Fault-tolerant Clock Synchronization with High Precision

Attila Kinali
Max-Planck Institute for Informatics
Saarbrücken, Germany
Email: adogan@mpi-inf.mpg.de

Florian Huemer
University of Technology Vienna
Vienna, Austria
Email: florian.huemer@tuwien.ac.at

Christoph Lenzen
Max-Planck Institute for Informatics
Saarbrücken, Germany
Email: clenzen@mpi-inf.mpg.de

***Abstract*—We present the first FPGA implementation of a distributed clock synchronization algorithm with sub-nanosecond skews that can tolerate arbitrary faults of individual components. Each of n nodes is equipped with its own quartz oscillator and the nodes broadcast their clock pulses to enable synchronization. The algorithm provably maintains synchronization even if fewer than $n/3$ nodes exhibit *arbitrary* faulty behavior. Moreover, as long as more than $2n/3$ nodes remain synchronized, nodes will recover and resynchronize after transient faults.**

Using 4 boards with Cyclone IV FPGAs, our implementation achieves precision better than 300 ps. This is in accordance with the worst-case precision of 870 ps predicted by theory. Furthermore, our experiments demonstrate that nodes recover from transient faults as described above. Finally, frequency stability of the overall system improved by an order of magnitude.

I. Introduction

Reliably clocking complex very large scale integration (VLSI) circuits is a highly challenging problem. The traditional approach of using a global clock tree brings a variety of scalability issues: In high-performance designs, minimizing the clock *skew*, i.e., the time difference between the earliest and latest clock transition arrivals at the clock tree leafs, requires advanced buffer insertion, snaking wires, and wire sizing techniques [1], [2], [3]. These techniques typically rely on high precision delay models or symmetry assumptions [4] of the involved components. In addition, monolithic clock trees make the system dependent on a single clock source and its system-wide distribution. This introduces a single point of failure, entailing that any dependable architecture clocked in this way requires an extremely robust clock tree design; naturally, this aggravates scalability issues even further.

In light of these obstacles, Globally Asynchronous Locally Synchronous (GALS) systems [5] offer a paradigm shift away from centralized clocking. Instead, the system is partitioned into clock domains, each generating and distributing its own clock. Depending on the relation between the clock domains, such systems are called *mesochronous* (same frequency, bounded phase relation), *plesiochronous* (same nominal frequency), or *heterochronous* (else) [6].

While clock generation and distribution for plesiochronous and heterochronous systems are easily realizable by independent sources and distribution layers, these solutions introduce an entire batch of new problems:

- Asynchronous communication across clock domains requires the use of synchronizers for each data path, increasing delays and buffer sizes, and thus decreasing the overall throughput.
- Slight differences in clock speeds may result in different rates of data production and processing, potentially causing buffer overflows.
- Introducing communication (handshaking, etc.) to resolve this issue increases design complexity at the application level and shifts the difficulty of providing strong real-time response guarantees to the application designer.

This advocates re-introducing timing guarantees between the different clock domains, i.e., using mesochronous GALS systems, allowing for higher inter-domain communication throughput and slimmer communication circuits [6], [7], as well as metastability-free communication [8]. However, while such systems may not suffer from throughput penalties and do not rely on a clock tree as the top-level synchronization mechanism, this top-level synchronization is critical to its operation: unless the inter-domain clocking mechanism itself is fault-tolerant, again a single point of failure has been created.

A. Our Contribution

We propose the use of a fault-tolerant distributed algorithm for inter-domain synchronization. We present our algorithm, which is a variant of the clock synchronization algorithm by Lynch and Welch [9], in Section II-B. To make it suitable for hardware implementation, we modify their routine to work by broadcasting simple pulses (as opposed to round numbers). Based on some preliminary results [10], we analyze the fault-tolerance properties and skew of the algorithm in Section II. The algorithm provides the following guarantees:

1) In a system with n nodes, at any point in time any $f < n/3$ nodes may exhibit arbitrary faulty behavior without disrupting synchronization of correct nodes. For example, this includes adverse behavior such as asymmetrically sent pulses, due to, e.g., faulty output drivers that are connected to nodes with slightly different input driver thresholds.
2) Any unsynchronized node resynchronizes in a bounded number of iterations.
3) If local clock sources have small phase drift (cheap quartz oscillators suffice), clock skew is bounded by roughly $4(U + G)$, where U is the uncertainty in communication delays (i.e., difference between minimum and maximum)

and G is the resolution of the time-to-digital converters (TDCs) used to measure phase offsets.

