L being the load capacitance, β and V_{DD} representing the CMOS transistors' gain and supply voltage, respectively [47]. Note that the above mentioned temperature dependence of CMOS circuits is hidden inside β. The carrier mobility (electrons and holes) decreases with increasing temperature, thus β decreases, yielding a slowdown of the circuit as temperature rises. Table 2 shows the measured average frequency and the corresponding current drawn by the design.

As expected from (5) the achieved clock frequency scales proportional to the supplied core voltage. Figure 13(a) presents results of detailed measurements in which the applied core voltage has been changed in 10 mV steps in an interval starting with 1.30 V and ending at the nominal voltage of 1.80 V. An improved illustration of the measurement data which makes the correlation of voltage and clock frequency more evident is given in Figure 13(b). Core voltage and frequency are given in percentages of their respective maximum value. This way it can be observed that a voltage change of 1% yields approximately 1% variation in clock frequency (red line in Figure 13(b)). The strong impact of the core voltage on the operating frequency of the asynchronous tick generation implementation meets the expectation. A second important factor of influence is temperature. Again, according to (5) the switching speed and propagation delay of CMOS circuits scale indirectly proportional to temperature. This expectation has also been confirmed by the measurements.

6.2.3. Short-Term Jitter. Short-term fluctuations of the frequency and discontinuities in the clock periods during fault-free operation are expected from at least two sources. Firstly, the above-mentioned supply voltage dependence will project its voltage jitter to a frequency jitter. Temperature changes

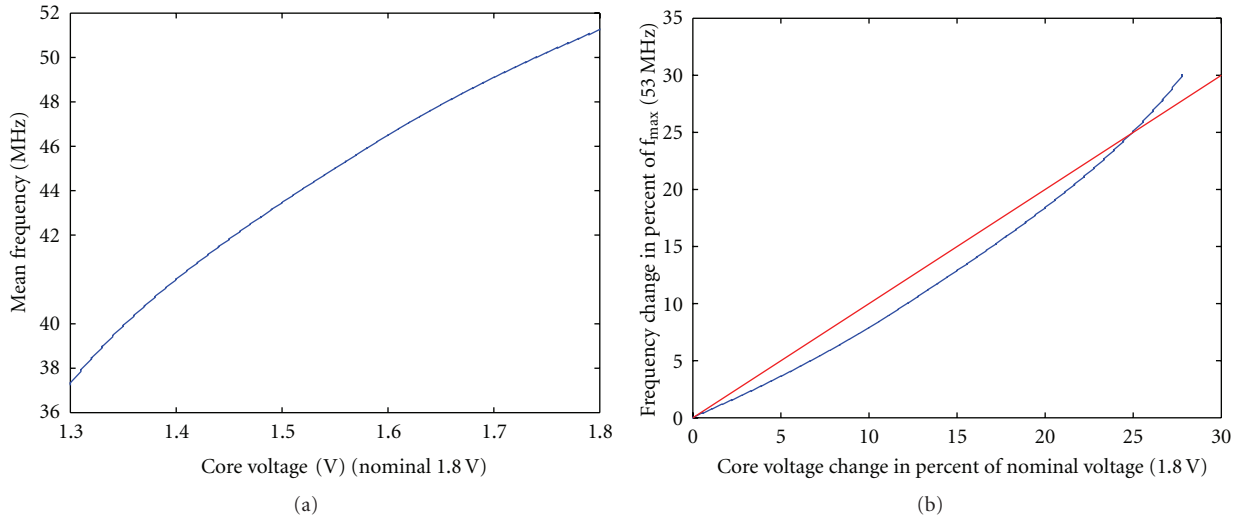
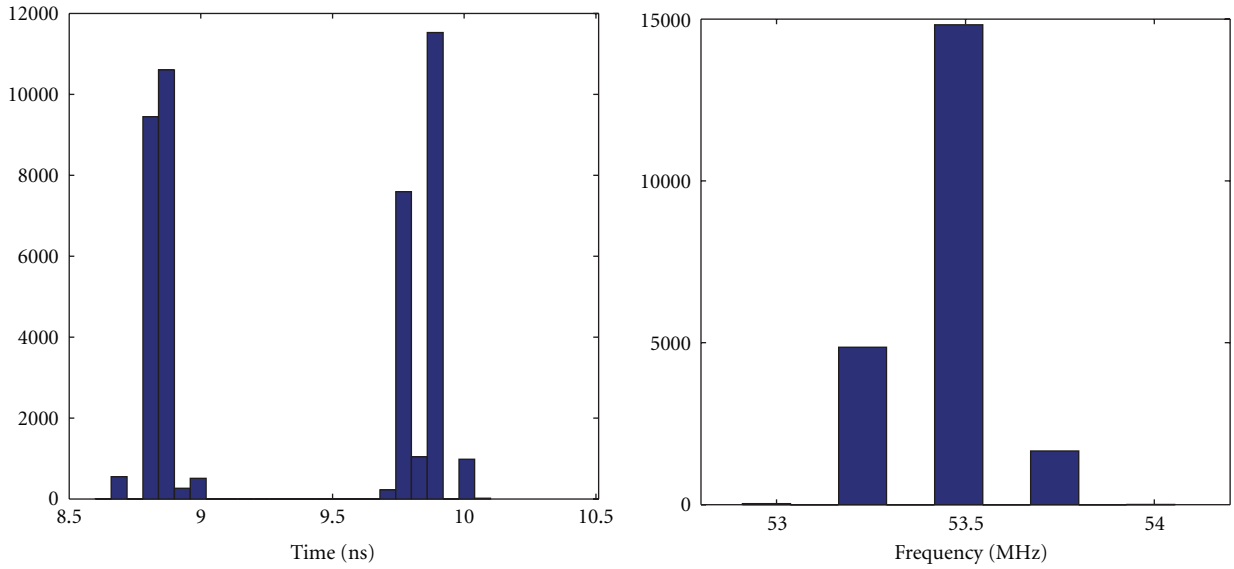


