Fault-tolerant High-Performance Clock Distribution

Attila Kinali, Christoph Lenzen
Max Planck Institute for Informatics
Campus E1 4
Saarbrücken, Germany
{adogan,clenzen}@mpi-inf.mpg.de

Martin Perner
TU Wien
Treitlstrasse 3
Vienna, Austria
mperner@ecs.tuwien.ac.at

ABSTRACT

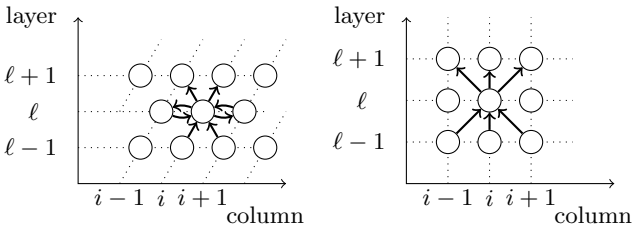
We consider low-degree clock distribution grids that are resilient to failures of a fraction of its nodes and have the ability to recover from an unbounded number of transient faults. In [1] Dolev et al. present such a network. However, this approach's synchronization quality deteriorates significantly in face of faults. We provide a novel solution with reduced node degree, for which faults affect synchronization only in the order of the *differences* in delays. We evaluate the performance of our approach in simulations, and implement the design in a transistor cell. The cell is modeled in SPICE, and measured under different parameter settings.

1. INTRODUCTION

The synchronous hardware design paradigm utilizes a global clock that is distributed to all components of the design. This allows seamless communication between every part of the design. While this approach is predominantly used and well-understood, it has its limits: The distribution of the global clock requires power-intensive clock tree design to achieve the required synchronization of the clock edges, called *clock skew*. Scaling clock trees, yet keeping a small skew, is impractical for larger designs. However, if the clock skew becomes too large, timing constrains will be violated and metastability can manifest. Purely asynchronous designs, on the other hand, remove the need for a global clock tree, which constitutes a single point of failure. The downside of this approach is increased complexity in application design, especially when real-time guarantees need to be met.

Globally Asynchronous Locally Synchronous (GALS) design offers a compromise avoiding the need for a global clock: It uses multiple clock trees, each driving just a few components. The challenges here are (i) to provide synchronization guarantees between the clock trees, (ii) to ensure sufficient reliability of the system, yet (iii) to keep the resources overhead for synchronization under control.

A promising approach is to provide a synchronization basis for the clock domains by using a low-degree clock distribution grid. Nodes connected to a few neighbors by wires



(a) Grid topology of HEX. (b) Grid topology of TRIX.

Figure 1: Overview of node (ℓ, i) and its incident links in the different grid topologies. Column coordinates are modulo W and layer coordinates (rows) are between 0 and H.

of roughly the same length are sufficient to handle faults of some of the nodes in the grid, while maintaining a reasonably synchronized clock signal. The advantages of such designs are (i) better scalability compared to clock trees, (ii) less stringent layout and energy requirements, and (iii) fault-tolerance. The clock domains connected to the (few) faulty nodes of the grid can be handled by means of adding fault-tolerance to the higher system layers; as we consider large systems, this is necessary anyways.

HEX Clock Distribution and its Downsides. In [1], a clock distribution grid, HEX, along these lines was proposed, see Figure 1a. As can be seen, it has a regular structure with node in- and out-degree of four. A node forwards the clock signal as soon as it received the clock signal from two adjacent neighbors. The assumption is that the signal propagation over the wire plus the processing time of the node takes between d^- and d^+, where $\varepsilon := d^+ - d^-$. Due to the direction of the links, the clock signal spreads from *layer* (i.e., "row") 0, which is the source of the clock signal, to the higher layers, at a speed of roughly one layer per $(d^+ + d^-)/2$ time. Further, the rule to wait for two adjacent neighbors ensures that an isolated faulty node (i) cannot emanate a spurious clock signal that propagates through the grid and (ii) cannot block propagation of a correct clock signal. In combination with a careful implementation of the nodes, involving timed decay of memorized states of the incoming link ports of a node, the systems as a whole *self-stabilizes* from an arbitrary state of the nodes: the grid re-establishes correct forwarding of clock signals after an arbitrary number of transient faults, in spite of the ongoing presence of at most one faulty node in each neighborhood.

