Metastability-Containing Circuits

Stephan Friedrichs, Matthias Függer, Christoph Lenzen

Metastability-Containing Circuits

Stephan Friedrichs, Matthias Függer, Christoph Lenzen

Abstract—In digital circuits, *metastability* can cause deteriorated signals that neither are logical 0 nor logical 1, breaking the abstraction of Boolean logic. Synchronizers, the only traditional countermeasure, exponentially decrease the odds of maintained metastability over time. We propose a fundamentally different approach: It is possible to deterministically *contain* metastability by fine-grained logical masking so that it cannot infect the entire circuit.

At the heart of our approach lies a time- and value-discrete model for metastability in synchronous clocked digital circuits, in which metastability is propagated in a worst-case fashion. The proposed model permits positive results and passes the test of reproducing Marino's impossibility results. We fully classify which functions can be computed by circuits with standard registers. Regarding masking registers, we show that more functions become computable with each clock cycle, and that masking registers permit exponentially smaller circuits for some tasks. Demonstrating the applicability of our approach, we present the first fault-tolerant distributed clock synchronization algorithm that deterministically guarantees correct behavior in the presence of metastability. As a consequence, clock domains can be synchronized without using synchronizers, enabling metastability-free communication between them.

Index Terms—Metastability, Metastability-Containment, Logical Masking, Masking Register, Clock Synchronization.

1 Introduction

A classic image invoked to explain metastability is a ball "resting" on the peak of a steep mountain. In this unstable equilibrium the tiniest displacement exponentially self-amplifies, and the ball drops into a valley. While for Sisyphus metastability admits some nanoseconds of respite, it fundamentally disrupts operation in VLSI circuits by breaking the abstraction of Boolean logic.

In digital circuits, every bistable storage element can become *metastable*. Metastability refers to volatile states that usually involve an internal voltage strictly between logical 0 and 1. A metastable storage element can output deteriorated signals, e.g., voltages stuck between logical 0 and logical 1, oscillations, late or unclean transitions, or otherwise unspecified behavior. Such deteriorated signals may violate timing constraints or input specifications of gates and further storage elements. Hence, deteriorated signals may spread through combinational logic and drive further bistables into metastability. While metastability refers to a state of a bistable, we refer to the above mentioned deteriorated signals as "metastable" for the sake of exposition.

Unfortunately, any way of reading a signal from an unsynchronized clock domain or performing an analog-to-digital or time-to-digital conversion incurs the risk of a metastable result; no physical implementation of a non-trivial digital circuit can deterministically avoid, resolve, or detect metastability [28].

Traditionally, the only countermeasure is to write a potentially metastable signal into a synchronizer [3], [4], [5], [17], [23], [24] and wait. Synchronizers exponentially decrease the odds of maintained metastability over time [23], [24], [37]: In this unstable equilibrium the tiniest displacement exponentially self-amplifies and the bistable resolves metastability. Put differently, the waiting time determines the probability to resolve to logical 0 or 1. Accordingly, this approach delays subsequent computations and does not guarantee success.

We propose a fundamentally different approach: It is possible to *contain* metastability by fine-grained logical masking so that it cannot infect the entire circuit. This technique *guarantees* a limited degree of metastability in — and uncertainty about — the output. At the heart of our approach lies a model for metastability in synchronous clocked digital circuits. Metastability is propagated in a worst-case fashion, allowing to derive deterministic guarantees, without and unlike synchronizers.

The Challenge. The problem with metastability is that it fundamentally disrupts operation in VLSI circuits by breaking the abstraction of Boolean logic: A metastable signal can neither be viewed as being logical 0 or 1. In particular, a metastable signal is not a random bit, and does not behave like an unknown but fixed Boolean signal. As an example, the circuit that computes $\neg x \vee x$ using a NOT and a binary OR gate may output an arbitrary signal value if x is metastable: 0, 1, or again a metastable signal. Note that this is not the case for unknown, but Boolean, x. The ability of such signals to "infect" an entire circuit poses a severe challenge.

The Status Quo. The fact that metastability cannot be avoided, resolved or detected, the hazard of infecting entire circuits, and the unpleasant property of breaking the abstraction of Boolean logic have led to the predominant belief that waiting — using well-designed synchronizers — essentially is the *only* method of coping with the threat of metastability: Whenever a signal is potentially metastable, e.g., when it is communicated across a clock boundary, its

Full proofs available at http://arxiv.org/abs/1606.06570

- *Stephan Friedrichs and Christoph Lenzen are with the Max Planck Institute for Informatics, Saarland Informatics Campus, Germany, Email: {sfriedri, clenzen}@mpi-inf.mpg.de*
- *Matthias Függer is affiliated with the CNRS & LSV, ENS Paris-Saclay & Inria, Email: mfuegger@lsv.fr*
- *Stephan Friedrichs is with the Saarbrücken Graduate School of Computer Science*

value is written to a synchronizer. After a predefined time, the synchronizer output is assumed to have stabilized to logical 0 or 1, and the computation is carried out in classical Boolean logic. In essence, this approach trades synchronization delay for increased reliability; it does, however, not provide deterministic guarantees.

Relevance. VLSI circuits grow in complexity and operating frequency, leading to a growing number unsynchronized clock domains, technology becomes smaller, and the operating voltage is decreased to save power [21]. These trends increase the risk of metastable upsets. Treating these risks in the traditional way — by adding synchronizer stages — increases synchronization delays and thus is counterproductive w.r.t. the desire for faster systems. Hence, we urgently need alternative techniques to reliably handle metastability in both mission-critical and day-to-day systems.

Our Approach. We challenge the point of view that synchronizers are the only solution to metastability and exploit that *logical masking* provides some leverage. If, e.g., one input of a NAND gate is stable 0, its output remains 1 even if its other input is arbitrarily deteriorated. This is owed to the way gates are implemented in CMOS logic and to transistor behavior under intermediate input voltage levels.

We conclude that it is possible to *contain* metastability to a limited part of the circuit instead of attempting to resolve, detect, or avoid it altogether. Given Marino's result [28], this is surprising, but not a contradiction. More concretely, we show that a variety of operations can be performed in the presence of a limited degree of metastability in the input, maintaining an according guarantee on the output.

As an example, recall that in Binary Reflected Gray Code (BRGC) x and $x+1$ always only differ in exactly one bit; each upcount flips one bit. Suppose Analog-to-Digital Converters (ADCs) output BRGC but, due to their analog input, a possibly metastable bit u decides whether to output x or $x+1$. As x and $x+1$ only differ in a single bit, this bit is the only one that may become metastable in an appropriate implementation. Hence, all possible stabilizations are in $\{x, x+1\}$, we refer to this as *precision*-1. Among other things, we show that it is possible to sort such inputs in a way that the output still has precision-1.

We assume worst-case metastability propagation and still are able to *guarantee* correct results. This opens up an alternative to the classic approach of *postponing* the actual computation by first using synchronizers. Advantages over synchronizers are:

1) No time is lost waiting for (possible) stabilization. This permits fast response times as, e.g., useful for high-frequency clock synchronization in hardware, see Section 9. Note that this removes synchronization delay from the list of fundamental limits to the operating frequency.

2) Correctness is guaranteed deterministically instead of probabilistically.

3) Stabilization can, but is not required to, happen "during" the computation, i.e., synchronization and calculation happen simultaneously. In [33] our approach has been applied to a Network-on-Chip router: the authors replaced the synchronizers in the receive circuit and replaced it with a metastability-containing state machine implementation, resulting in lower packet delivery time.

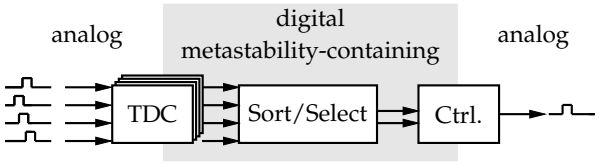


Figure 1. The separation of concerns (analog – digital metastability-containing – analog) for fault-tolerant clock synchronization in hardware.

Separation of Concerns. Clearly, the impossibility of resolving metastability still holds; metastability may still occur, even if it is contained. Hence, a *separation of concerns*, compare Figure 1, is key to our approach.

For the purpose of illustration, consider a hardware clock-synchronization algorithm, which is discussed in more detail in Section 9. We start in the *analog* world: nodes generate clock pulses. Each node measures the time differences between its own and all other nodes' pulses using Time-to-Digital Converters (TDCs). Since this involves entering the *digital* world, metastability in the measurements is unavoidable [28]. The traditional approach is to hold the TDC outputs in synchronizers, spending time and thus imposing a limit on the operating frequency. But as discussed above, it is possible to limit the metastability of each measurement to at most one bit in BRGC-encoded numbers, where the metastable bit represents the "uncertainty between x and $x+1$ clock ticks," i.e., precision-1.

We apply *metastability-containing* components to digitally process these inputs to derive digital correction parameters for the node's oscillator. These parameters contain at most one metastable bit, as above accounting for precision-1. We convert them to an *analog* control signal for the oscillator, translating the metastability to a small frequency offset within the uncertainty from the initial TDC measurements.

In short, metastability is introduced at the TDC, *deterministically* contained in the digital subcircuit, and ultimately absorbed by the analog control signal.

Our Contribution. In Section 3, we present a rigorous time-discrete value-discrete model for metastability in clocked as well as in purely combinational digital circuits. We consider two types of registers: simple (standard) registers that do not provide any guarantees regarding metastability and masking registers that can "hide" internal metastability to some degree using high- or low-threshold inverters. The propagation of metastability is modeled in a worst-case fashion and metastable registers may or may not stabilize to 0 or 1. Hence, the resulting model allows us to derive deterministic guarantees concerning circuit behavior under metastable inputs.