Figure 13: DARTS cluster's mean clock frequency core voltage dependence.



(a) Half periods (b) Clock frequency

Figure 14: Single clock evaluation of a running standard node cluster.

are considered to be too slow to yield perceivable short-time effects, but since propagation delay is known to be affected by thermal noise, so will be the frequency produced by our system. The evaluations are based on short-time measurements with very high resolution (up to 10 GS/s) including approximately 40,000 clock transitions. These measurements aim at characterizing the clock of a single node running within a DARTS cluster.

The measured half periods are presented in the histogram plot shown in Figure 14(a). Two cluster points can be identified, one at $\approx$ 8.7 ns and the other at $\approx$ 9.8 ns. A separate examination (not shown) revealed that these accumulation points correspond with the distributions of the *HI* and *LO* periods. This observation can be explained by the slightly different processing speed of the respective clock signals, in particular, (a) the separate processing of rising and falling transitions in our implementation yielding different path delays and (b) the different timing behavior (rise time/fall time, e.g.,) of Muller C-Elements for rising and falling transitions. Figure 14(b) presents the distribution of the clock frequency with a mean frequency of 53.4 MHz and a standard deviation of 0.153 MHz.

6.2.4. Long-Term Jitter. In the tick generation system's long-term operation especially the effect of varying temperature is expected to be noticeable. Self-heating of the TG-Alg chips is anticipated to continuously slow down the tick generation process. The numbers presented in Table 2 show that the stability of the clock frequency heavily depends on

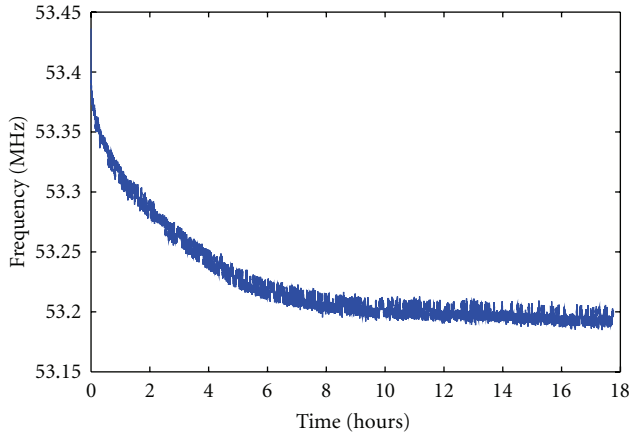


Figure 15: Long-term clock stability: 17-hour run.