Unfortunately, HEX has two shortcomings. First, the

ACM ISBN .
DOI:

HEX nodes are fairly involved. They have separate memory for each link, timeouts resetting the respective memory cells after a while, and need to combine the four resulting signals into the decision when to trigger the next clock edge locally. Second, a faulty node may delay the clock propagation of its upper-layer neighbors by at least d^- time. As in practice $\varepsilon \ll d^-$, this results in larger skews than necessary. The simulation results in [1] indicate better behavior, i.e., skews in the order of ε, in the absence of faults. Ideally, one would like to achieve this also in presence of faults.

Our Contribution. We conducted explorative simulations to find alternative topologies which do not have these shortcomings. The simulations revealed that the synchronization of the inputs significantly affect the choice of the topology. In this paper, we assume well-synchronized input to the distribution grid, arguing that the clock grid neither can nor should "fix" synchronization issues of the clock sources. Under this constraint, the architecture *TRIX*, given in Figure 1b, delivers excellent performance despite a compellingly simple interconnection topology, significantly outperforming HEX.

This is demonstrated by the following further individual contributions:

1. We show that for the TRIX topology a much simpler firing rule suffices to achieve the same fault-tolerance properties (tolerance of up to one faulty in-neighbor per node and self-stabilization) as HEX.
2. We provide a highly efficient transistor-level implementation of this firing rule. This implementation provably guarantees that faulty nodes cannot cause metastability of correct nodes, regardless of the input signals they provide to correct nodes.
3. We characterize the behavior of this implementation through extensive SPICE simulations, yielding a processing delay variation of nodes (depending on arrival times of inputs) below 10 ps for 65 nm technology.
4. Based on this characterization, we simulate clock wave propagation through a TRIX grid, representing temperature and other noise sources by randomizing propagation times using truncated normal distributions of standard deviation $\sigma = 1$ ps (both on wires and firing gates). We observed almost no degradation of skew even under faulty conditions.

In our simulations, we observed firing gate delays of 35 ps to 43 ps. This implies that $d^- > 35$ ps, where d^- also comprises the highly layout-dependent wire delays. This suggests improvements over HEX of at least an order of magnitude. More importantly, the obtained skews *are hardly degraded by faults* and are comparable to those of clock trees, rendering the proposed grid a viable candidate for scalable clock distribution.

Related Work. Fault-tolerance, especially in aerospace applications, has been an important factor during design. Most fault models assume a single error where the faulty device either remains silent or can be identified and masked. In recent years, the focus has shifted to Byzantine faults, dropping the assumption that faults can either be detected or masked [2]; our approach can tolerate multiple such faults.

Clock trees and meshes have been the prevalent clocking scheme in chip design. They represent an inherent single point of failure, especially in aerospace applications where single-event-transients (SET) due to radiation are common. There have been several papers analyzing the impact of SET on clock trees in recent years [3, 4, 5, 6, 7, 8]. Various chip designs were proposed to harden the system to SET on the clock tree, either addressing the clock tree directly [9] or improving the tolerance of the latches [10].

Recent designs using 3D chip design techniques face similar difficulties due to reliability issues with through silicon vias (TSV) [11]. Analysis of faults in TSV and synthesizing clock trees that are resilient to these faults is a complex problem and leads to large computing times, especially if redundant TSV's cannot be supplied locally [12, 13].

A number of proposals avoid clock trees. Fairbanks proposes a highly accurate distributed oscillator consisting of simple elements in a grid [14], but does not examine its reliability under faults. Byzantine fault-tolerant [15, 16] and self-stabilizing Byzantine fault-tolerant solutions [17, 18] exist, but require impractical full connectivity; such algorithms could provide a fault-tolerant clock source as input to our clock grid, however.

The most closely related work, HEX [1], exhibits the same fault-tolerance properties and comparable connectivity, but, as discussed above, much worse skews.

Paper Organization. In Section 2, we describe our clock distribution grid. A node for the chosen topology is then implemented as a transistor cell in Section 3, where we also discuss the fault-tolerance properties of our approach. Section 4 provides simulation results. Finally, Section 5 concludes the paper.