We demonstrate that the model is not too pessimistic, i.e., that it allows non-trivial positive results. At the same time, we are obligated to verify that it properly reflects the physical behavior of digital circuits, i.e., that it is sufficiently pessimistic. We perform a reality check in Section 5, showing that the physical impossibility of avoiding, resolving, or detecting metastability [28] holds in our model.

Having established some confidence that our model properly reflects the physical world and allows reasoning about circuit design, we turn our attention to the question of computability in Section 6. In Section 6, we analyze what functions are computable by circuits w.r.t. the available

register types and the number of clock cycles. Let Fun_M^r denote the class of functions that can be implemented in r clock cycles; let Fun_S^r denote the class of functions implementable in r clock cycles of circuits that can only use simple registers. We show that the number of clock cycles is irrelevant for combinational and simple circuits, reflecting the intuition from electrical engineering that synchronous Boolean circuits can be unrolled, but that this is not the case in the presence of masking registers:

$$\cdots = \mathrm{Fun}_S^2 = \mathrm{Fun}_S^1 = \mathrm{Fun}_M^1 \subsetneq \mathrm{Fun}_M^2 \subsetneq \cdots . \quad (1)$$

In Section 7, we fully classify Fun_S. Furthermore, we establish the *metastable closure,* the strictest possible extension of a function specification that allows it to be computed without masking registers.

Section 8 establishes that the closure can be efficiently computed using masking registers. This is exponentially more efficient than the best implementation without masking registers that we are aware of. Moreover, recently the existence of circuits for which implementing the closure without masking registers *must* incurr an exponential overhead has been shown [20].

Finally, we apply our techniques to show that an advanced, useful circuit is in reach. We show in Section 9 that all operations required by the widely used [6], [25] fault-tolerant clock synchronization algorithm of Lundelius Welch and Lynch [27] — $\max$ and $\min$, sorting, and conversion between Thermometer Code (TC) and BRGC — can be performed in a metastability-containing manner. Employing the above mentioned separation of concerns, a hardware implementation of the entire algorithm is within reach, providing a deterministic correctness guarantee despite metastable upsets originating in the TDC and without synchronizers. As a consequence, we show that (1) synchronization delay poses no fundamental limit on the operating frequency of clock synchronization and that (2) clock domains can be synchronized without synchronizers. The latter shows that we may eliminate communication across unsynchronized clock domains as a source of metastable upsets altogether.

2 Related Work

Metastability. The phenomenon of metastability has been studied for decades [23] with the following key results. (1) No physical implementation of a digital circuit can reliably avoid, resolve, or detect metastability; any non–constant digital circuit, including "detectors," can become metastable [28]. (2) The probability of an individual event generating metastability can be kept low. Large transistor counts and high operational frequencies, low supply voltages, temperature effects, and changes in technology, however, disallow to neglect the problem [4]. (3) Being an unstable equilibrium, the probability that, e.g., a memory cell remains in a metastable state decreases exponentially over time [23], [24], [37]. Thus, waiting for a sufficiently long time reduces the probability of sustained metastability to within acceptable bounds.

Synchronizers. The predominant technique to cope with metastable upsets is to use synchronizers [3], [4], [5], [17], [23], [24], carefully designed [3], [17] bistable storage elements that hold potentially metastable signals. After a predefined time, the synchronizer output is assumed to have stabilized and the computation is carried out in classical Boolean logic. In essence, this approach trades synchronization delay for increased reliability, typically expressed as MTBF. Synchronizers, however, do not provide deterministic guarantees and avoiding synchronization delay is an important issue [34], [35].

Glitch/Hazard Propagation. Metastability-containing circuits are related to glitch/hazard-free circuits, which have been extensively studied since Huffman [19] and Unger [36] introduced them. Eichelberger [12] extended these results to multiple switching inputs and dynamic hazards, Brzozowski and Yoeli extended the simulation algorithm [8], Brzozowski et al. surveyed techniques using higher-valued logics [7] such as Kleene's 3-valued extension of Boolean logic, and Mendler et al. studied delay requirements needed to achieve consistency with simulated results [29].

While we too resort to Kleene's 3-valued to model metastability, there are differences to the classical work on hazard-tolerant circuits: (1) A common assumption in hazard detection is that inputs only perform well-defined, clean transitions, i.e., the assumption of a hazard-free input-generating circuitry is made. This is the key difference to metastability-containment: Metastability encompasses much more than inputs that are in the process of switching; metastable signals may or may not be in the process of completing a transition, may be oscillating, and may get "stuck" at an intermediate voltage. (2) Another common assumption in hazard detection is that circuits have a constant delay. This is no longer the case in the presence of metastability; unless metastability is properly masked, circuit delays can deteriorate in the presence of metastable input signals, even if the circuit eventually generates a stable output [15]. This can cause late transitions that potentially drive further registers into metastability. (3) Glitch-freedom is no requirement for metastability-containment. (4) When studying synthesis, we allow for specifications in which outputs may contain metastable bits. This is necessary for non-trivial specifications in the presence of metastable inputs [28]. (5) We allow a circuit to compute a function in multiple clock cycles. (6) Circuits may comprise masking registers [23].

While static hazards as studied by [19] inherently model different signal behavior than metastable signals, our Theorem 23 shows that similar techniques can be applied in both cases: like in static-hazard-free circuits, covering prime-implicants is a technique to achieve metastability-containment. However, this method potentially leads to exponential size circuits, which was recently proven to be inevitable in general [20]. Sections 8 and 9 demonstrate that metastability-containing circuits are not necessarily large: we present a method that circumvents exponential blow-up by using masking registers, and show that clock synchronization components do not suffer from this blow-up.

OR Causality. The work on weak (OR) causality in asynchronous circuits [38] studies the computation of functions under availability of only a proper subset of its parameters. As an example, consider a Boolean function $f(x, y)$, where $f(0, 0) = f(0, 1)$. An early-deciding asynchronous module may set its output as soon as $x = 0$ arrives at its input, disregarding the value of y. Early-deciding

circuits, however, differ from our work because they are neither clocked synchronous designs nor do they necessarily operate correctly in presence of metastable input bits: $f(0,\mathrm{M}) = f(0,0) = f(0,1)$ does not necessarily hold.

Speculative Computing. To the best of our knowledge, the most closely related work is that by Tarawneh et al. on speculative computing [34], [35]. The idea is the following: When computing $f(x,y)$ in presence of a potentially metastable input bit x, (1) speculatively compute both $f(0,y)$ and $f(1,y)$, (2) in parallel, store the input bit x in a synchronizer for a predefined time that provides a sufficiently large probability of resolving metastability of x, and (3) use x to select whether to output $f(0,y)$ or $f(1,y)$. This hides (part of) the delay needed to synchronize x.

Like our approach, speculative computations allow for an overlap of synchronization and computation time. The key differences are: (1) Relying on synchronizers, speculative computing incurs a non-zero probability of failure; metastability-containment insists on deterministic guarantees. (2) In speculative computing, the set of potentially metastable bits X must be known in advance. Regardless of the considered function, the complexity of a speculative circuit grows exponentially in $|X|$. Neither is the case for metastability-containment, as illustrated by several circuits [9], [16], [26], [33]. (3) Our model is rooted in an extension of Boolean logic, i.e., uses a different function space. Hence, we face the question of computability of such functions by digital circuits; this question does not apply to speculative computing as it uses traditional Boolean functions.

Metastability-Containing Circuits. Many of the proposed techniques have been successfully employed to obtain metastability-aware TDCs [16], metastability-containing BRGC sorting networks [9], [26], metastability-containing multiplexers [15], and metastability-tolerant network-on-chip routers [33]. Simulations verify the positive impact of metastability-containing techniques [9], [15], [33]. Most of these works channel efforts towards metastability-containing FPGA and ASIC implementations of fault-tolerant distributed clock synchronization; this paper establishes that all required components are within reach.

3 Model of Computation

$\mathbb{N}_0$ and $\mathbb{N}$ denote the natural numbers with and without 0. We abbreviate $[k] := \{\ell \in \mathbb{N}_0 \mid \ell < k\}$ for $k \in \mathbb{N}_0$. Tuples a, b are concatenated by $a \circ b$, and given a set S, $\mathcal{P}(S) := \{S' \subseteq S\}$ is its power set.