We stress that in absence of faults, the algorithm achieves a better precision of roughly $2(U + G)$, matching the lower bound from [11] up to factor 2. The remaining factor 2 can, in principle, be shaved off, too, but this might not always be the best choice; we discuss this in Section II-B.

Moreover, we discuss a proof-of-concept implementation of the algorithm on Cyclone IV FPGAs in Section III. Our goal here is neither to provide the best possible FPGA realization nor to achieve the smallest possible skews. Rather, we seek to demonstrate (i) that implementing the adapted fault-tolerant distributed clock synchronization algorithm in hardware is feasible, and (ii) that – even with an implementation struggling with the limitations of low-priced of-the-shelf hardware – the algorithm not only works, but offers skews that significantly outperform state-of-the-art fault-tolerant solutions. In the 4-node system we implemented, we observed a maximum skew of 180 ps over 10^9 clock pulses between correct nodes, in the presence of a faulty node. The theoretical analysis implies that letting one node send early pulses to a subset of the remaining nodes only leads to a worst-case skew. We ran several experiments and tested the skew under these worst-case faults, resulting in a maximum skew of 300 ps for the same number of runs. Finally, we verified that nodes successfully resynchronize in bounded time. Due to peculiarities of our setup, namely the phase corrections step size being limited to 300 ps per round, this required 7 s, but the analysis shows that this will be orders of magnitude faster for a faithful implementation of the algorithm.

B. Related Work

A canonical approach to distributed clock synchronization is to let nodes agree on an approximate common notion of time periodically, and readjusting their local clocks to the value upon which they agreed. Early work on reaching approximate agreement in distributed systems [12] lead to fault-tolerant clock synchronization algorithms based on this method, see e.g. [13] for an overview.

In this work, we make use of the algorithm by Welch and Lynch [9], for the following reasons:

(i) Its skew is proportional to the delay uncertainty rather than the maximum delay as, e.g., the algorithm proposed in [14].

(ii) In contrast to more involved algorithms that require multi-round communication for a single resynchronization [13], it is well suited for an on-chip VLSI implementation.

In the context of systems with high dependability requirements, such as automotive and aerospace, similar fault-tolerant clock synchronization algorithms have been deployed at a higher level of abstraction. Examples are the the time triggered protocol (TTP) [15] and FlexRay [16]. Both protocols provide round-wise communication based on TDMA slots aligned via a fault-tolerant clock synchronization primitive. These primitives are also based on the algorithm by Welch and Lynch and have been proven correct, see e.g. [17], [18]. However, due to their involved communication protocols and treatment of special cases, both TTP and FlexRay feature hybrid implementations which join soft- and hardware components; neither is suitable for a slim hardware implementation with multiple nodes on a single GALS chip.

We are aware of only two on-chip distributed clock synchronization algorithms that can tolerate arbitrary behavior of faulty nodes: DARTS [19], [20] and FATAL [21], [22]. Both are based on Srikanth and Toueg's primitive for simulating authenticated reliable broadcast in the presence of arbitrarily failing nodes [14]. DARTS is implemented using clock-less logic, e.g., comprising Muller C-Elements as central basic components. Since gate-level implementations of C-Elements are inefficient (with regard to both space and latency), DARTS lends itself to an ASIC implementation, but does not work well for FPGAs [20]. By contrast, the solution proposed in this work requires standard synchronous design elements only. Consequently, it is well-suited for both FPGA and ASIC implementation.

Furthermore, the worst-case skew exhibited by the algorithm by Srikanth and Toueg, and thus also by DARTS and FATAL, is proportional to the *maximum* communication delay [14], as opposed to the delay uncertainty U which is typically at least an order of magnitude smaller.

In contrast to DARTS, FATAL provides the additional guarantee of *self-stabilization* [23]: after an arbitrary disruption of all nodes' states and the communication medium, correct nodes will re-synchronize again (assuming that sufficiently many nodes recover and recommence executing the algorithm correctly) [22]. This comes at the price of a more complex clock synchronization algorithm, incurring considerable hardware overhead and requiring additional methods to generate fast clocks from the slower FATAL clock. The approach we propose in this work guarantees "limited" self-stabilization: after transient faults, nodes re-synchronize, provided that at any point in time sufficiently many nodes are correctly synchronized. This is a deliberate restriction to keep the algorithm simple; in [10], it is shown how to ensure full self-stabilization by coupling the algorithm to a concurrently running instance of FATAL.