2. TOPOLOGY & PULSE FORWARDING

We consider nodes executing a pulse generation and forwarding system, which communicate by sending pulses over links of the grid. The node set is $\{(\ell, i) \in [H+1] \times [W]\}$. Herein, $[H+1] := \{0, \dots, H\}$ denotes the row index set, referred to as *layers*, and $[W] := \{0, \dots, W-1\}$ the column index set. Each node (ℓ, i) on layer $\ell < H$ has outgoing links to the three nodes $(\ell+1, i-1)$, $(\ell+1, i)$, and $(\ell+1, i+1)$, where we take all column indices modulo W. Figure 1b shows the resulting topology from the perspective of a node.[1] We stress that, unlike the grid from [1], we do not rely on links between nodes of the same layer, cf. Figure 1a.

For each clock pulse, each (non-faulty) node in layer $\ell \neq 0$ executes a simple firing rule: as soon as it received pulses from any two of its neighbors on layer $\ell - 1$, it broadcasts a pulse to its neighbors on layer $\ell + 1$. We show that our implementation of this firing rule, given in Section 3, guarantees that even if one in-neighbor of a node is faulty, it fires within the time interval given by the pulses from its other two neighbors. This ensures that the grid can tolerate up to one in-neighbor of a node failing *in an arbitrary manner,* including sending deteriorated signals; as we discuss in

[1] We choose this highly regular topology for obtaining a better understanding of the effects of clock propagation on the skew and easier comparison to [1]. In practice, layers would be concentrical rings around the clock source, resulting in the "width" of the grid scaling with the layer number. This will have no negative impact on synchronization quality, as this means to "distribute" the skew from previous layers over more nodes, and we are interested in minimizing the skew between adjacent nodes.

Section 3, our firing rule implementation masks any internal metastability that could be caused by such signals. Nodes on layer 0 represent the clock source; for each pulse, the non-faulty nodes on layer 0 fire at roughly the same time by assumption.

After a pulse, a node temporarily disables its firing mechanism (to avoid firing multiple times for a single pulse), waits until sufficient time has passed for correct in-neighbors pulses to have passed, and then enables it again. This does not require accurate timing, unless one aims for a very high frequency of pulses. Note that one is free to drive a local phase-locked loop (PLL) at each node using the pulses, so a high pulse frequency is not necessary for achieving a high operational frequency of the system; accordingly, we do not analyze the respective timing constraints in detail, but simply assume that the nodes on layer 0 wait for sufficiently long before sending the next pulse. However, this mechanism is relevant to *self-stabilization* of the system, i.e., recovery from arbitrary transient faults. In Section 3, we also argue why the combination of our implementation of the firing rule and the simple employed topology guarantee self-stabilization. As TRIX does not rely on communication between nodes in the same layer, straightforward induction over the layers shows this highly desirable property.

Following [1], we measure the quality of the synchronization in terms of the *skew* between physically close-by nodes, that is, the difference in their firing time (for the same pulse). The rationale is that close-by nodes typically have the most stringent timing requirements Accordingly, we do not only take into consideration neighboring nodes in terms of the communication topology, but also the skew between nodes (ℓ, i) and $(\ell, i+1)$, i.e., adjacent nodes on the same layer. Once the system has stabilized, each pulse causes each (non-faulty) node to fire exactly once, meaning that we can determine the skew for an individual pulse, by comparing the firing times as discussed above.

DEFINITION 1 (SKEW). *For a given pulse, denote by $t_{\ell,i}$ the firing time of non-faulty node (ℓ, i). For $\ell \in [H+1]$, the* intra-layer skew *of layer ℓ is $\sigma_\ell := \max_{i\in[W]}\{|t_{\ell,i} - t_{\ell,i+1}|\}$. For $\ell \in [H+1] \setminus \{0\}$, the* inter-layer skew *of layer ℓ is*

$$\hat{\sigma}_\ell := \max_{i\in[W]}\{t_{\ell,i} - t_{\ell-1,i-1}, t_{\ell,i} - t_{\ell-1,i}, t_{\ell,i} - t_{\ell-1,i+1}\}.$$

Note that the inter-layer skew is biased due to the pulse propagation delay. As one can shift the local clock relative to the pulse arrival times by a fixed amount, this is not an issue if we can predict this skew accurately, i.e., it is crucial that the *range* of the inter-layer skew is kept small. In contrast, the intra-layer skew is unbiased and symmetrical.