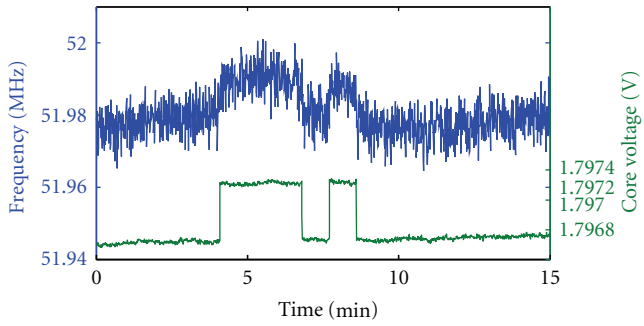


Figure 16: Frequency and voltage trace showing power supply variations.

the stability of the operating conditions. Figure 15 presents a long-term assessment of a node's mean clock frequency with an evaluation interval of more than 17 hours. It can be observed that clock frequency noticeably decreases over time by about 250 kHz. The operating conditions, that is, core voltage and ambient temperature, were not varied in this experiment setup. The measurements start with all nodes in reset state with no activity. Thus no mentionable current is drawn by the chips. As soon as the reset gets deactivated the designs start to draw substantial current that contributes to self-heating of the running chips and causes the asymptotic decrease of the mean clock frequency depicted in Figure 15.

A 15-minute snapshot of another frequency measurement including a high resolution trace of the core voltage is presented in Figure 16. In this figure it can be observed that a discrete jump of the core voltage is directly followed by a frequency jump. This behavior perfectly fits into the design's voltage dependence presented earlier. In the depicted measurement the voltage changed by $\approx 0.5\,\mathrm{mV_{rms}}$ which led to the aforementioned shift of average frequency by 10–20 kHz. The initial cause for the observed minor voltage change is hidden in the used digital power supply which has quantization steps of $0.5\,\mathrm{mV_{rms}}$. The correctness of this interpretation for the frequency jumps has additionally been confirmed by crosscheck measurements with analog power supplies. In these evaluations overall increased frequency

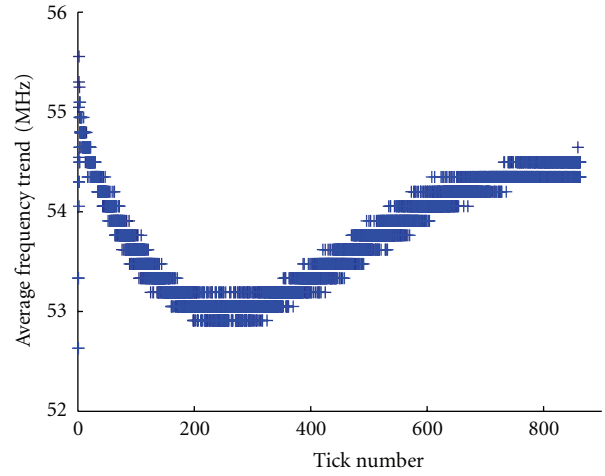


Figure 17: Mean frequency trend over all 8 nodes.

jitter was observed due to the higher level of voltage noise. However, neither discrete steps in the voltage level nor in the mean frequency were encountered.

In addition to the presented assessment of a single clock signal, the clock signals of the whole ensemble were evaluated as well, albeit with a lower precision, since these measurements had to be performed with a logic analyzer, whose time resolution was limited to 250 ps. The main interest clearly resides in the synchronization of the clock ensemble. Detailed short-term measurement showed that for the fault-free case the ensemble starts with tight synchrony and remains closely synchronized (the small initial offset is due to differences in the propagation of the reset signal). Under normal conditions, that is, nominal core voltage and room temperature, evaluations yielded initial offsets in the range of 1 ns to 1.5 ns. In these short-term measurements the maximum skew among any two TICK(k) clock transitions never exceeded its initial offset of 1.5 ns. Hence, a fault-free clock ensemble running under nominal operating conditions has precision $\pi = 1$. In Figure 17 all 8 nodes' frequencies of a DARTS cluster (starting from reset state) are depicted. (To enhance the expressiveness of the graph the data values actually have been smoothed to compensate for the limited resolution of the logic analyzer. Note that this did *not* affect the general trend but only the magnitude of the frequency changes). It can be observed that the frequencies of all DARTS clocks change jointly, thus yielding close synchronization.