We propose a time-discrete and value-discrete model in which registers can become metastable and their resulting output signals deteriorated. The model supports synchronous, clocked circuits composed of registers and combinational logic and purely combinational circuits. Specifically, we study the generic synchronous state-machine design depicted in Figure 2. Data is initially written into input registers. At each rising clock transition, local and output registers update their state according to the circuit's combinational logic. Figure 2(b) shows the circuit's behavior over time: (1) During the first phase, the output of the recently updated local and output registers stabilizes. This is accounted for by the *clock-to-output* time that can be

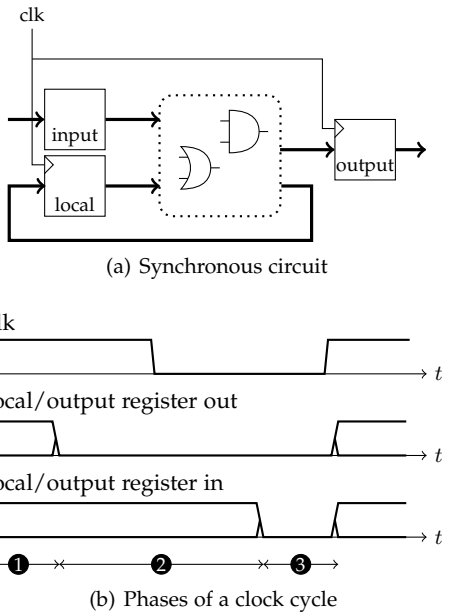


Figure 2. Generic synchronous state machine design in (a). The input register is initially prefilled. Local and output registers are updated at each rising clock transition. The circuit behavior over time is depicted in (b). The three phases of a clock cycle are shown: (1) register output stabilization, (2) propagation of outputs through combinational logic to register inputs, and (3) stable register inputs.

bounded, except for the case of a metastable register. In this case, no deterministic upper bound exists. (2) During phase two, the stable register output propagates through the combinational logic to the register inputs. Its duration can be upper-bounded by the worst-case propagation delay through the combinational part. (3) In the third phase, the register inputs are stable, ready to be read (sampled), and result in updated local and output register states. The duration of this phase is chosen such that it can account for potential delays in phase (1); this can mitigate some metastable upsets. If the stabilization in phase (1), however, also exceeds the additional time in phase (3), a register may read an unstable input value, potentially resulting in a metastable register.

As motivated, metastable registers output an undefined, arbitrarily deteriorated signal. Deteriorated can mean any constant voltage between logical 0 and logical 1, arbitrary signal behavior over time, oscillations, or simply violated timing constraints, such as late signal transitions. Furthermore, deteriorated signals can cause registers to become metastable, e.g., due to violated constraints regarding timing or input voltage. Knowing full well that metastability is a state of a *bistable* element and not a signal value or voltage, we still need to talk about the "deterioration caused by or potentially causing metastability in a register" in *signals.* For the sake of presentation — and as these effects are causally linked — we refer to both phenomena using the term metastability without making the distinction explicit.

Our model uses Kleene's 3-valued logic, a ternary extension of binary logic; the third value appropriately expresses the uncertainty about gate behavior in the presence of metastability. In the absence of metastability, our model behaves like a traditional, deterministic, binary circuit model. In order to obtain *deterministic* guarantees, we assume worst-case

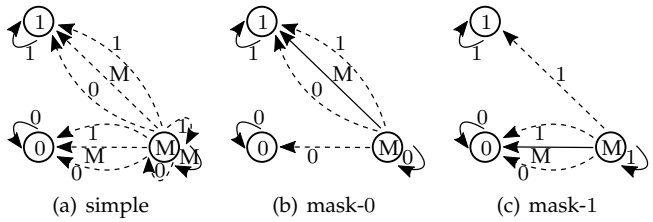


Figure 3. Registers as non-deterministic state machines; state transitions represent reads and are associated with an output. As we propose a worst-case model, the dashed state transitions can be left out.

propagation of metastability: If a signal can be "infected" by metastability, there is no way to prevent that.

With the example of metastability-containing multiplexers (CMUXes) we demonstrates our model, and Section 5 ensures that it is not "too optimistic" by proving that it reproduces well-known impossibility results. Concretely, we show that for circuits in our model avoiding, detecting, and resolving metastability is impossible, just as in physical circuits [28]. Clearly, this obliges us to provide evidence that our model has practical relevance, i.e., that it is indeed possible to perform meaningful computations. Surprisingly, the classification derived in Section 7 entails that many interesting functions can be implemented by circuits, which is discussed in Section 9.

In our model circuits are synchronous state machines: Combinational logic, represented by gates, maps a circuit state to possible successor states. The combinational logic uses, and registers store, *signal values* $\mathbb{B}_\mathrm{M} := \{0, 1, \mathrm{M}\}$. M represents a *metastable* signal, the only source of nondeterminism. The classical *stable* Boolean signal values are $\mathbb{B} := \{0, 1\}$. Let $x \in \mathbb{B}_\mathrm{M}^k$ be a k-bit tuple. Stored in registers over time, the metastable bits may resolve to 0 or 1. The set of partial resolutions of x is $\mathrm{Res}_\mathrm{M}(x)$, and the set of metastability-free, i.e., completely stabilized, resolutions is $\mathrm{Res}(x)$. If m bits in x are metastable, $|\mathrm{Res}_\mathrm{M}(x)| = 3^m$ and $|\mathrm{Res}(x)| = 2^m$, since M serves as "wildcard" for $\mathbb{B}_\mathrm{M}$ and $\mathbb{B}$, respectively. Formally,

$$\mathrm{Res}_\mathrm{M}(x) := \left\{ y \in \mathbb{B}_\mathrm{M}^k \mid \forall i \in [k] \colon x_i = y_i \vee x_i = \mathrm{M} \right\}, \quad (2)$$

$$\mathrm{Res}(x) := \mathrm{Res}_\mathrm{M}(x) \cap \mathbb{B}^k. \quad (3)$$

Registers. We consider three types of single-bit registers, all of which behave just like in binary circuit models unless metastability occurs: (1) *simple* registers which are oblivious to metastability, and (2) registers that mask an internal metastable state to an output of 1 *(mask-1)* or (3) to 0 *(mask-0)*. Physical realizations of masking registers are obtained by flip-flops with high- or low-threshold inverters at the output, amplifying an internal metastable signal to 1 or 0; see, e.g., Section 3.1 on metastability filters in [23]. A register R has a *type* (simple, mask-0, or mask-1) and a state $x_R \in \mathbb{B}_\mathrm{M}$. R behaves according to x_R and its type's non-deterministic state machine in Figure 3. Each clock cycle, R performs one state transition annotated with some $o_R \in \mathbb{B}_\mathrm{M}$, which is the result of sampling R at that clock cycle's rising clock edge. This happens exactly once per clock cycle in our model and we refer to it as *reading* R. The state transitions are not caused by sampling R but account for the possible resolution of metastability during the preceding clock cycle.

Table 1
Gate behavior under metastability corresponds to Kleene logic.

$f_\mathrm{M}^{\mathrm{AND}}$	0	1	M
0	0	0	0
1	0	1	M
M	0	M	M

$f_\mathrm{M}^{\mathrm{OR}}$	0	1	M
0	0	1	M
1	1	1	1
M	M	1	M

Consider a simple register in Figure 3(a). When in state 0, its output and successor state are both 0; it behaves symmetrically in state 1. In state M, however, any output in $\mathbb{B}_\mathrm{M}$ combined with any successor state in $\mathbb{B}_\mathrm{M}$ is possible.

Since our goal is to design circuits that operate correctly under metastability even if it never resolves, we make two pessimistic simplifications: (1) If there are three parallel state transitions from state x to x' with outputs $0, 1, \mathrm{M}$, we only keep the one with output M, and (2) if, for some fixed output $o \in \mathbb{B}_\mathrm{M}$, there are state transitions from a state x to multiple states including M, we only keep the one with successor state M. This simplification is obtained by ignoring the dashed state transitions in Figure 3, and we maintain it throughout the paper. Observe that the dashed lines are a remnant of the highly non-deterministic "anything can happen" behavior in the physical world; if one is pessimistic about the behavior, however, one obtains the proposed simplification that ignores the dashed state transitions.

The mask-b registers, $b \in \mathbb{B}$, shown in Figures 3(b) and 3(c), exhibit the following behavior: As long as their state remains M, they output $b \neq \mathrm{M}$; only when their state changes from M to $1 - b$ they output M once, after that they are stable.

Gates. We model the behavior of combinational gates in the presence of metastability. A *gate* is defined by $k \in \mathbb{N}_0$ input ports, one output port — gates with $k \geq 2$ distinct output ports are represented by k single-output gates — and a Boolean function $f \colon \mathbb{B}^k \to \mathbb{B}$. We generalize f to $f_\mathrm{M} \colon \mathbb{B}_\mathrm{M}^k \to \mathbb{B}_\mathrm{M}$ as follows. Each metastable input can be perceived as 0, as 1, or as metastable superposition M. Hence, to determine $f_\mathrm{M}(x)$, consider $O := \{f(x') \mid x' \in \mathrm{Res}(x)\}$, the set of possible outputs of f after x fully stabilized. If there is only a single possible output, i.e., $O = \{b\}$ for some $b \in \mathbb{B}$, the metastable bits in x have no influence on $f(x)$ and we set $f_\mathrm{M}(x) := b$. Otherwise, $O = \mathbb{B}$, i.e., the metastable bits can change $f(x)$, and we set $f_\mathrm{M}(x) := \mathrm{M}$. Observe that this is equivalent to Kleene's 3-valued logic and that $f_\mathrm{M}(x) = f(x)$ for all $x \in \mathbb{B}^k$.

See Table 1 for an AND-gate and an OR-gate. Refer to Figure 6(a) for an example of metastability propagation through combinational logic.

Combinational Logic. We model combinational logic as Directed Acyclic Graph (DAG) $G = (V, A)$ with parallel arcs, compare Figure 4. Each node either is an *input node*, an *output node*, or a gate.