3. FIRING LOGIC IMPLEMENTATION

The transistor-level implementation of the firing logic is designed with various goals in mind: (i) minimizing the delay variation (as it is the principal source of skew), (ii) repeatedly producing well-shaped pulses, (iii) being insensitive to erroneous input from a faulty node, and (iv) allowing for recovery from transient faults.

Achieving (i), (ii), and (iv) in conjunction with (iii) becomes challenging, but a customized implementation of the logic in the critical path can meet all design goals using standard components only. It is depicted in Figure 2, where

- A, B, and C are the pulse inputs from the in-neighbors,
- A', B', and C' are the outputs for the out-neighbors,
- L is the local output driving the node's clock tree, and
- E is the enable signal, which is briefly set to low after a pulse was forwarded.

Here, E is set to low, by the logic that is not in the critical path, in response to a pulse being generated. After a sufficient timeout E is set back to high. The circuit assumes that two of the inputs are provided by non-faulty nodes. That means that "clean" pulses are received with a duration that is long enough for the circuit to generate a pulse and short enough for the timeout of the enable signal to avoid generation of spurious pulses.

Before analyzing the behavior under faults, let us walk through a typical pulse cycle without faults. The system is initialized by setting E low. This forces the storage loop high and thus accordingly the outputs A', B', C', and L are stable low. As all nodes are initialized the same way, the inputs A, B and C are also low.

The pulse cycle starts by setting E high. As the circuit contains an incomplete P-stack, the internal node would be floating, if not for the storage loop. Observe that, since E is high, the three parallel paths controlled by A, B, and C connect the internal node to ground if at least two of these signals are high. As soon as any two of the inputs go high, the storage loop is forced low and the outputs go high. Note that once the storage loop stabilized, it does not matter if any of the input signals returns to low again. Thus, having output pulses with (roughly) fixed width can be achieved by setting E to low for a fixed time after L went high; as the timing here is uncritical, we do not discuss this part of the implementation. As the width of the pulses from the previous layer is fixed, too, the input signals return to low before or shortly after the node forced its own output to low via E. Hence, after E was low for some (short) fixed amount of time, it is safe to return it to high again. Thus, the cycle is complete and the node is ready for the next pulse.

Fault-tolerance. The above discussion assumed that all three inputs are connected to non-faulty nodes. However, a minor tweak is sufficient to completely mask a faulty node at one of the inputs. First, assume that the initial state of the circuit is as above, except that one of the inputs, say A, provides an arbitrary (faulty) signal. Clearly, this does not close a circuit pulling the internal node down, as B and C are still low. Hence, the storage loop is unaffected and no pulse is generated before a pulse signal from either B or C (or both) arrives.

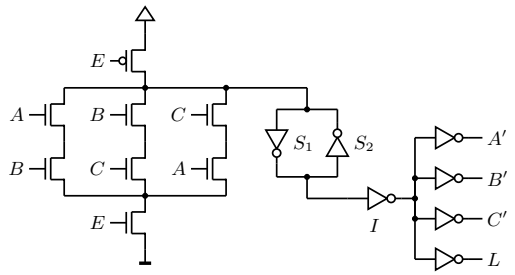


Figure 2: Firing circuit. A, B, and C are the inputs, A', B', C', and L the outputs, and E is a locally generated enable signal shaping the pulse. I is a high-threshold inverter, masking metastability of the storage loop consisting of S_1 and S_2, which may be caused by degenerate input from a faulty node.

Once one of B or C arrives and the storage loop "captured" the pulse, i.e., the outputs of S_1 and S_2 stabilized to low and high, respectively, the storage loop's state is independent of inputs A, B, and C. However, the "capturing" may fail if the signal at A is not well-shaped or goes low again before the third input goes high, possibly resulting in the storage loop becoming metastable. To cover this case, observe that if the output of S_2 moves notably above the threshold voltage of S_1, the storage loop is already stabilizing to having low output at S_1, which then is maintained regardless of whether there is a connection to ground (until E switches to low). Thus, by choosing a higher threshold for inverter I than for S_2, we mask any such effects at the output: When the output voltage of S_1 is sufficiently high for I to start to switch, the storage loop will be driven into the stable state, thus ensuring a clean pulse at the outputs A', B', C', and L. We conclude that the pulse is correctly forwarded, both with respect to timing and signal shape. In particular, the logic not in the critical path is driven by a clean signal, implying that the node's logic is not affected by bad input from a faulty node.