6.3. Fault Tolerance Properties. Up to now all evaluations have assumed TG-Alg nodes operating fault-free. In contrast, the evaluations presented in this paragraph consider scenarios with faults artificially introduced into a running cluster of 8 nodes (which by design should be resilient to $f = 2$ Byzantine faults). In the conducted experiments the consequences of crashing TG-Alg nodes are examined in particular. The node crash scenarios are implemented by resetting one or two nodes of the DARTS cluster. Note that these scenarios do not necessarily have the benign properties

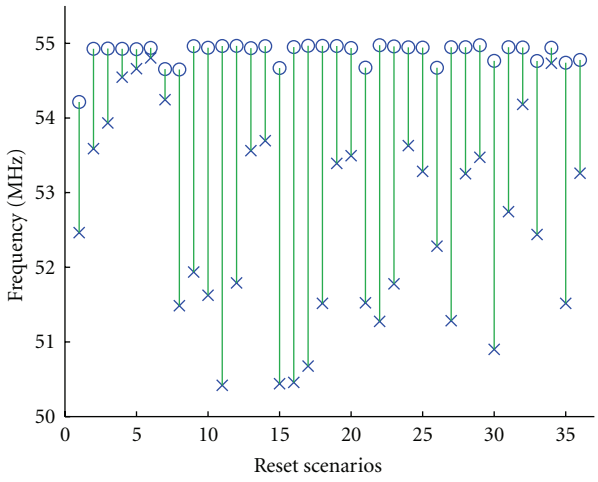


Figure 18: Mean Frequency (o) before and (x) after reset of 1 or 2 nodes.

of crashes as they are assumed in distributed systems. Even stuck-at faults can be outside the scope of the crash fault scenario. For instance, an early clock transition, that is, changing a clock rail from *HI* to *LO* (stuck-at-0) through the activation of a node's reset is already within the class of malicious/Byzantine failures. All combinations of scenarios with one or two nodes crashing yielded Figure 18. For each of the reset scenarios the mean frequency before and after the crash has been derived from measurement data. The lines interconnecting these two mean frequency values illustrate the actual drop of the clock frequency. As anticipated, in all 36 reset scenarios the deactivation of nodes leads to a decrease of the mean clock frequency. This slowdown is quite natural since for the remaining nonfaulty nodes of the cluster the crashing of nodes implies that one or two of the previously $2f + 1$ fastest node(s) has/have been deactivated. Hence the correct nodes have to wait until tick messages are received from slower nodes which are still up and running, consequently leading to additional delay before the next tick can be generated. Note that due to small differences in propagation delays and the close synchronization of all clocks, each node's set of $2f + 1$ fastest neighbors might be different. This leads to the effect that the reset of each node at least slightly influences the clock overall clock frequency.

The synchronization precision π represents one of the most important synchronization properties of the DARTS tick generation system. It may simply be assessed by measuring the clocks' relative offset. However, this evaluation is unlikely to reflect worst-case conditions. As already pointed out in Section 6.2.4 the computation and interconnect delays of the DARTS cluster are almost perfectly matched—which yields a precision $\pi = 1$. Thus an appropriate scenario has to be derived and established for worst-case measurements. Figure 19(a) shows the generic setup to statically force a system of $3f + 2$ TG-Alg nodes into an operation mode with worst-case precision. The only relevant parameters for this scenario are given by the interconnecting remote delays τ_{rem}. As depicted, a set of f nodes have to be *faulty* in a way that no TICK(k) messages are delivered to a second set of f *slow* TG-Algs. It further has to be ensured that ticks sent among the set of slow nodes as well as those received from the group of $f + 2$ fast TG-Algs are issued with the maximum remote delay τ^{+}_{rem}. Connections not explicitly shown in Figure 19(a) can be assumed to have delay τ^{-}_{rem}. More formally speaking, in a system where P denotes the set of all nodes there are three distinct sets of TG-Algs with A comprising the fast nodes, B the slow, and the F faulty ones. The remote delays from p to q in this setup are given by

$$\begin{aligned}
\tau_{\text{rem}}(p \in A, q \in B) &= \tau^{+}_{\text{rem}} \\
\tau_{\text{rem}}(p \in F, q \in B) &= +\infty \\
\tau_{\text{rem}}(p \in B, q \in B) &= \tau^{+}_{\text{rem}} \\
\tau_{\text{rem}}(p \in P, q \in A) &= \tau^{-}_{\text{rem}} \\
\tau_{\text{rem}}(p \in P, q \in F) &= \tau^{-}_{\text{rem}}.
\end{aligned} \tag{6}$$