II. Algorithm and Analysis

We modify the algorithm by Lynch and Welch [9] to broadcast simple clock pulses, as opposed to nodes communicating round numbers. In large parts, this modification is inconsequential. However, the algorithm from [9] makes use of the round numbers to achieve a powerful recovery property: any node can resynchronize after a transient fault, provided that out of the n nodes never more than $f := \lfloor (n-1)/3 \rfloor$ are faulty or out-of-sync.

We first describe the assumptions on system behavior underlying the algorithm and its analysis in Section II-A, then give the basic algorithm using clock pulses in Section II-B, and finally present a recovery mechanism that can operate based on clock pulses only in Section II-C.

A. System Model

The system consists of a set V of n nodes that are fully connected by (1-bit) broadcast channels. Each node $v \in V$ is a fault-containment region in the sense defined by Kopetz [24]: a single (physical) fault, such as a gate malfunction, does not directly affect correctness of the components outside the fault-containment region that contains the faulty component. Node v comprises a local physical clock H_v (e.g. a ring oscillator), the circuitry implementing the algorithm's logic for v, and its outgoing links. Note that this means that communication does not use a shared bus, which would be a single point of failure. Any potential application logic clocked by v will be part of its fault containment region as well. Thus, any transient or permanent faults of components (in the fault-containment region) of v affect other nodes only indirectly via communication. A faulty node (i.e., one whose containment region contains faulty components) can behave arbitrarily; in particular, it may send a clock pulse to a subset of the nodes only and at different times. We assume that at most $f = \lfloor (n-1)/3 \rfloor$ nodes are faulty, and refer to the set of correct nodes as $C \subseteq V$.

Nodes in C communicate by broadcasts. If $v \in C$ broadcasts at time t_v, any other correct node $w \in C$ has received and processed the respective pulse at some time $t_{wv} \in [t_v + d - U, t_v + d]$, where d is the *maximum delay* and U is the *delay uncertainty*. For faulty senders in $V \setminus C$, such restrictions are irrelevant, as they may deviate from the protocol in an arbitrary way, i.e., send pulses at arbitrary times and independently to different receivers.

A correct node measures the time of arrival of other nodes' pulses relative to the time of arrival of its own pulse of the same round (cf. Algorithm 1). This is done by looping the broadcast signal back and using time-to-digital converters (TDCs) to determine the respective time difference. (cf. Section III-B). We assume a one-sided[1] worst-case measurement error of our TDCs when comparing signals arriving at times t and t' that fulfills $e(|t - t'|) = G + \nu|t - t'|$, where G is the *granularity* of the TDC (i.e., its discretization error) and $\nu \ll 1$ is the maximum relative deviation of the frequency of the TDC's time reference from its nominal frequency.

A node v has no access to real-time, but only to its local clock $H_v \colon \mathbb{R}_0^+ \to \mathbb{R}_0^+$, where $H_v(t)$ is the local clock value at real-time t. For the purpose of a straightforward presentation of the algorithm, we assume that

$$\forall t, t' \in \mathbb{R}_0^+, t > t' \colon t - t' \leq H_v(t) - H_v(t') \leq \vartheta(t - t'),$$

where $\vartheta > 1$ is a constant close to 1, describing the frequency offset uncertainty of the local clock[2]. For the sake of simplicity, we set $\nu = \vartheta - 1$ in the following, i.e., the clock source of a node and its TDCs have the same worst-case phase drift. We assume that $H_v(0) \in [0, F)$ for all $v \in C$, where F is determined by the precision of the booting process. For better readability, we denote real-times with t and local times with τ, with respective indices.

[1]By this we mean that we specify the length of the interval around the true value the measurements may come from.

[2]Naturally, in practice H_v will be discrete and bounded. However, H_v is merely used to control the local logic of the algorithm, rendering this inconsequential to our considerations.