Self-stabilization. Self-stabilization means that the system recovers from arbitrary transient faults (particle hits, voltage droop, etc.), which result in arbitrary node states. In other words, at some point in time the transient faults cease, and non-faulty nodes work correctly again. However, their internal state may be corrupted and/or inconsistent with that of other nodes.

For lack of space, we only sketch how TRIX can be made self-stabilizing from a high-level perspective here. First, observe that we can reason inductively over the layers, where we assume that the input to the clock grid (i.e., layer 0), operates correctly. Thus, it suffices to show that a node on layer $\ell > 0$ starts to forward pulses correctly again after the nodes on previous layers did, i.e., the induction hypothesis is that at least two of the three inputs of a node are well-behaved as above (the third may be from a faulty node).

An important point to note is that after transient faults, the internal logic of the node may be corrupted and even suffer from metastability. However, metastability is, by definition, a transient state that is extremely likely to subside within a few nanoseconds. Moreover, the control logic (including the parts not in the critical path) is very simple, meaning that standard design practices can easily ensure that the state-holding components are forced out of any invalid (i.e., inaccessible under fault-free operation) node state. Thus, the node ends up in an internal state consistent with some (unspecified) part of the standard pulse cycle described above.

From there, we can argue that the node will soon capture a pulse from its correct in-neighbors. This pulse will cause that, after the respective timeouts, E is set to low, re-initializing the node; at the latest in the following pulse, the node will pulse in correct phase relation with its preceding layer. As this applies to all correct nodes on layer ℓ, we can argue that within less than three "clean" pulses from layer $\ell - 1$, layer ℓ stabilizes. This guarantees that the system recovers from transient faults within a number of pulses proportional to L; we remark that, in practice, stabilization typically occurs within a few pulses.

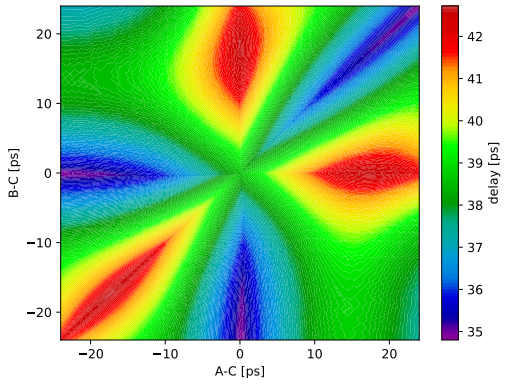


Figure 3: Delay of the circuit from Figure 2 as function of the relative times of the input transitions. It increases if two pulses together arrive early and the third one is late. It decreases if one pulse is early and the other two together arrive late.

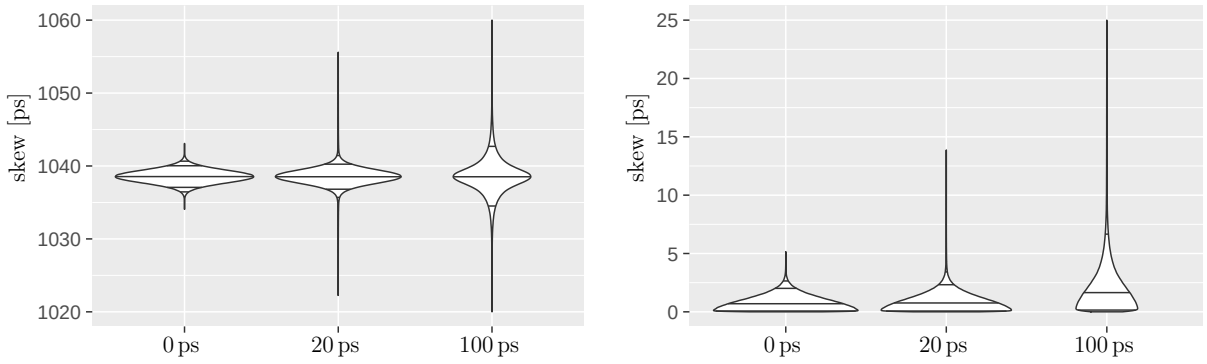
4. SIMULATIONS

In this section, we derive an estimation of the skews to be expected for a large single-chip system based on simulations. This choice allows for a reasonably fair comparison with the performance of clock trees. For single-die, multi-die or even larger systems, one would need to adapt parameters and model wires (and possibly ports) more carefully. Nonetheless, the knowledgeable reader may derive educated guesses on such systems from the qualitative behavior demonstrated by the simulations. Our approach is based on first determining a delay profile of our implementation of the critical path of a TRIX node (see Figure 2) and then using this profile for grid-level simulations.