To get a better understanding for the reasons why this static evaluation setup represents a valid worst-case scenario for the tick generation system, Figure 19(b) depicts an execution trace of the relevant (non-faulty) nodes. As indicated in the trace, it is assumed that all nodes start at approximately the same time by issuing TICK(0). For the example, it is assumed that τ_{rem} alone determines the processing speed of the tick generation system. (Recall from Section 3.1 that only the ratio Θ of fastest to slowest path determines the algorithm's properties, thus it makes no difference if τ_{rem} or the whole delay of the tick generation path is considered in the experiment scenarios.) In the given setup, set A comprises $f + 2$ fast TG-Algs. Together with f fast, but faulty nodes $\in F$, ticks are generated continuously at a rate determined by τ^{-}_{rem} and according to the algorithm's "Increment Rule" ($= 2f + 1$ threshold). Analogously, the f slow TG-Algs $\in B$ also start to issue clock ticks triggered by the "Increment Rule" (I), however, at a period determined by τ^{+}_{rem}. Thus, group A starts "running away" with τ^{-}_{rem} while the slow group B "runs behind" with period τ^{+}_{rem}. At some point the slow nodes' flow of issuing ticks changes to the operation mode where the "Relay Rule" (R) takes over tick generation. This switching point is reached when TICK(k) messages arrive at the slow nodes, indicating that the fast remote nodes are ahead by at least one tick, that is, $k > l$, with ℓ being the current local tick number. This way the "Relay Rule" ensures that the system stays in a synchronized state. The maximum offset in time between the first sending of TICK(k) at t_k and the last sending of TICK(k) at t'_k for any pair of correct nodes p, q can be used to derive the cluster's precision π (for more precise calculations of the maximum clock skew of correct nodes refer to the detailed formal analysis presented in [33, 48]).

The implementation of the above described evaluation scenario confirmed the predictions from theory. As expected the measurements showed that the synchronization of fault-free nodes only depends on the ratio of fastest to slowest path Θ. Via these artificially introduced path delays the worst case ratios of fastest to slowest path could be set to achieve precision $\pi = 3$ which still poses no threat to the clocking

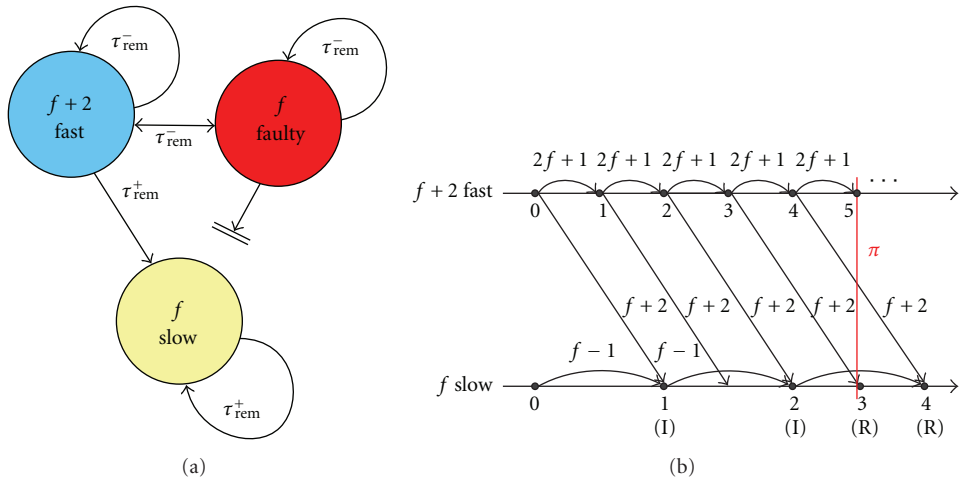


Figure 19: Evaluation scenario to attain worst-case precision π.

approach and can be handled by the buffering of the four-stage elastic pipelines.