Input nodes are sources in the DAG, i.e., have in-degree 0 and an arbitrary out-degree, and output nodes are sinks with in-degree 1, i.e., have in-degree 1 and out-degree 0. If $v \in V$ is a gate, denote by $f_v \colon \mathbb{B}_\mathrm{M}^{k_v} \to \mathbb{B}_\mathrm{M}$ its gate function with $k_v \in \mathbb{N}_0$ parameters. For each parameter of f_v, v is connected to exactly one input node or gate w by an arc $(w, v) \in A$. Every output node v is connected to exactly one

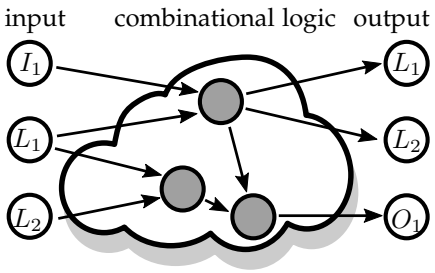


Figure 4. Combinational logic DAG with gates (gray) and registers (white). The input (I_1), output (O_1), and local (L_1 and L_2) registers occur as input nodes, output nodes, and both, respectively.

input node or gate w by an arc $(w, v) \in A$. Input nodes and gates can serve as input to multiple gates and output nodes.

Suppose G has m input nodes and n output nodes. Then G defines a function $f^G \colon \mathbb{B}_{\mathrm{M}}^m \to \mathbb{B}_{\mathrm{M}}^n$ as follows. Starting with input $x \in \mathbb{B}_{\mathrm{M}}^m$, we evaluate the nodes $v \in V$. If v is an input node, it evaluates to x_v. Gates of in-degree 0 are constants and evaluate accordingly. If v is a gate of non-zero in-degree, it evaluates to $f_v(\bar{x})$, where $\bar{x} \in \mathbb{B}_{\mathrm{M}}^{k_v}$ is the recursive evaluation of all nodes w with $(w, v) \in A$. Otherwise, v is an output node, has in-degree 1, and evaluates just as the unique node w with $(w, v) \in A$. Finally, $f^G(x)_v$ is the evaluation of the output node v.

Circuits.

Definition 1 (Circuit)**.** *A* circuit C *is defined by:*

(1) m input registers, k local registers, *and* n output registers, $m, k, n \in \mathbb{N}_0$. *Each register has exactly one* type — *simple, mask*-0*, or mask*-1 — *and is either input, output, or local register.*

(2) A combinational logic DAG G*.* G *has* $m + k$ *input nodes, exactly one for each non-output register, and* $k + n$ *output nodes, exactly one for each non-input register. Local registers appear as both input node and output node.*

(3) An initialization $x_0 \in \mathbb{B}_{\mathrm{M}}^{k+n}$ *of the non-input registers.*

Each $s \in \mathbb{B}_{\mathrm{M}}^{m+k+n}$ *defines a* state *of* C*.*

A meaningful application clearly uses a stable initialization $x_0 \in \mathbb{B}^{k+n}$; this restriction, however, is not formally required. Furthermore, observe that Definition 1 does not allow registers to be an input and an output register at the same time. This overlap in responsibilities, however, is often used in digital circuits. We note that we impose this restriction for purely technical reasons; our model supports registers that are read and written — local registers — and it is possible to emulate the above mentioned behavior: If the computation consists of a single round, read from the input and write to the output register. Otherwise, read form the input register in the first round, write and read from local registers in successive rounds, and write to the output register in in the last round. Hence, this formal restriction has no practical implications.

We denote by

$$\mathrm{In} \colon \mathbb{B}_{\mathrm{M}}^{m+k+n} \to \mathbb{B}_{\mathrm{M}}^m, \tag{4}$$

$$\mathrm{Loc} \colon \mathbb{B}_{\mathrm{M}}^{m+k+n} \to \mathbb{B}_{\mathrm{M}}^k, \text{ and} \tag{5}$$

$$\mathrm{Out} \colon \mathbb{B}_{\mathrm{M}}^{m+k+n} \to \mathbb{B}_{\mathrm{M}}^n \tag{6}$$

the projections of a circuit state to its values at input, local, and output registers, respectively. In fact, the initialization of the output registers, $\mathrm{Out}(x_0)$, is irrelevant, because output registers are never read (see below). We use the convention that for any state s of a circuit, $s = \mathrm{In}(s) \circ \mathrm{Loc}(s) \circ \mathrm{Out}(s)$.

Executions. Consider a circuit C in state s, and let $x = \mathrm{In}(s) \circ \mathrm{Loc}(s)$ be the state of the non-output registers. Suppose each register R is read, i.e., makes a non-dashed state transition according to its type, state, and corresponding state machine in Figure 3. This state transition yields a value read from, as well as a new state for, R. We denote by

$$\mathrm{Read}^C \colon \mathbb{B}_{\mathrm{M}}^{m+k} \to \mathcal{P}\left(\mathbb{B}_{\mathrm{M}}^{m+k}\right) \tag{7}$$

the function mapping x to the set of possible values read from non-output registers of C depending on x. When only simple registers are involved, the read operation is deterministic:

Observation 2. *In a circuit* C *with only simple registers,* $\mathrm{Read}^C(x) = \{x\}$.

In the presence of masking registers, $x \in \mathrm{Read}^C(x)$ can occur, but the output may partially stabilize:

Observation 3. *Consider a circuit* C *in state* s*. Then for* $x = \mathrm{In}(s) \circ \mathrm{Loc}(s)$

$$x \in \mathrm{Read}^C(x), \textit{ and} \tag{8}$$

$$\mathrm{Read}^C(x) \subseteq \mathrm{Res}_{\mathrm{M}}(x). \tag{9}$$

Let G be the combinational logic DAG of C with $m + k$ input and $k + n$ output nodes. Suppose $o \in \mathbb{B}_{\mathrm{M}}^{m+k}$ is read from the non-output registers. Then the combinational logic of C evaluates to $f^G(o)$, uniquely determined by G and o. We denote all possible evaluations of C w.r.t. x by $\mathrm{Eval}^C(x)$:

$$\mathrm{Eval}^C \colon \mathbb{B}_{\mathrm{M}}^{m+k} \to \mathcal{P}\left(\mathbb{B}_{\mathrm{M}}^{k+n}\right), \tag{10}$$

$$\mathrm{Eval}^C(x) := \left\{ f^G(o) \mid o \in \mathrm{Read}^C(x) \right\}. \tag{11}$$

When registers are written, we allow, but do not require, signals to stabilize. If the combinational logic evaluates the new values for the non-input registers to $\bar{x} \in \mathbb{B}_{\mathrm{M}}^{k+n}$, their new state is in $\mathrm{Res}_{\mathrm{M}}(\bar{x})$; the input registers are never overwritten. We denote this by

$$\mathrm{Write}^C \colon \mathbb{B}_{\mathrm{M}}^{m+k} \to \mathcal{P}\left(\mathbb{B}_{\mathrm{M}}^{k+n}\right), \tag{12}$$

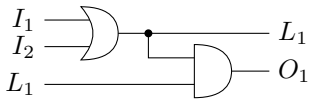
$$\mathrm{Write}^C(x) := \bigcup_{\bar{x} \in \mathrm{Eval}^C(x)} \mathrm{Res}_{\mathrm{M}}(\bar{x}). \tag{13}$$

Observe that this is where metastability can cause inconsistencies: If a gate is read as M and this is copied to three registers, it is possible that one stabilizes to 0, one to 1, and one remains M.

For the sake of presentation, we write $\mathrm{Read}^C(s)$, $\mathrm{Eval}^C(s)$, and $\mathrm{Write}^C(s)$ for a circuit state $s \in \mathbb{B}_{\mathrm{M}}^{m+k+n}$, meaning that the irrelevant part of s is ignored.

Let s_r be a state of C. A *successor state* s_{r+1} *of* s_r is any state that can be obtained from s_r as follows.

Read phase First read all registers, resulting in read values $o \in \mathrm{Read}^C(s_r)$. Let $\iota_{r+1} \in \mathbb{B}_{\mathrm{M}}^m$ be the state of the input registers after the state transitions leading to reading o.



(a) Circuit

r	state s_r $I_1I_2L_1O_1$	read o $I_1I_2L_1$	eval $\bar{x}_{r+1}$ L_1O_1	write x_{r+1} L_1O_1
0	MM11	0M1	MM	1M
1	MM1M	MM1	MM	MM
2	1MMM	1MM	1M	10
3	1M10	1M1	11	11
4	1M11			

(b) States, reads, evaluations, and writes

Figure 5. Example execution in a circuit (a). The node states as well as the results of the read, evaluation, and write phases are listed in the table (b). Register I_1 is a mask-0 register, all others are simple registers. The initialization is 11, the input is MM, and hence $s_0 = \text{MM11}$.

Evaluation phase Then evaluate the combinational logic according to the result of the read phase to $\bar{x}_{r+1} = f^G(o) \in \text{Eval}^C(s_r)$.

Write phase Pick a partial resolution $x_{r+1} \in \text{Res}_\text{M}(\bar{x}_{r+1}) \subseteq \text{Write}^C(s_r)$ of the result of the evaluation phase. The successor state is $s_{r+1} = \iota_{r+1} \circ x_{r+1}$.

In each clock cycle, our model determines some successor state of the current state of the circuit; we refer to this as *round*.

Note that due to worst-case propagation of metastability, the evaluation phase is deterministic, while read and write phase are not: Non-determinism in the read phase is required to model the non-deterministic read behavior of masking registers, and non-determinism in the write phase allows copies of metastable bits to stabilize inconsistently. In a physical circuit, metastability may resolve within the combinational logic; we do not model this as a non-deterministic evaluation phase, however, as it is equivalent to postpone possible stabilization to the write phase.