Circuit Delay Evaluation. In order to evaluate the behavior of TRIX, a comprehensive model of the delay within each node has been established. The circuit has been modeled in SPICE using transistor models and parasitics for an UMC 65nm process. The inverter S_2 was modeled using a minimum sized inverter, while the others were chosen to be CLKINV20 to have enough drive strength to ensure fast rise/fall times. The input transistors sizing has been slightly optimized for minimum delay variation under different relative arrival times of the inputs A, B and C. The delay was measured from the 50% point of the second arriving pulse to the 50% point of an output.

The simulation results show a high symmetry between the three input pulse arrival times (see Figure 3) with the minimum delay being 35 ps and the maximum delay 43 ps. The delay decreases when two inputs go high early in close proximity, while the third is late. And conversely, the delay increases when one pulse goes high early and the other two go high late and in close proximity. It is interesting to note that this behavior causes the delay minimum not to be in the center of the plots, where all three pulses arrive at the same time, but there are three distinct minima.

The effects of degenerate inputs have not been simulated in SPICE, because finding the pulse form that increases or decreases the delay maximally is intractable. However, only faulty nodes produce degenerate signals, and these are assumed to be few; neither stuck-at faults nor early or late transitions produce degenerate signals. Hence, while one



(a) (b)

Figure 4: (a) violin plots of the inter-layer skew in the three settings without faults. For better readability, we cropped the plot for $\Delta = 100$; the minimum resp. maximum are 941 ps resp. 1134 ps. (b) the corresponding intra-layer skews were, again, the 100 ps dataset is cropped due to a maximum skew of 81 ps.

may expect that a degenerate signal may affect minimum and maximum delay by a few picoseconds, this "additional" source of skew is very rare. Moreover, as we can see from the grid-level simulations (cf. Figure 5), large skews at the input are reduced with increasing layer number, strongly suggesting that the grid will easily distribute and thereby mitigate the effect of degenerate signals on the skew.

Simulation of the Grid. The simulations were performed with a custom simulation tool written in C++. We performed 500 single pulse propagations in a grid with $W = 25$ columns and $H = 50$ layers, both with and without faults, and for varying skews on layer 0, i.e., at the input. Faults were placed uniformly at random under the constraint that each correct node had at most one fault on its incoming links. As faulty nodes may behave arbitrarily, they were not considered for skew evaluation, i.e., skews are measured only between adjacent correct nodes. Concerning the pulse times (of correct nodes) on layer 0, we choose each independently and uniformly distributed between 0 and Δ ps, for $\Delta \in \{0, 20, 100\}$, resulting in $\sigma_0 \approx \Delta$ ps. Here, $\Delta = 0$ permits to study the skew exclusively caused during forwarding of the pulse, $\Delta = 20$ represents inputs that are reasonably well-synchronized, and $\Delta = 100$ showcases how the grid behaves under bad inputs.

The time when node (ℓ, i) on layer $\ell > 0$ pulsed was determined from the inputs provided by nodes $(\ell - 1, i - 1)$, $(\ell - 1, i)$, and $(\ell - 1, i + 1)$ as follows. First, we applied independently normally distributed wire delays with 1 ns mean and 1 ps standard deviation. Then, we determined the delay of the circuit from Figure 2 from the result of the SPICE simulations. Finally, we added another normally distributed random value with zero mean and 1 ps standard deviation to introduce some noise into the system. In this process, the randomness in the delay models deviations due to manufacturing, temperature, supply voltage, etc. Note that only the deviation between the nodes matters; global bias does not affect skews. Moreover, we verified that the difference between maximum and minimum delay of the circuit in the SPICE simulations was essentially unaffected by having either all transistors operate at their slow corner or all of them at their fast corner, respectively. Accordingly, we consider