7. Conclusions

Considering the increasing need for fault tolerance in general, and the lack of fault-tolerant clocking schemes that allow globally synchronized operation even in the presence of significant skew in particular, we have proposed a novel clock generation approach that closes this gap. DARTS provides a clock with scalable fault tolerance for both, clock generation units as well as interconnect, that is globally synchronized with a bounded precision.

The key idea behind DARTS is to use a tick synchronization algorithm from the distributed systems community whose performance and fault-tolerance features can be formally proven. By moving this originally software-based algorithm to a hardware implementation, the attainable precision can be improved to a level that is suitable for clocking hardware units with reasonable synchrony. When doing so, however, two crucial issues had to be mastered. Firstly, the algorithm had to be appropriately chosen and modified so as to suit to a hardware implementation, and secondly the hardware implementation as such raised considerable challenges. Many of these challenges originated not only from the desire to attain a fast and area-efficient solution, but also from the fact that many facets of the abstract algorithm turned out to be difficult to project to hardware, such as unbounded count values or atomicity of actions.

We have presented a solution that is based on asynchronous logic design, partly based on the original indication principle with handshaking and partly on timing assumptions. The latter turned out to be necessary to attain the desired fault tolerance and also for keeping the clock distribution in a single-rail fashion. Among the key measures for achieving a robust and efficient solution were the reduction of the problem from an absolute to a relative comparison, the differential transmission of the counter values, the realization of the required +/− Counter by means of elastic pipelines, the separated treatment of rising and falling ticks to facilitate the interlocking, and the masking of glitches during idle phases of the threshold gates.

We have reported on the implementation and measurement results obtained with a CMOS ASIC design of the DARTS concept. In this context we have identified the threshold gates as the major contributors to area consumption. Beyond serving as a proof of concept, our experiments have investigated the clocking scheme's behavior in terms of clock stability, clock jitter, precision of synchronization, and fault tolerance. Overall the expectations from the theoretical models could be confirmed.

Although the DARTS approach as it is already represents an attractive solution for fault-tolerant clock generation, we are still considering a lot of future improvements. One of these is the use of a weaker fault model in order to reduce the area and improve the scaling with the number n of nodes. Other ideas include the pipelining of clock ticks to speed up the clock frequency or to build an efficient global communication scheme on top of the DARTS clocks that is metastability-free by construction.

Acknowledgments

This work originates in the *DARTS* (Distributed Algorithms for Robust Tick-Synchronization) project, which has been a joint effort of Vienna University of Technology and RUAG Space, see http://ti.tuwien.ac.at/darts for details. The authors would like to thank all DARTS projects members from RUAG Space GmbH as well as the Vienna University of Technology, specifically Ulrich Schmid for initiating the DARTS project and pushing it forward all the time. Many thanks also go to Matthias Függer for his never ending enthusiasm and his great job on the formal aspects of the novel tick generation approach. Last but not least they are grateful for the support by the Austrian bm:vit via FIT-IT project grant *DARTS* (809456-SCK/SAI) and the Austrian FWF project Theta (P17757).

References

[1] G. Moore, "Progress in digital integrated electronics," in *Proceedings of the Technical Digest of IEEE International Electron Devices Meeting (IEDM '75)*, pp. 11–13, 1975.

[2] "International technology roadmap for semiconductors," 2009, http://www.itrs.net/.

[3] L. Wissel, S. Pheasant, R. Loughran, C. LeBlanc, and B. Klaasen, "Managing soft errors in ASICs," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 85–88, May 2002.

[4] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design and Test of Computers*, vol. 22, no. 3, pp. 258–266, 2005.

[5] D. Rossi, M. Omaña, F. Toma, and C. Metra, "Multiple transient faults in logic: an issue for next generation ICs?" in *Proceedings of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT '05)*, pp. 352–360, October 2005.

[6] A. Narasimhan and R. Sridhar, "Impact of variability on clock skew in H-tree clock networks," in *Proceedings of the 8th International Symposium on Quality Electronic Design (ISQED '07)*, pp. 458–463, March 2007.