Let C be a circuit in state s_0. For $r \in \mathbb{N}_0$, an *r-round execution (w.r.t. s_0) of C* is a sequence of successor states $s_0, s_1, \ldots, s_r$. We denote by $S_r^C(s_0)$ the set of possible states resulting from r-round executions w.r.t. s_0 of C:

$$S_0^C(s_0) := \{s_0\}, \text{ and} \tag{14}$$

$$S_r^C(s_0) := \left\{ s_r \mid s_r \text{ successor of some } s \in S_{r-1}^C(s_0) \right\}. \tag{15}$$

An *initial state of C w.r.t. input $\iota \in \mathbb{B}_\text{M}^m$* is $s_0 = \iota \circ x_0$. We use $C_r \colon \mathbb{B}_\text{M}^m \to \mathcal{P}(\mathbb{B}_\text{M}^n)$ as a function mapping an input to all possible outputs resulting from r-round executions of C:

$$C_r(\iota) := \left\{ \text{Out}(s_r) \mid s_r \in S_r^C(\iota \circ x_0) \right\}. \tag{16}$$

We say that *r rounds of C implement $f \colon \mathbb{B}_\text{M}^m \to \mathcal{P}(\mathbb{B}_\text{M}^n)$* if and only if $C_r(\iota) \subseteq f(\iota)$ for all $\iota \in \mathbb{B}_\text{M}^m$, i.e., if all r-round executions of C result in an output permitted by f. If there is some $r \in \mathbb{N}$, such that r rounds of C implement f, we say that C implements f.

Observe that our model behaves exactly like a traditional, deterministic, binary circuit model if $s_0 \in \mathbb{B}^{m+k+n}$.

Example. Figure 5 specifies a circuit and its states, as well as the results of the read, evaluation, and write phases. The input registers are I_1 and I_2, the only local register is L_1, and the only output register is O_1. Regarding register types, the input register I_1 is a mask-0 register and all other registers are simple registers.

The initialization is $x_0 = 11$, the input is $\iota = \text{MM}$, and the initial state hence is $s_0 = \iota \circ x_0 = \text{MM11}$, which is indicated in the upper left entry in Figure 5(b). In the read phase, all non-output registers are read. Since I_2 and L_1 are simple registers, their read deterministically evaluates to M and 1, respectively, by the state machine in Figure 3(a). The mask-0 register I_1 in state M may either be read as 0 and remain in state M, or be read as M and transition to state 1, compare Figure 3(b); in this case it does the former. So far, we fixed the outcome of the read phase, 0M1, and the follow-up state of the input registers, MM; the other registers are overwritten at the end of the write phase. The evaluation is uniquely determined, a read phase resulting in o evaluates to $f^G(o)$, here, $f^G(\text{0M1}) = \text{MM}$. We are left with only one more step in this round: The non-input registers are overwritten with some value in the resolution of the evaluation phase's result, in our case with $\text{1M} \in \text{Res}_\text{M}(\text{MM})$. Together we obtain the successor state $s_1 = \text{MM1M}$.

In the next round, I_1 uses the other state transition, i.e., is read as M, and hence has state 1 in the next round. Hence its state remains fixed in all successive rounds by the state machine in Figure 3(b). The other reads are deterministic, so we obtain $o = \text{MM1}$ as the result of the read phase and successor states 1M for I_1 and I_2. The evaluation is $f^G(o) = f^G(\text{MM1}) = \text{MM}$ the state of L_1 and O_1 is overwritten with some value from $\text{Res}_\text{M}(\text{MM})$, here by MM.

By round $r = 2$, the result of the read phase is deterministic because the only masking register stabilized, we read $o = \text{1MM}$, and evaluate to 1M. The remaining non-determinism is whether to write 1M or some stabilization thereof. We examine the case that 10 is written.

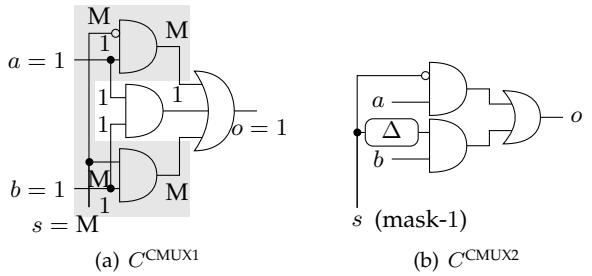
Rounds $r \geq 3$ now are entirely deterministic. The only possible read is 1M1, which evaluates to $f^G(\text{1M1}) = 11$, fixing the result of the write phase to 11. Further rounds are identical, the only metastable register, I_2, remains metastable but has no impact on the evaluation phase as the OR gate always receives input 1 from I_2 and hence masks the metastable input.

Metastability-Containing Multiplexers. We demonstrate the model by developing a CMUX. Despite its simplicity, it demonstrates our concept, and is a crucial part of the more complex metastability-containing components required for the clock synchronization circuit outlined in Section 9.2 [9], [16], [26]. From a broader perspective, this shows that our model, especially the worst-case propagation of metastability, is not "too pessimistic" to permit positive results. We show in Section 5 that it is not "too optimistic," either.

Prior to discussing improved variants, let us examine a standard Multiplexer (MUX). A *(k-bit) Multiplexer (MUX)* is a circuit C with $2k + 1$ inputs, such that C implements

$$f_\text{MUX} \colon \mathbb{B}_\text{M}^k \times \mathbb{B}_\text{M}^k \times \mathbb{B}_\text{M} \to \mathbb{B}_\text{M}^k \tag{17}$$

$$f_\text{MUX}(a, b, s) = \begin{cases} \text{Res}_\text{M}(a) & \text{if } s = 0, \\ \text{Res}_\text{M}(b) & \text{if } s = 1, \text{ and} \\ \mathbb{B}_\text{M} & \text{if } s = \text{M}, \end{cases} \tag{18}$$



(a) C^{CMUX1} (b) C^{CMUX2}

Figure 6. MUX gate-level implementations. Both circuits mask a metastable select bit s in the case of $a = b$ employing additional gates (a) and a masking register (b), respectively. Further, Figure (a) indicates how metastability can be masked.

where we use $k = 1$ for the sake of presentation. In the case of a stable select bit s, it determines whether to output (some stabilization of) a or b. If s is metastable, an arbitrary output may be produced.

A desirable property of a MUX is that if $a = b$, the output is a, regardless of s. Being uncertain whether to select a or b should be insubstantial in this case. If, however, $s = \mathrm{M}$ and $a = b = 1$, a standard implementation with two AND2 and a successive OR2 yields

$$(\neg s \wedge a) \vee (s \wedge b) = (\neg \mathrm{M} \wedge 1) \vee (\mathrm{M} \wedge 1) = \mathrm{M} \vee \mathrm{M} = \mathrm{M}. \quad (19)$$

Hence, we ask for an improved circuit that implements

$$f_{\mathrm{CMUX}}(a, b, s) = \begin{cases} \mathrm{Res}_{\mathrm{M}}(a) & \text{if } s = 0 \text{ or } a = b, \\ \mathrm{Res}_{\mathrm{M}}(b) & \text{if } s = 1, \text{ and} \\ \mathbb{B}_{\mathrm{M}} & \text{if } a \neq b \wedge s = \mathrm{M}. \end{cases} \quad (20)$$

We call such a circuit *(k-bit) Metastability-Containing Multiplexer (CMUX)*. Circuit C^{CMUX1} in Figure 6(a) implements (20): The problematic case of $s = \mathrm{M}$ and $a = b = 1$ is handled by the third AND-gate which becomes 1, providing the OR-gate with a stable 1 as input, see Figure 6(a).

Lemma 4. *$C_1^{CMUX1} \subseteq f_{CMUX}$ from Equation* (20)*.*

We next show how to implement (20) using a masking register in two rounds of computation. Algorithm 1 specifies the clocked circuit by assignments of logic expressions to registers. The trick is to sequentially read s from a mask-1 register, ensuring that at most one copy of s can be metastable. This guarantees that in the case of $s = \mathrm{M}$ and $a = b = 1$, one of the AND-clauses is stable 1.

Algorithm 1 Metastability-Containing Multiplexer.

input: a and b (simple), s (mask-1)
local: s' (simple)
output: o (simple)
each round:
  $s' \leftarrow s$; $o \leftarrow (\neg s \wedge a) \vee (s' \wedge b)$
end

Lemma 5. *Two rounds of Algorithm 1 implement* (20)*.*

One may argue that a direct realization of Algorithm 1 in hardware as a clocked state machine may be too large for practical applications. In fact, however, the algorithm has an optimized unclocked realization, that cannot directly be expressed in our synchronous circuit model: The serialization of assignments in Algorithm 1 ensured by the two clock cycles can also be enforced by local delay constraints instead of clock cycles, see Figure 6(b): if the propagation delay from s to the AND-gate with non-negated s input is larger than the gate delay from s to the AND-gate with negated input $\neg s$, the circuit exhibits the specified behavior. The delay constraint can be enforced by appropriate routing or insertion of inverters. Note that this implementation scales well with increasing bit widths of a and b, since only the select bit needs to be stored in a masking register.

4 Basic Properties

We establish basic properties regarding computability in the model from Section 3. Regarding the implementability of functions by circuits, we focus on two resources: the number $r \in \mathbb{N}$ of rounds and the register types available to it. In order to capture this, let Fun_S^r be the class of functions implementable with r rounds of circuits comprising only simple registers. Analogously, Fun_M^r denotes the class of functions implementable with r rounds that may use masking and simple registers.

First consider the combinational logic. Provided with a partially metastable input x, some gates — those where the collective metastable input ports have an impact on the output — evaluate to M. So when stabilizing x bit by bit, no new metastability is introduced at the gates. Furthermore, once a gate stabilized, its output is fixed; stabilizing the input leads to stabilizing the output.