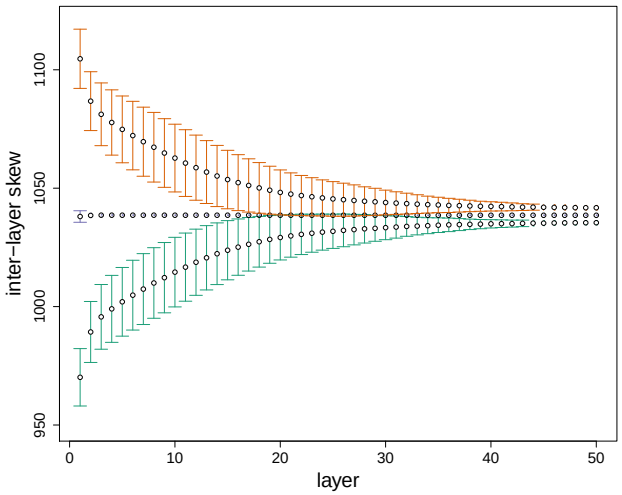


Figure 5: Visualization of the inter-layer skew of the data set with $\Delta = 100$ and no faults. Averages and standard deviations over 500 simulation runs are plotted on a per-layer basis. To avoid clutter, the standard deviation was dropped if it was below 0.1. The top data series shows the maximum, the middle the average, and the lower the minimum.

the small standard deviation of 1 ps appropriate for estimating the skew of a TRIX grid on a single chip.

The results are given in violin plots, i.e., by plotting the kernel density plot in a mirrored fashion, where a kernel density plot is, for our purposes, a continuous histogram. For an easy comparison of the distributions, all violins inside our plots occupy the same area. Additionally, horizontal lines inside the violin plot mark the median, $\{95, 90, 5\}$% quantiles, and in case of the inter-layer plots also the 1% quantile.

Figure 4 depicts the result for the fault-free case. As we see from the simulations for $\Delta = 0$, the grid itself maintains small skews under perfect input. The extreme values (over 500 runs and the entire grid!) of the inter-layer skew are less

f	σ_{avg}[ps]	σ_{max}[ps]	$\hat{\sigma}_{min}$[ps]	$\hat{\sigma}_{avg}$[ps]	$\hat{\sigma}_{max}$[ps]
0	0.821	5.146	1027.312	1031.868	1036.391
1	0.831	6.920	1024.472	1031.864	1036.855
5	0.865	7.441	1023.636	1031.847	1037.967
10	0.908	8.263	1021.374	1031.826	1038.988

Table 1: An overview over the skew development under an increasing number f of stuck-at-0 faults with $\Delta = 0$. We denote with σ the intra-layer skews, while $\hat{\sigma}$ denotes the corresponding inter-layer skews.

than 10 ps apart. This shows that skews do not accumulate, but are even better than the simulations of the firing logic of a single node might suggest.

For $\Delta = 20$ and $\Delta = 100$, we see that the extreme values all change by roughly Δ, while the majority of skews still remains small. This indicates a "garbage in, garbage out" situation. Indeed, when plotting the average maximum, average minimum, and average inter-layer skews as function of the layer index for $\Delta = 100$ (Figure 5), skews drop until in the later layers the range is, again, smaller than 10 ps.

While one should take these results with a grain of salt—our simulations represent noise sources, manufacturing differences, etc. by the sum of two normal distributions with standard deviation of 1 ps—we believe that these results demonstrate that, in terms of skew, a well-engineered TRIX grid will perform very similar to more conventional clock distribution topologies. However, what about faults? As one might already suspect from Figure 5, the performance of the grid is hardly affected by faults at all, see Table 1. Even an error rate of close to 1% (10 out of 1275 nodes) affected skews by at most roughly 3 ps under perfectly synchronized inputs. With larger skews at the input, even this effect disappears, as the input skews dominate the extreme values. Moreover, more involved fault patterns resulted in very similar behavior. We argue that this data predicts excellent scalability of our clock distribution method.

5. CONCLUSION

We presented TRIX, a fault-tolerant clock distribution scheme. We provided an efficient and small transistor level implementation of the central node component, resulting in skews comparable to traditional clocking schemes. While the need for employing small clock trees to locally distribute a node's clock signal is likely to result in slightly larger skews, due to its high degree of fault-tolerance TRIX allows for avoiding the single point of failure constituted by standard clocking mechanisms. Thus, one may retain small skews comparable to traditional clocking schemes, yet build truly redundant systems without giving up the guarantees that design engineers have gotten used to.