Algorithm 1: Synchronization algorithm, code for node v

```
1  // H_w(0) ∈ [0, F) for all w ∈ V
2  wait until time t_v(0) with H_v(t_v(0)) = F;
3  foreach round r ∈ ℕ do
4      start listening for messages;
5      wait for τ_1 local time;        // all nodes are in round r
6      broadcast clock pulse to all nodes (including self);
7      wait for τ_2 local time;        // all messages arrived
8      for each node w ∈ V do
9          τ_vw := H_v(t_vw), with reception time t_vw of first
            message from w (τ_vw := ∞ if no message
            received from w);
10     T_v := {τ_vw − τ_vv | w ∈ V} (as multiset);
11     let T_v^k denote the k^th smallest element of T_v;
12     δ_v ← (T_v^{f+1} + T_v^{n−f}) / 2;        // clock correction
13     wait until time t_v(r) with
        H_v(t_v(r)) = H_v(t_v(r − 1)) + T_R − δ_v;// round ends
```

B. Basic Algorithm

Algorithm 1 gives the pseudocode of the algorithm. Each node $v \in V$ starts round $r \in \mathbb{N}$ at time $t_v(r-1)$, where $t_v(0) = F$, and ends round r at $t_v(r)$. To fully specify the algorithm, we need to determine τ_1, τ_2 and T_R. The following conditions are sufficient for the algorithm to work as intended.

$$\begin{aligned} \tau_1 &\geq \vartheta F \\ \tau_2 &\geq \vartheta(F + \tau_1 + d) \\ T_R &\geq \vartheta(\tau_1 + F + U) + \tau_2 + t_{\text{comp}} + G, \end{aligned}$$

where t_{comp} is the time required to compute and apply the phase correction. It is desirable to keep the round length T_R small, unless one seeks to lower the communication frequency. Since any values satisfying these inequalities are acceptable, one may always round up to the next integer multiple of the cycle time of the oscillators controlling the logic, i.e., no constraints on oscillator frequencies are needed. In the full paper, we prove that the minimal feasible choices result in a steady-state skew of $E \approx 4(U + G)$ for $\vartheta - 1 \ll 1$. More detailed calculations show that the algorithm can handle frequency offsets of up to $\vartheta - 1 \approx 1\%$ without dramatic impact on E.

C. Node Recovery

So far we assumed that nodes are initially synchronized and maintain this property. We now address the case that $n - f$ nodes are synchronized, but an additional node is out-of-sync (possibly after a transient fault) and attempts to resynchronize. The modification to the algorithm is extremely simple: whenever a node receives fewer than $n - f$ signals while listening for them in a given round, it will cut this round short. Thus, it quickly catches up with the main field.

In the full paper, we prove that under slightly more conservative constraints on τ_1, τ_2, and T_R, this results in resynchronization in a constant number of rounds. We stress, however, that this requires that an implementation (i) makes sure that indeed a node starts executing the next round within a time bounded by the maximum round duration, regardless of the content of its volatile memory, and (ii) it does not introduce any variables whose values are carried over to the next round.

III. Implementation and Experiments

We implemented the algorithm on four Cyclone IV FPGA development boards. We designed a simple additional board to carry the clock oscillator for the FPGA and the connectors for the coaxial cables between the nodes. In order to allow corrections of the pulse position with sub-clock cycle granularity, we apply phase shifts using a voltage controlled crystal oscillator (VCXO), which supplies the reference frequency for the PLL within the FPGA. The nodes are connected to each other using coaxial cables of the same length (ca. 30 cm), one for each pair of nodes and direction. The FPGA implements four TDCs (see below) to measure the timing of the incoming pulses, implements the logic of the algorithm, and controls the VCXO. An additional pulse output is available for measurements.

Due to limitations of the development board, pulses use 3.3V LVCMOS signaling. The resulting reflections slightly add to the measurement uncertainties. Furthermore, the FPGA development board only provides two pins for ground connection. This resulted in an involuntary test of the algorithm's fault-tolerance properties: having many high-speed signals over the same connector, the setup suffered from significant ground bounce of up to 200 mV between the ground potentials of the development board and the interface board; this caused one of the nodes to lose several clock ticks during our experiments.

A. Cycle Structure and Phase Control

We clock the FPGA with 130 MHz derived from a 20 MHz VCXO on our interface board. As discussed above, to achieve sub-cycle length (i.e. smaller than 7.7 ns) corrections of the phase of the pulse, we control the reference oscillator's output frequency. We implemented this using a 16-bit, 1 Msps DAC with SPI interface. This design choice imposed two important restrictions on our implementation. First, the oscillator's modulation bandwidth of about 10 kHz imposes a lower bound on the round length, as we need to allow for sufficient time for the oscillator to respond to a changed control input. Therefore, we chose a fairly large round length of $T_R = 50\,\mu$s, of which $40\,\mu$s are allocated for shifting the clock phase.