Lemma 6. *Let G be a combinational logic DAG with m input nodes. Then for all $x \in \mathbb{B}_{\mathrm{M}}^m$,*

$$x' \in \mathrm{Res}_{\mathrm{M}}(x) \Rightarrow f^G(x') \in \mathrm{Res}_{\mathrm{M}}\left(f^G(x)\right). \quad (21)$$

The proof is by inductively applying the definition of a gate to the DAG's nodes.

Stabilizing the input of the combinational logic stabilizes its output. The same holds for the evaluation phase: If one result of the read phase is x and another is $x' \in \mathrm{Res}_{\mathrm{M}}(x)$, the combinational logic stabilizes its output to $f^G(x') \in \mathrm{Res}_{\mathrm{M}}(f^G(x))$. Recall Observations 2 and 3: In state x, simple registers are deterministically read as x, and masking registers as some $x' \in \mathrm{Res}_{\mathrm{M}}(x)$. Hence, the use of masking registers might partially stabilize the input to the combinational logic and, by Lemma 6, its output. The same stabilization can also occur in the write phase. This implies that Write^C is not influenced by the register types.

Lemma 7. *Consider a circuit C in state s. Let C_S be a copy of C that only uses simple registers, and $x = \mathrm{In}(s) \circ \mathrm{Loc}(s)$ the projection of s to the non-output registers. Then*

$$\mathrm{Write}^C(s) = \mathrm{Write}^{C_S}(s) = \mathrm{Res}_{\mathrm{M}}\left(f^G(x)\right). \quad (22)$$

Proof. In C_S, we have $\mathrm{Read}^{C_S}(s) = \{x\}$ by Observation 2. So $\mathrm{Eval}^{C_S}(s) = \{f^G(x)\}$, and $\mathrm{Write}^{C_S}(s) = \mathrm{Res}_{\mathrm{M}}(f^G(x))$ by definition.

In C, it holds that $x \in \mathrm{Read}^C(s)$ by Observation 3, so $\mathrm{Res}_{\mathrm{M}}(f^G(x)) \subseteq \mathrm{Write}^C(s)$. All other reads $x' \in \mathrm{Read}^C(s)$ have $x' \in \mathrm{Res}_{\mathrm{M}}(x)$ by Observation 3, and $f^G(x') \in$

$\mathrm{Res}_{\mathrm{M}}(f^G(x))$ by Lemma 6. It follows that $\mathrm{Write}^C(s) = \mathrm{Res}_{\mathrm{M}}(f^G(x))$. □

Carefully note that the write phase only affects non-input registers; input registers are never written. Hence, Lemma 7 does not generalize to multiple rounds: State transitions of input registers in the read phase affect future read phases.

In 1-round executions, however, masking and simple registers are equally powerful, because their state transitions only affect rounds $r \geq 2$ (we show in Section 6 that these state changes lead to differences for $r \geq 2$ rounds).

Corollary 8. $\mathrm{Fun}_S^1 = \mathrm{Fun}_M^1$.

In contrast, simple and masking registers used as non-input registers behave identically, regardless of the number of rounds: A circuit C in state s_r overwrites them regardless of their state. Since $\mathrm{Write}^C(s_r)$ is oblivious to register types by Lemma 7, so is $\mathrm{Loc}(s_{r+1}) \circ \mathrm{Out}(s_{r+1})$ for a successor state s_{r+1} of s_r.

Corollary 9. *Simple and masking registers are interchangeable when used as non-input registers.*

Consider a circuit C in state s, and suppose $x \in \mathrm{Read}^C(s)$ is read. Since the evaluation phase is deterministic, the evaluation $y = f^G(x) \in \mathrm{Eval}^C(s)$ is uniquely determined by x and C. Recall that we may resolve metastability to $\mathrm{Res}_{\mathrm{M}}(y) \subseteq \mathrm{Write}^C(s)$ in the write phase: The state of an output register R becomes 0 if $y_R = 0$, 1 if $y_R = 1$, and some $b \in \mathbb{B}_{\mathrm{M}}$ if $y_R = \mathrm{M}$. Consequently, output registers resolve independently:

Corollary 10. *For any circuit C, $C_1 = g_0 \times \cdots \times g_{n-1}$, where $g_i \colon \mathbb{B}_{\mathrm{M}}^m \to \{\{0\}, \{1\}, \mathbb{B}_{\mathrm{M}}\}$.*

Proof. Let $s = \iota \circ x_0$ be the initial state of C w.r.t. input ι, and $x = \mathrm{In}(s) \circ \mathrm{Loc}(s)$. By Lemma 7, $\mathrm{Write}^C(s) = \mathrm{Res}_{\mathrm{M}}(f^G(x))$, i.e., $C_1(\iota) = \{\mathrm{Out}(s') \mid s' \in \mathrm{Res}_{\mathrm{M}}(f^G(x))\}$. By definition, $\mathrm{Res}_{\mathrm{M}}(f^G(x)) = \prod_{i \in [n]} \mathrm{Res}_{\mathrm{M}}(f^G(x))_i$. Hence, the claim follows with $g_i(\iota) := \mathrm{Res}_{\mathrm{M}}(f^G(x))_i$ for all $\iota \in \mathbb{B}_{\mathrm{M}}^m$ and $i \in [n]$. □

We show in Section 7 that Corollary 10 generalizes to multiple rounds of circuits with only simple registers. This is, however, not the case in the presence of masking registers, as demonstrated in Section 6.

Lemmas 6 and 7 apply to the input of circuits: Partially stabilizing an input partially stabilizes the possible inputs of the combinational logic, and hence its evaluation and the circuit's output after one round.

Observation 11. *For a circuit C and input $\iota \in \mathbb{B}_{\mathrm{M}}^m$,*

$$\iota' \in \mathrm{Res}_{\mathrm{M}}(\iota) \Rightarrow C_1(\iota') \subseteq C_1(\iota). \tag{23}$$

Proof. Let x_0 be the initialization of C, $s = \iota \circ x_0$ its initial state w.r.t. input ι, and $x = \mathrm{In}(s) \circ \mathrm{Loc}(s)$ the state of the non-output registers; define s' and x' equivalently w.r.t. input $\iota' \in \mathrm{Res}_{\mathrm{M}}(\iota)$. Using Lemmas 6 and 7, and that $\mathrm{Res}_{\mathrm{M}}(x') \subseteq \mathrm{Res}_{\mathrm{M}}(x)$ for $x' \in \mathrm{Res}_{\mathrm{M}}(x)$, we obtain that $\mathrm{Write}^C(s') = \mathrm{Res}_{\mathrm{M}}(f^G(x')) \subseteq \mathrm{Res}_{\mathrm{M}}(f^G(x)) = \mathrm{Write}^C(s)$. □

Finally, note that adding rounds of computation cannot decrease computational power, i.e., result in less functions being implementable, since a circuit determining x in r rounds can be transformed into one using $r+1$ rounds by buffering x for one round. Furthermore, allowing masking registers does not decrease computational power.

Observation 12. *For all $r \in \mathbb{N}_0$, we have $\mathrm{Fun}_S^r \subseteq \mathrm{Fun}_S^{r+1}$, $\mathrm{Fun}_M^r \subseteq \mathrm{Fun}_M^{r+1}$, and $\mathrm{Fun}_S^r \subseteq \mathrm{Fun}_M^r$.*

5 Reality Check

We demonstrated that our model permits the design of metastability-containing circuits. Given the elusive nature of metastability and Marino's impossibility result [28], non-trivial positive results of this kind are surprising, and raise the question whether the proposed model is "too optimistic" to derive meaningful statements about the physical world. Put frankly, a reality check is in order!

In particular, Marino established that no digital circuit can reliably (1) avoid, (2) resolve, or (3) detect metastability [28]. It is imperative that these impossibility results are maintained by any model comprising metastability. We show in Theorem 16 and Corollaries 17–18 that (1)–(3) are impossible in the model proposed in Section 3 as well. We stress that this is about putting the model to the test rather than reproducing a known result.

We first verify that avoiding metastability is impossible in non-trivial circuits. Consider a circuit C that produces different outputs for inputs $\iota \neq \iota'$. The idea is to observe how the output of C behaves while transforming ι to ι' bit by bit, always involving intermediate metastability, i.e., switching the differing bits from 0 to M to 1 or vice versa. This can be seen as a discrete version of Marino's argument for signals that map continuous time to continuous voltage [28]. Furthermore, the bit-wise transformation of ι to ι', enforcing a change in the output in between, has parallels to the classical impossibility of consensus proof of Fischer et al. [13]; our techniques, however, are quite different. The following definition formalizes the step-wise manipulation of bits.

Definition 13 (Pivotal Sequence)**.** *Let $k \in \mathbb{N}_0$ and $\ell \in \mathbb{N}$ be integers, and $x, x' \in \mathbb{B}_{\mathrm{M}}^k$. Then $(x^{(i)})_{i \in [\ell+1]}$, $x^{(i)} \in \mathbb{B}_{\mathrm{M}}^k$, is a* pivotal sequence (from x to x' over $\mathbb{B}_{\mathrm{M}}^k$) *if and only if*

(1) $x^{(0)} = x$ and $x^{(\ell)} = x'$,
(2) for all $i \in [\ell]$, $x^{(i)}$ and $x^{(i+1)}$ differ in exactly one bit, and
(3) this bit is metastable in either $x^{(i)}$ or $x^{(i+1)}$.