[7] E. G. Friedman, "Clock distribution networks in synchronous digital integrated circuits," *Proceedings of the IEEE*, vol. 89, no. 5, pp. 665–692, 2001.

[8] P. J. Restle, C. A. Carter, J. P. Eckhardt et al., "The clock distribution of the power4 microprocessor," in *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC '02)*, vol. 2, pp. 144–137, February 2002.

[9] M. Omaña, D. Rossi, and C. Metra, "Fast and low-cost clock deskew buffer," in *Proceedings of the 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT '04)*, pp. 202–210, October 2004.

[10] D. W. Dobberpuhl, R. T. Witek, R. Allmon et al., "A 200-MHz 64-b dual-issue CMOS microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 11, pp. 1555–1567, 1992.

[11] S. D. Naffziger, G. Colon-Bonet, T. Fischer, R. Riedlinger, T. J. Sullivan, and T. Grutkowski, "The implementation of the itanium 2 microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 11, pp. 1448–1460, 2002.

[12] W. Hu, R. Wang, Y. Chen et al., "Godson-3B: a 1 GHz 40W 8-core 128GFLOPS processor in 65 nm CMOS," in *Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC '11)*, pp. 76–77, 2011.

[13] J. Calamia, "China's godson gamble," *IEEE of Spectrum*, vol. 48, no. 5, pp. 14–16, 2011.

[14] N. Seifert, P. Shipley, M. D. Pant, V. Ambrose, and B. Gill, "Radiation-induced clock jitter and race," in *Proceedings of the 43rd Annual IEEE International Reliability Physics Symposium*, pp. 215–222, April 2005.

[15] D. M. Chapiro, *Globally-asynchronous locally-synchronous systems*, Ph.D. thesis, Stanford University, Palo Alto, Calif, USA, 1984.

[16] R. Ginosar, "Fourteen ways to fool your synchronizer," in *Proceedings of the 9th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC '03)*, pp. 89–96, May 2003.

[17] J. Muttersbach, T. Villiger, and W. Fichtner, "Practical design of globally-asynchronous locally-synchronous systems," in *Proceedings of the 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC '00)*, pp. 52–59, April 2000.

[18] R. Dobkin, R. Ginosar, and C. P. Sotiriou, "Data synchronization issues in GALS SoCs," *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, vol. 10, pp. 170–179, 2004.

[19] M. S. Maza and M. L. Aranda, "Interconnected rings and oscillators as gigahertz clock distribution nets," in *Proceedings of the 13th ACM Great Lakes Symposium on VLSI (GLSVLSI '03)*, pp. 41–44, 2003.

[20] M. S. Maza and M. L. Aranda, "Analysis and verification of interconnected rings as clock distribution networks," in *Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI '04)*, pp. 312–315, 2004.

[21] S. Fairbanks, "Method and apparatus for a distributed clock generator," 2004, US patent no. US2004108876, http://v3.espacenet.com/textdoc?DB=EPODOCn&IDX=US2004108876

[22] S. Fairbanks and S. Moore, "Self-timed circuitry for global clocking," in *Proceedings of the 7th International Symposium on Asynchronous Circuits and Systems*, pp. 86–96, March 2005.

[23] S. Hauck, "Asynchronous design methodologies: an overview," *Proceedings of the IEEE*, vol. 83, no. 1, pp. 69–93, 1995.

[24] A. J. Martin, M. Nyström, K. Papadantonakis et al., "The Lutonium:a sub-nanojoule asynchronous 8051 microcontroller," in *Proceedings of the 9th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC '03)*, pp. 14–23, May 2003.

[25] C. Uri, "Terabit crossbar switch core for multi-clock-domain SoCs," in *Proceedings of the 15th Symposium on High Performance Chips (HOT CHIPS '03)*, p. 102ff, 2003.

[26] J. Dama and A. Lines, "GHz asynchronous SRAM in 65nm," in *Proceedings of the 15th International Symposium on Asynchronous Circuits and Systems (ASYNC '09)*, pp. 85–94, May 2009.