Second, the tuning range of the oscillator is roughly 10 ppm, limiting the phase correction per round to ≈ 400 ps. This is smaller than the duration of clock cycle of the FPGA (≈ 7.7 ns), preventing a simple implementation of larger phase shifts by enabling to adjust the (integral) number of clock cycles per round. Fortunately, the convergence analysis shows that the algorithm achieves the same steady-state error with this limitation on phase corrections. However, the number of rounds required for recovering nodes to resynchronize is much larger; with a frequency correction of at most 10 ppm, this takes up to about 10^5 rounds, yielding a worst-case bound on the time to recover in the order of seconds.

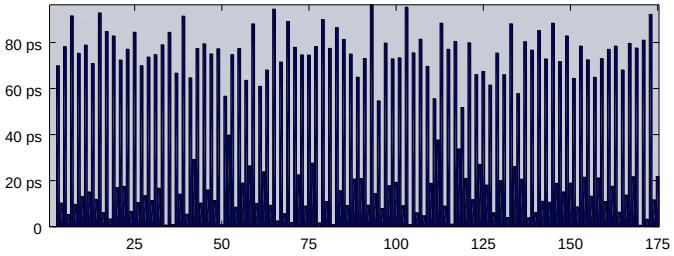

Fig. 1. Histogram of the encoded TDL output values during offline calibration. These values correspond to the bin sizes of the delay line.

B. Time-to-Digital Converter

We employ standard techniques in our TDC design (like in, e.g., [25]). Our implementation is based on the tdc-core by the White Rabbit Project [26] (details are presented in [27] and [28]), which we adapted with minimal changes to Cyclone IV. The TDC uses an adder carry chain as delay line and a coarse counter for the measurements. Additionally, there is a ring oscillator to measure and compensate for voltage and temperature effects during operation (see [28] for details). We used the internal startup calibration system to get an estimate on the bin size of the TDC and thus its precision (see Figure 1). The largest observed bin size is 140 ps. Estimating a calibration error of up to 20 ps, this yields a single-shot precision of $G \leq 160$ ps.

C. Parameter Extraction

The performance-critical parameters from the setup are:

- As discussed above, we have $G \leq 160$ ps for the TDC.
- We calibrated the differences in wire delays on the development and interface boards using the TDCs. This results in an uncertainty of $U \leq G + 40\,\text{ps} \leq 200$ ps, where 40 ps is an estimated upper bound on the delay variations in equivalent paths between the TDCs.
- We measured a frequency deviation between one pair of oscillators of < 1.5 ppm. The manufacturer lists a typical frequency deviation including initial deviation and over temperature range of typical 3 ppm, i.e., $\vartheta - 1 \approx 3 \cdot 10^{-6}$.

Inserting these values into the bound obtained from the analysis, the estimated worst-case clock skew without faults is $2(G + U) + (\vartheta - 1)T_R = 870$ ps, where $T_R = 50\,\mu$s is the nominal duration of a round. With faults, this becomes $4(G + U) + 2(\vartheta - 1)T_R = 1740$ ps.

D. Experimental Setup and Results

We fully connected the nodes using cables of length 30 cm. The physical diameter of the whole setup is approximately 50 cm, cf. Figure 2. Measurements are taken by a WaveCrest DTS-2075, which has a single-shot accuracy of ± 25 ps and calibrated the input port skew to achieve better than 2 ps accuracy. To rule out any spurious effects from the instrument, we used two Stanford Research SR620 to verify these bounds.

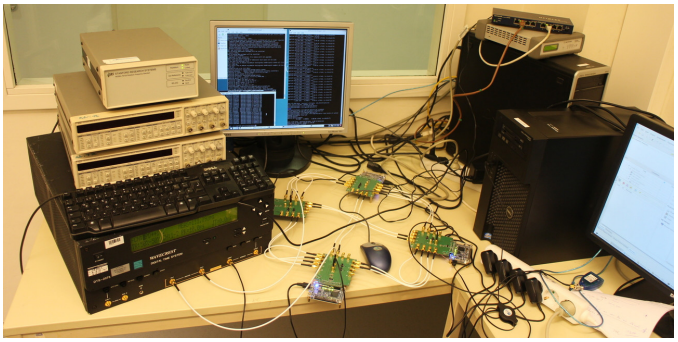

Fig. 2. Experimental setup. Left: measurement instruments. Center: nodes with FPGA and interface boards, and a stray mouse. Right: recording PC.