6. REFERENCES

[1] D. Dolev, M. Függer, C. Lenzen, M. Perner, and U. Schmid, "HEX: Scaling honeycombs is easier than scaling clock trees," *JCSS*, vol. 82, no. 5, pp. 929–956, 2016.

[2] K. Driscoll, B. Hall, M. Paulitsch, P. Zumsteg, and H. Sivencrona, "The real byzantine generals," in *DACS*, vol. 2, 2004, pp. 6.D.4–61–11 Vol.2.

[3] R. Chipana and F. L. Kastensmidt, "Set susceptibility analysis of clock tree and clock mesh topologies," in *ISVLSI*, 2014, pp. 559–564.

[4] R. Chipana, F. L. Kastensmidt, J. Tonfat, and R. Reis, "Set susceptibility estimation of clock tree networks from layout extraction," in *LATW*, 2012, pp. 1–6.

[5] V. Malherbe, G. Gasiot, S. Clerc, F. Abouzeid, J. L. Autran, and P. Roche, "Investigating the single-event-transient sensitivity of 65 nm clock trees with heavy ion irradiation and monte-carlo simulation," in *IRPS*, 2016, pp. SE–3–1–SE–3–5.

[6] L. Wissel, D. F. Heidel, M. S. Gordon, K. P. Rodbell, K. Stawiasz, and E. H. Cannon, "Flip-flop upsets from single-event-transients in 65 nm clock circuits," *IEEE Trans. Nucl. Sci.*, vol. 56, no. 6, pp. 3145–3151, 2009.

[7] H. B. Wang, N. Mahatme, L. Chen, M. Newton, Y. Q. Li, R. Liu, M. Chen, B. L. Bhuva, K. Lilja, S. J. Wen, R. Wong, R. Fung, and S. Baeg, "Single-event transient sensitivity evaluation of clock networks at 28-nm cmos technology," *IEEE Trans. Nucl. Sci.*, vol. 63, no. 1, pp. 385–391, 2016.

[8] R. Chipana, F. L. Kastensmidt, J. Tonfat, R. Reis, and M. Guthaus, "Set susceptibility analysis in buffered tree clock distribution networks," in *RADECS*, 2011, pp. 256–261.

[9] F. Abouzeid, S. Clerc, C. Bottoni, B. Coeffic, J. M. Daveau, D. Croain, G. Gasiot, D. Soussan, and P. Roche, "28nm fd-soi technology and design platform for sub-10pj/cycle and ser-immune 32bits processors," in *ESSCIRC*, 2015, pp. 108–111.

[10] A. Gujja, S. Chellappa, C. Ramamurthy, and L. T. Clark, "Redundant skewed clocking of pulse-clocked latches for low power soft error mitigation," in *RADECS*, 2015, pp. 1–7.

[11] C. L. Lung, Y. S. Su, S. H. Huang, Y. Shi, and S. C. Chang, "Fault-tolerant 3d clock network," in *DAC*, 2011, pp. 645–651.

[12] H. Park and T. Kim, "Comprehensive technique for designing and synthesizing TSV fault-tolerant 3D clock trees," in *ICCAD*, 2013, pp. 691–696.

[13] ——, "Synthesis of TSV fault-tolerant 3-D clock trees," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 2, pp. 266–279, 2015.

[14] S. Fairbanks, "Method and Apparatus for a Distributed Clock Generator," 2006, US patent no. 7,126,405 B2, Filed December 2nd., 2003, Issued October 24th., 2006.

[15] M. Függer, A. Dielacher, and U. Schmid, "How to Speed-Up Fault-Tolerant Clock Generation in VLSI Systems-on-Chip via Pipelining," in *EDCC*, 2010, pp. 230–239.

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

[17] D. Dolev, M. Függer, 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," *JCSS*, vol. 80, no. 4, pp. 860–900, 2014.

[18] P. Khanchandani and C. Lenzen, "Self-stabilizing Byzantine Clock Synchronization with Optimal Precision," in *SSS*, 2016, pp. 213–230.