For $i \in [\ell]$, we call the differing bit the pivot from i to $i+1$ *and P_i its corresponding* pivotal register.

Carefully note that we do not use pivotal sequences as temporal sequences of non-output register states and circuit successor states; The bit-wise manipulation does not happen over time, instead, we aim at examining closely related circuit states.

We begin with Lemma 14 which applies to a single round of computation. It states that feeding a circuit C with a pivotal sequence x of non-output register states results in a pivotal sequence of possible successor states s of the circuit. Hence, if C is guaranteed to output different results for $x^{(0)}$ and $x^{(\ell)}$, some intermediate element of s must contain a metastable output bit, i.e., there is an execution in which an output register of C becomes metastable. We argue about

successor states rather than just the output because we inductively apply Lemma 14 in Corollary 15.

Lemma 14. *Let C be a circuit, and $(s^{(i)})_{i\in[\ell+1]}$, $s^{(i)} \in \mathbb{B}_{\mathrm{M}}^{m+k+n}$, a pivotal sequence of states of C. Then there is a pivotal sequence $(\hat{s}^{(j)})_{j\in[\ell'+1]}$, $\hat{s}^{(j)} \in \mathbb{B}_{\mathrm{M}}^{m+k+n}$, where each $\hat{s}^{(j)}$ is a successor state of some $s^{(i)}$, satisfying that $\hat{s}^{(0)}$ and $\hat{s}^{(\ell')}$ are successor states of $s^{(0)}$ and $s^{(\ell)}$, respectively.*

Given a pivotal sequence of inputs, there are executions producing a pivotal sequence of attainable successor states. Using these states for another round, Lemma 14 can be applied inductively.

Corollary 15. *Let C be a circuit, x_0 its initialization, and $(\iota^{(i)})_{i\in[\ell+1]}$, $\iota^{(i)} \in \mathbb{B}_{\mathrm{M}}^{m}$, be a pivotal sequence of inputs of C. Then there is a pivotal sequence of states $(s^{(j)})_{j\in[\ell'+1]}$, $s^{(j)} \in \mathbb{B}_{\mathrm{M}}^{m+k+n}$, that C can attain after $r \in \mathbb{N}$ rounds satisfying $s^{(0)} \in S_r^C(\iota^{(0)} \circ x_0)$ and $s^{(\ell')} \in S_r^C(\iota^{(\ell)} \circ x_0)$.*

We wrap up our results in a compact theorem. It states that a circuit which has to output different results for different inputs can produce metastable outputs.

Theorem 16. *Let C be a circuit with $C_r(\iota) \cap C_r(\iota') = \emptyset$ for some $\iota, \iota' \in \mathbb{B}_{\mathrm{M}}^m$. Then C has an r-round execution in which an output register becomes metastable.*

Proof. Apply Corollary 15 to a pivotal sequence from ι to ι' and C, yielding a pivotal sequence y of states that C can attain after r-round executions. Since $C_r(\iota) \ni \mathrm{Out}(s^{(0)}) \neq \mathrm{Out}(s^{(\ell')}) \in C_r(\iota')$, some $\mathrm{Out}(s^{(j)})$ contains an M bit. □

Marino proved that no digital circuit, synchronous or not, can reliably (1) compute a non-constant function and guarantee non-metastable output, (2) detect whether a register is metastable, or (3) resolve metastability of the input while faithfully propagating stable input [28]. Theorem 16 captures (1), and Corollaries 17 and 18 settle (2) and (3), respectively. The key is to observe that a circuit detecting or resolving metastability is non-constant, and hence, by Theorem 16, can become metastable — defeating the purpose of detecting or resolving metastability in the first place.

Corollary 17. *There exists no circuit that implements $f\colon \mathbb{B}_{\mathrm{M}} \to \mathcal{P}(\mathbb{B}_{\mathrm{M}})$ with*

$$f(x) = \begin{cases} \{1\} & \text{if } x = \mathrm{M}, \text{ and} \\ \{0\} & \text{otherwise.} \end{cases} \tag{24}$$

Proof. Assume such a circuit C exists and implements f in r rounds. $C_r(0) \cap C_r(\mathrm{M}) = \emptyset$, so applying Theorem 16 to $\iota = 0$ and $\iota' = \mathrm{M}$ yields that C has an r-round execution with metastable output, contradicting the assumption. □

Corollary 18. *There exists no circuit that implements $f\colon \mathbb{B}_{\mathrm{M}} \to \mathcal{P}(\mathbb{B}_{\mathrm{M}})$ with*

$$f(x) = \begin{cases} \{0,1\} & \text{if } x = \mathrm{M}, \text{ and} \\ \{x\} & \text{otherwise.} \end{cases} \tag{25}$$

Proof. As in Corollary 17 with $\iota = 0$ and $\iota' = 1$. □

In summary, our circuit model (Section 3) is consistent with physical models of metastability, yet admits the computation of non-trivial functions that are crucial in constructing complex metastability-containing circuits [9], [16], [26].

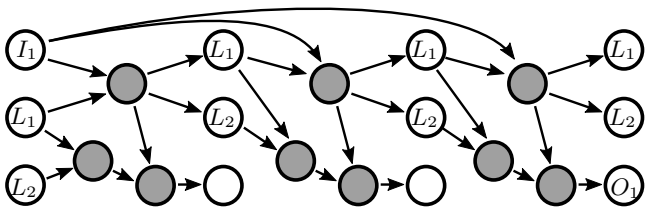


Figure 7. Unrolling three rounds of the circuit in Figure 4 with three gates (gray), and four registers (white). Local registers become fan-out buffers, and early output is ignored.

This gives rise to further questions: (1) Is there a fundamental difference between simple and masking registers? (2) Which functions can be implemented? We study these questions in Sections 6 and 7, respectively.

6 Computational Hierarchy

In this section, we determine the impact of the number of rounds $r \in \mathbb{N}$ and the available register types on the *computational power* of circuits, i.e., the set of functions that are implementable by such circuits. Recall that Fun_S^r denotes the functions implementable using r rounds and simple registers only, and Fun_M^r those implementable using r rounds and arbitrary registers. The main results are:

(1) Even in the presence of metastability, circuits restricted to simple registers can be unrolled (Theorem 19): $\mathrm{Fun}_S^r = \mathrm{Fun}_S^{r+1}$.
(2) With masking registers, however, more functions become implementable with each additional round (Theorem 21): $\mathrm{Fun}_M^r \subsetneq \mathrm{Fun}_M^{r+1}$.

With Corollary 8, we obtain the following hierarchy:

$$\cdots = \mathrm{Fun}_S^2 = \mathrm{Fun}_S^1 = \mathrm{Fun}_M^1 \subsetneq \mathrm{Fun}_M^2 \subsetneq \cdots . \tag{26}$$

We believe this to make a strong case for further pursuing masking registers in research regarding metastability-containing circuits.

Simple Registers. It is folklore that binary-valued synchronous circuits can be unrolled such that the output after $r \in \mathbb{N}$ clock cycles of the original circuit is equal to the output after a single clock cycle of the unrolled circuit. Theorem 19 states that this result also holds in presence of potentially metastable simple registers. Note that — defying intuition — masking registers do not permit this, see Theorem 21.

Theorem 19. *Given a circuit C with only simple registers such that $r \in \mathbb{N}$ rounds of C implement f, one can construct a circuit C' such that one round of C' implements f.*

Proof sketch. Arrange r copies of the combinational logic of C as in Figure 7 such that (1) input registers feed all copies of gates they feed in C, (2) local registers become fan-out buffers (gates forwarding their input), and (3) output registers are copied as well, but only the r-th copy is relevant. We have $C_1' = C_r$ because simple registers merely maintain and propagate metastability in the worst case. □

Naturally, the unrolled circuit can be significantly larger than the original one. However, the point is that adding rounds does not affect the computational power of circuits with simple registers only.

Corollary 20. *For all* $r \in \mathbb{N}$, $\mathrm{Fun}_S^r = \mathrm{Fun}_S^1 =: \mathrm{Fun}_S$.

Arbitrary Registers. For simple registers, additional rounds make no difference in terms of computability — the corresponding hierarchy collapses into Fun_S. In the following, we demonstrate that this is not the case in the presence of masking registers: $\mathrm{Fun}_M^r \subsetneq \mathrm{Fun}_M^{r+1}$ for all $r \in \mathbb{N}$. We demonstrate this using a metastability-containing fan-out buffer specified by Equation (27). It creates r copies of its input bit, at most one of which is permitted to become metastable:

$$f(x) = \begin{cases} \{x^r\} & \text{if } x \neq \mathrm{M}, \\ \bigcup_{i\in[r]} \mathrm{Res}_\mathrm{M}(0^i\mathrm{M}1^{r-i-1}) & \text{otherwise.} \end{cases} \tag{27}$$

Theorem 21. $\mathrm{Fun}_M^r \subsetneq \mathrm{Fun}_M^{r+1}$ *for all* $r \in \mathbb{N}$.

Proof sketch. Pick $2 \leq r \in \mathbb{N}$ and consider f from Equation (27). To see that $f \in \mathrm{Fun}_M^r$, have a circuit C store the input in a mask-0 register, and read one copy of it in each of r rounds. If $x \neq \mathrm{M}$, r rounds of C generate r stable copies of x. Otherwise $x = \mathrm{M}$ and the r outputs are specified by r state transitions of the mask-0 register starting in state M, i.e., behave exactly as specified in (27).