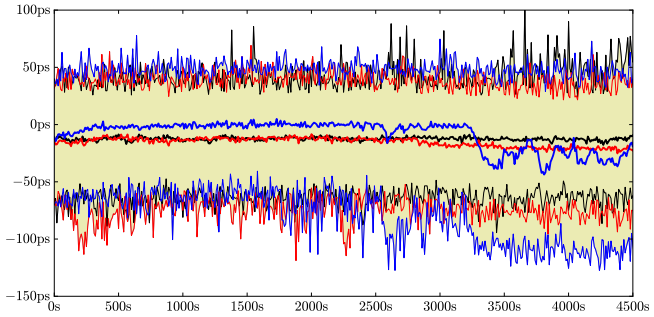

Fig. 3. Long-term evolution of the clock skew of three nodes against the same reference node over a period of an hour, measured sequentially. The thick lines depict the average clock skew over 10 s, the light yellow colored fill with the thin lines depict the minimum and maximum in the same interval.

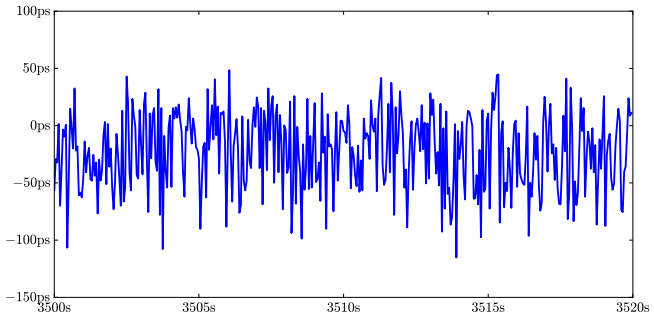

Fig. 4. Short-term behavior of the clock skew of the "blue" node vs. the reference node from Figure 3 over an arbitrarily selected period of 20 s.

1) Skew Measurements: We measured the skew between all pairs of nodes sequentially for at least one hour each, which corresponds to $7.2 \cdot 10^7$ rounds. We observed a maximum clock skew of 180 ps between correctly working nodes. Figure 4 showcases the short-term behavior of the clock skew.

To test the behavior under worst-case faults, we modified one node with the aim to maximize the skew of the remaining nodes. The analysis indicates that the maximum impact of faults is achieved when faulty nodes send early pulses to nodes that are already ahead and none to those that lag behind. After implementing this behavior, we observed an increase in the maximum skew to 270 ps.

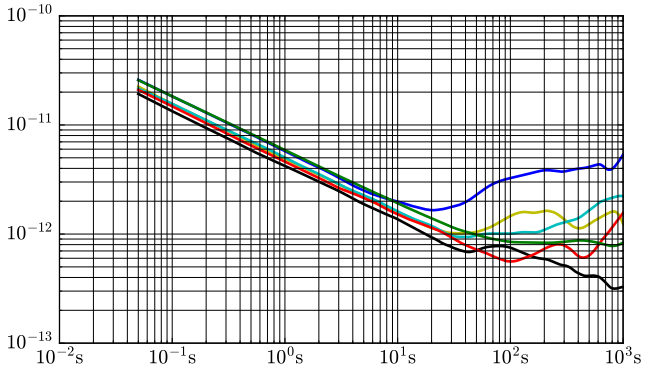

Fig. 5. TDEV between all pairs of nodes, measured sequentially. The colors of pairs match those from Figure 3.

2) Resynchronization: To verify that nodes resynchronize after transient fault conditions, we modified one node to drop out using a manually actuated switch. Triggering the switch every couple of seconds results in randomly distributed restarting times with respect to the clock phase of the correctly synchronized nodes. In 20 measurements, we observed the expected stabilization behavior. In accordance with our earlier discussion, recovery took up to 7 s for our implementation.

3) Time and Frequency Stability: We analyzed the statistical time and frequency stability (cf. [29]) of the system in long term measurements. The TDEV plots (Figure 5) are measured between pairs of nodes of the synchronized system. As it can be seen, the noise of the system behaves mostly like white phase noise up to a τ of approximately 10 s.

The results significantly exceed our expectations in the range below 10 s. While the algorithm inherently suppresses effects from outliers, as it drops the largest and smallest measurement value in each round, and subsequently averages between the remaining two, this merely suggests improvements of factor 3 to 5 over a free-running oscillator (TDEV of $\sim 1 \cdot 10^{-9}$ s @ 1 s). In contrast, uncertainties of parts in 10^{-12} s are already reached above 1 s for the correctly working nodes. These are quite astonishing stability values, especially in light of the crude setup resulting from the employed affordably priced hardware.