[27] A. J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," in *AUSCRYPT '90 Proceedings of the sixth MIT Conference on Advanced Research in VLSI*, pp. 263–278, MIT Press, Cambridge, Mass, USA, 1990.

[28] W. Jang and A. J. Martin, "SEU-tolerant QDI circuits," in *Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC '05)*, pp. 156–165, March 2005.

[29] W. Friesenbichler, T. Panhofer, and M. Delvai, "Improving fault tolerance by using reconfigurable asynchronous circuits," in *Proceedings of the IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS '08)*, pp. 267–270, April 2008.

[30] M. Delvai, *Design of an asynchronous processor based on code alternation logic—treatment of non-linear data paths*, Ph.D. thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2005.

[31] D. Dolev, J. Y. Halpern, and H. R. Strong, "On the possibility and impossibility of achieving clock synchronization," *Journal of Computer and System Sciences*, vol. 32, no. 2, pp. 230–250, 1986.

[32] T. Polzer, T. Handl, and A. Steininger, "A metastability-free multi-synchronous communication scheme for SoCs," in *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS '09)*, vol. 5873 of *Lecture Notes in Computer Science*, pp. 578–592, 2009.

[33] M. Függer, U. Schmid, G. Fuchs, and G. Kempf, "Fault-tolerant distributed clock generation in VLSI systems-on-chip," in *Proceedings of the 6th European Dependable Computing Conference (EDCC '06)*, pp. 87–96, IEEE Computer Society Press, October 2006.

[34] L. Lamport, "The mutual exclusion problem: part I–the theory of interprocess communication," *Journal of the ACM*, vol. 33, no. 2, pp. 313–326, 1986.
[35] L. Lamport, "Arbitration-free synchronization," *Distributed Computing*, vol. 16, no. 2-3, pp. 219–237, 2003.
[36] T. K. Srikanth and S. Toueg, "Optimal clock synchronization," *Journal of the ACM*, vol. 34, no. 3, pp. 626–645, 1987.
[37] J. Widder and U. Schmid, "The Theta-Model: achieving synchrony without clocks," *Distributed Computing*, vol. 22, no. 1, pp. 29–47, 2009.
[38] G. Fuchs, *Fault-tolerant distributed algorithms for on-chip tick generation: concepts, implementations and evaluations*, Ph.D. thesis, Vienna University of Technology, Fakultät für Informatik, Vienna, Austria, August 2009.
[39] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design*, DIMES, 2001.
[40] M. Ferringer, G. Fuchs, A. Steininger, and G. Kempf, "VLSI implementation of a fault-tolerant distributed clock generation," in *Proceedings of the 21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT '06)*, pp. 563–571, October 2006.
[41] D. L. Black, "On the existence of delay-insensitive fair arbiters: trace theory and its limitations," *Distributed Computing*, vol. 1, no. 4, pp. 205–225, 1986.
[42] A. J. Martin, "Compiling communicating processes into delay-insensitive VLSI circuits," *Distributed Computing*, vol. 1, no. 4, pp. 226–234, 1986.
[43] I. E. Sutherland, "Micropipelines," *Communications of the ACM, Turing Award*, vol. 32, no. 6, pp. 720–738, 1989.
[44] K. van Berkel, "Beware the isochronic fork," *Integration, the VLSI Journal*, vol. 13, no. 2, pp. 103–128, 1992.
[45] G. Fuchs, J. Grahsl, U. Schmid, A. Steininger, and G. Kempf, "Threshold Modules—Die Schlüsselelemente zur Verteilten Generierung eines Fehlertoleranten Taktes," in *Proceedings of the Austrian National Conference on the Design of Integrated Circuits and Systems (Austrochip '06)*, pp. 149–156, Vienna, Austria, October 2006.
[46] S. M. Nowick and C. W. O'Donnell, "On the existence of hazard-free multi-level logic," in *Proceedingsof the International Symposium on Asynchronous Circuits and Systems*, pp. 109–120, May 2003.
[47] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, Addison Wesley Longman, Boston, Mass, USA, 1985.
[48] M. Fuegger, U. Schmid, G. Fuchs, A. Steininger, G. Kempf, and M. Sust, "Fault-tolerant distributed tick generation in VLSI systemson-chip," Tech. Rep. 53/2009, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2009.