As the primary application of the clock synchronization system is to serve as a clock source for circuits, we also analyzed the absolute frequency fluctuations against a Stanford Research FS275 rubidium frequency standard. We show two ADEV plots, see Figure 6. The first compares a free-running node to the rubidium, i.e., the algorithm is deactivated in order to measure the raw performance of the oscillator. The second depicts the behavior of the same node, but now synchronized to the other nodes, via the algorithm. We observe that the long term stability over $\tau > 10$ s is approximately the same. This is expected, as the long-term behavior is dominated by the temperature-induced frequency fluctuation of the used oscillators. Below a τ of 1 s, however, the stability of the synchronized system is higher than the one of the free running node, as the noise of the oscillators is averaged by the synchronization of the nodes. Surprisingly, we gain almost an

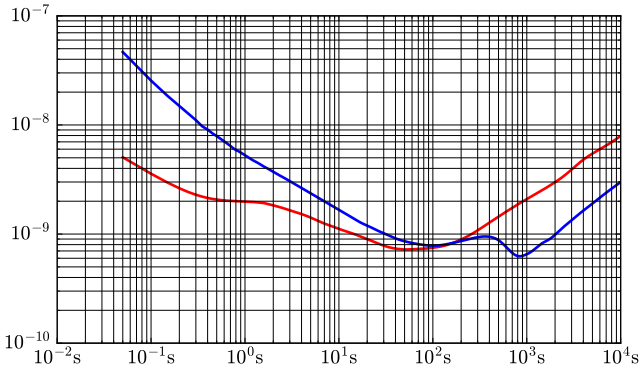

Fig. 6. ADEV between a node and a rubidium frequency standard. The top (blue) curve compares to the free-running oscillator, with the algorithm disabled. The bottom (red) curve is the same node with the algorithm enabled and the system fully synchronized. The temperature effects beyond $\tau = 100\,\mathrm{s}$ differ because the traces were recorded on subsequent days with different weather conditions and thus different heating patterns of the building.

order of magnitude in stability in the short-term range, again significantly exceeding our predictions.

IV. Conclusion

We presented a sub-nanosecond skew FPGA prototype implementation of a fault-tolerant clock synchronization algorithm. The algorithm can sustain $f < n/3$ Byzantine faults and allows for resynchronization of nodes after transient faults. Our experiments demonstrate excellent performance, featuring 270 ps clock skew despite faults and an order of magnitude improvement in clock stability in the sub-second range. Therefore, we consider the approach a promising candidate for reliable clock generation in VLSI circuits. Our current FPGA-based implementation is clearly limited by the accuracy of the TDCs. We are currently working on a hybrid solution with an ASIC-based TDC, with the prospect of achieving skews below 100 ps. Moreover, our findings concerning clock stability enhancement for $\tau < 1\,\mathrm{s}$ warrant exploration of leveraging the technique in communication applications, for which short-term clock stability is crucial.

Acknowledgments

We thank Ulrich Schmid and Andreas Steininger for kindly providing the lab space and the equipment for the measurements and Magnus Danielson for many fruitful discussions and checking our measurements for potential issues.

References

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

[2] W.-H. Liu, Y.-L. Li, and H.-C. Chen, "Minimizing clock latency range in robust clock tree synthesis," in *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, ser. ASPDAC '10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 389–394.

[3] G. Shamanna, N. Kurd, J. Douglas, and M. Morrise, "Scalable, sub-1W, sub-10ps Clock Skew, Global Clock Distribution Architecture for Intel® Core™ i7/i5/i3 Microprocessors," in *Proc. Symposium on VLSI Circuits (VLSIC)*, 2010, pp. 83–84.

[4] X.-W. Shih and Y.-W. Chang, "Fast timing-model independent buffered clock-tree synthesis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 9, pp. 1393–1404, Sept 2012.

[5] D. M. Chapiro, "Globally-Asynchronous Locally-Synchronous Systems," Ph.D. dissertation, Stanford University, 1984.

[6] P. Teehan, M. Greenstreet, and G. Lemieux, "A survey and taxonomy of gals design styles," *Design Test of Computers, IEEE*, vol. 24, no. 5, pp. 418–428, Sept 2007.

[7] Y. Semiat and R. Ginosar, "Timing Measurements of Synchronization Circuits," in *Proc. Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2003, pp. 68–77.

[8] T. Polzer, T. Handl, and A. Steininger, "A metastability-free multi-synchronous communication scheme for socs," in *Stabilization, Safety, and Security of Distributed Systems*, ser. Lecture Notes in Computer Science, R. Guerraoui and F. Petit, Eds. Springer Berlin Heidelberg, 2009, vol. 5873, pp. 578–592.

[9] J. L. Welch and N. A. Lynch, "A New Fault-Tolerant Algorithm for Clock Synchronization," *Information and Computation*, vol. 77, no. 1, pp. 1–36, 1988.

[10] P. Khanchandani, "Accurate and Robust Clock Synchronization," Master's thesis, University of Saarbrücken, 2015.

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

[12] D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl, "Reaching Approximate Agreement in the Presence of Faults," *Journal of the ACM*, vol. 33, pp. 499–516, 1986.

[13] F. B. Schneider, "Understanding protocols for byzantine clock synchronization," Ithaca, NY, USA, Tech. Rep., 1987.

[14] T. K. Srikanth and S. Toueg, "Optimal Clock Synchronization," *Journal of the ACM*, vol. 34, no. 3, pp. 626–645, 1987.

[15] H. Kopetz and G. Bauer, "The Time-Triggered Architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[16] R. Belschner, J. Berwanger, F. Bogenberger, C. Ebner, H. Eisele, B. Elend, T. Forest, T. Führer, P. Fuhrmann, F. Hartwich *et al.*, "Flexray communication protocol," Oct. 22 2003, eP Patent App. EP20,020,008,171. [Online]. Available: https://www.google.com/patents/EP1355456A1?cl=en

[17] H. Pfeifer, D. Schwier, and F. W. Von Henke, "Formal verification for time-triggered clock synchronization," in *Dependable Computing for Critical Applications 7, 1999*. IEEE, 1999, pp. 207–226.

[18] M. Függer, E. Armengaud, and A. Steininger, "Safely Stimulating the Clock Synchronization Algorithm in Time-Triggered Systems - a Combined Formal & Experimental Approach," *IEEE Trans. Industrial Informatics*, vol. 5, no. 2, pp. 132–146, 2009.

[19] M. Függer and U. Schmid, "Reconciling Fault-Tolerant Distributed Computing and Systems-on-Chip," *Distributed Computing*, vol. 24, no. 6, pp. 323–355, 2012.

[20] M. Ferringer, G. Fuchs, A. Steininger, and G. Kempf, "VLSI Implementation of a Fault-Tolerant Distributed Clock Generation," *IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT2006)*, Oct. 2006.

[21] D. Dolev, M. Fuegger, C. Lenzen, M. Posch, U. Schmid, and A. Steininger, "Rigorously Modeling Self-Stabilizing Fault-Tolerant Circuits: An Ultra-Robust Clocking Scheme for Systems-on-Chip," *Journal of Computer and System Sciences*, vol. 80, no. 4, pp. 860–900, 2014.

[22] D. Dolev, M. Fuegger, C. Lenzen, and U. Schmid, "**F**ault-tolerant **A**lgorithms for **T**ick-generation in **A**synchronous **L**ogic: Robust Pulse Generation," *Journal of the ACM*, vol. 61, no. 5, pp. 30:1–30:74, 2014.

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

[24] H. Kopetz, "Fault containment and error detection in the time-triggered architecture," in *Autonomous Decentralized Systems, 2003. ISADS 2003. The Sixth International Symposium on*, April 2003, pp. 139–146.

[25] J. Song, Q. An, and S. Liu, "A high-resolution time-to-digital converter implemented in field-programmable-gate-arrays," *Nuclear Science, IEEE Transactions on*, vol. 53, no. 1, pp. 236–241, Feb 2006.

[26] "OHWR TDC core," http://www.ohwr.org/projects/tdc-core/wiki.

[27] S. Bourdeauducq, "A 26 ps RMS time-to-digital converter core for Spartan-6 FPGAs," March 2013. [Online]. Available: http://arxiv.org/pdf/1303.6840v1.pdf

[28] ——, "Time to digital converter core for Spartan-6 fpgas," November 2011. [Online]. Available: http://www.ohwr.org/documents/98

[29] D. Sullivan, D. Allan, D. Howe, and F. Walls, "Characterization of clocks and oscillators," National Institute of Standards and Technology, Technical Note 1337, March 1990.