As for $f \notin \mathrm{Fun}_M^{r-1}$, assume $r-1$ rounds of C implement (27) and observe that the input register R can only be read $r-1$ times. Since C produces r outputs, two of these outputs have to depend on the same read of R. If that read operation returns M, which is possible even for masking registers, both outputs can become metastable, violating the specification (27). □

7 The Power of Simple Registers

The design of metastability-containing circuits requires a quick and easy check which metastability-containing components are implementable. In this section, we present such a test for circuits without masking registers.

First, we present sufficient and necessary conditions for a function to be implementable with simple registers only. Using this classification, we demonstrate how to take an arbitrary Boolean function $f\colon \mathbb{B}^m \to \mathbb{B}^n$ and extend it to the most restrictive specification $[f]_\mathrm{M}\colon \mathbb{B}_\mathrm{M}^m \to \mathcal{P}(\mathbb{B}_\mathrm{M}^n)$, the *metastable closure of* f, that is implementable. This is an easy process — one simply applies Definition 24 to f.

The way to make use of this is to start with a function f required as component, "lift" it to $[f]_\mathrm{M}$, and check whether $[f]_\mathrm{M}$ is restrictive enough for the application at hand. If it is, one can work on an efficient implementation of $[f]_\mathrm{M}$, otherwise a new strategy, possibly involving masking registers, must be devised; in either case, no time is wasted searching for a circuit that does not exist.

Since we discuss functions implementable with simple registers only, recall that the corresponding circuits can be unrolled by Theorem 19, i.e., it suffices to understand C_1, a single round of a (possibly unrolled) circuit.

Natural Subfunctions. From Corollary 10 and Observation 11, we know that C_1, the set of possible circuit outputs after a single round, has three properties: (1) its output can be specified bit-wise, (2) each output bit is either 0, 1, or completely unspecified, and (3) stabilizing a partially metastable input restricts the set of possible outputs. Hence C_1 — and by Corollary 20 all circuits using only simple registers — can be represented in terms of bit-wise KV diagrams with values "$0, 1, \mathbb{B}_\mathrm{M}$" instead of "$0, 1, \mathrm{D}$" (D for "don't care"). We call such functions *natural* and show below that $f \in \mathrm{Fun}_S$ if and only if f has a natural subfunction.

Definition 22 (Natural and Subfunctions)**.** *The function* $f\colon \mathbb{B}_\mathrm{M}^m \to \mathcal{P}(\mathbb{B}_\mathrm{M}^n)$ *is* natural *if and only if it is* bit-wise, closed, *and* specific:

Bit-wise *The components* $f_1, \ldots, f_n$ *of* f *are independent:*

$$f(x) = f_1(x) \times \cdots \times f_n(x). \tag{28}$$

Closed *Each component of* f *is specified as either* 0, *as* 1, *or completely unspecified:*

$$\forall x \in \mathbb{B}_\mathrm{M}^m\colon \quad f(x) \in \{\{0\}, \{1\}, \mathbb{B}_\mathrm{M}\}^n. \tag{29}$$

Specific *When stabilizing a partially metastable input, the output of* f *remains at least as restricted:*

$$\forall x \in \mathbb{B}_\mathrm{M}^m\colon \quad x' \in \mathrm{Res}(x) \Rightarrow f(x') \subseteq f(x). \tag{30}$$

For functions $f, g\colon \mathbb{B}_\mathrm{M}^m \to \mathcal{P}(\mathbb{B}_\mathrm{M}^n)$, g *is a* subfunction of f *(we write* $g \subseteq f$*), if and only if* $g(x) \subseteq f(x)$ *for all* $x \in \mathbb{B}_\mathrm{M}^m$.

Suppose we ask whether a function f is implementable with simple registers only, i.e., if $f \in \mathrm{Fun}_S$. Since any (unrolled) circuit C implementing f must have $C_1 \subseteq f$, Corollary 10 and Observation 11 state a necessary condition for $f \in \mathrm{Fun}_S$: f must have a natural subfunction. Theorem 23 establishes that this condition is sufficient, too. For the if-direction, we use a technique introduced in [19] for hazard-free circuits: we cover all prime-implicants of f.

Theorem 23. *Let* $g\colon \mathbb{B}_\mathrm{M}^m \to \mathcal{P}(\mathbb{B}_\mathrm{M}^n)$ *be a function. Then* $g \in \mathrm{Fun}_S$ *if and only if* g *has a natural subfunction.*

Proof. For the only-if-direction, suppose that C is a circuit with only simple registers such that $C_1 \subseteq g$; by Theorem 19, such a circuit exists. C_1 is bit-wise and closed by Corollary 10, and specific by Observation 11. Hence, choosing $f := C_1$ yields a natural subfunction of g.

We proceed with the if-direction. Let $f \subseteq g$ be a natural subfunction of g, and construct a circuit C that implements f. As f is bit-wise, we may w.l.o.g. assume that $n = 1$. If $f(\cdot) = \{0\}$ or $f(\cdot) = \mathbb{B}_\mathrm{M}$, let C be the circuit whose output register is driven by a CONST0-gate; if $f(\cdot) = \{1\}$, use a CONST1-gate. Otherwise, we construct C as follows. Consider $f_\mathbb{B}\colon \mathbb{B}^m \to \{\{0\}, \{1\}\}$ given by

$$f_\mathbb{B}(x) = \begin{cases} \{0\} & \text{if } f(x) = \{0\} \text{ or } f(x) = \mathbb{B}_\mathrm{M}, \text{ and} \\ \{1\} & \text{if } f(x) = \{1\}. \end{cases} \tag{31}$$

Construct C from AND-gates, one for each prime implicant of $f_\mathbb{B}$, with inputs connected to the respective, possibly negated, input registers present in the prime implicant. All AND-gate outputs are fed into a single OR-gate driving the circuit's only output register.

By construction, $C_1(x) = f_\mathbb{B}(x) \subseteq f(x)$ for all $x \in \mathbb{B}^m$. To see $C_1 \subseteq f$, consider $x \in \mathbb{B}_\mathrm{M}^m \setminus \mathbb{B}^m$ and make a case distinction.

(1) If $f(x) = \mathbb{B}_\mathrm{M}$, then trivially $C_1(x) \subseteq f(x)$.
(2) If $f(x) = \{0\}$, we have for all $x' \in \mathrm{Res}(x)$ that $f(x') = f_\mathbb{B}(x') = \{0\}$ by (30). Thus, for each such x', all AND-gate outputs are 0. Furthermore, under input x and for

each AND-gate, there must be at least one input that is stable 0: Otherwise, there would be some $x' \in \mathrm{Res}(x)$ making one AND-gate output 1, resulting in $f_{\mathbb{B}}(x') = \{1\}$. By our definition of gate behavior, this entails that all AND-gates output 0 for all $x' \in \mathrm{Res}_{\mathrm{M}}(x)$ as well, and hence $C_1(x) = \{0\} = f(x)$.

(3) If $f(x) = \{1\}$, all $x' \in \mathrm{Res}(x)$ have $f(x') = f_{\mathbb{B}}(x') = \{1\}$ by (30). Thus, $f_{\mathbb{B}}$ outputs $\{1\}$ independently from the metastable bits in x, and there is a prime implicant of $f_{\mathbb{B}}$ which relies only on stable bits in x. By construction, some AND-gate in C implements that prime implicant. This AND-gate receives only stable inputs from x, and hence outputs a stable 1. The OR-gate receives that 1 as input and, by definition of gate behavior, outputs stable 1. Hence, $C_1(x) = \{1\} = f(x)$.

As f is closed, this case distinction is exhaustive. The claim follows as one round of C implements f. □

Theorem 23 is useful for checking if a circuit without masking registers implementing some function *exists;* its proof is constructive. However, we obtain no non-trivial bound on the *size* of such a circuit — covering all prime implicants can be exponentially costly in m [10]. While efficient metastability-containing implementations exist [9], [26], it is an open question (1) which functions can be implemented efficiently *in general,* and (2) what the overhead for metastability-containment w.r.t. an implementation oblivious to metastability is. However, we show in Section 8 that it is possible to efficiently implement such a circuit with the help of masking registers.

Metastable Closure. We propose a generic method of identifying and creating functions implementable with simple registers. Consider a classical Boolean function $f\colon \mathbb{B}^m \to \mathbb{B}^n$ defined for stable in- and outputs only. Lift the definition of f to $[f]_{\mathrm{M}}$ dealing with (partly) metastable inputs analogously to gate behavior in Section 3: Whenever all metastable input bits together can influence the output, specify the output as "anything in $\mathbb{B}_{\mathrm{M}}$." We call $[f]_{\mathrm{M}}$ the *metastable closure of* f, and argue below that $[f]_{\mathrm{M}} \in \mathrm{Fun}_S$. For $f\colon \mathbb{B}_{\mathrm{M}}^m \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^n)$, i.e., for more flexible specifications, $[f]_{\mathrm{M}}$ is defined analogously.

Definition 24 (Metastable Closure)**.** *For a function $f\colon \mathbb{B}_{\mathrm{M}}^m \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^n)$, we define its* metastable closure $[f]_{\mathrm{M}}\colon \mathbb{B}_{\mathrm{M}}^m \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^n)$ *component-wise for $i \in [n]$ by*

$$[f]_{\mathrm{M}}(x)_i := \begin{cases} \{0\} & \text{if } \forall x' \in \mathrm{Res}_{\mathrm{M}}(x)\colon\ f(x')_i = \{0\}, \\ \{1\} & \text{if } \forall x' \in \mathrm{Res}_{\mathrm{M}}(x)\colon\ f(x')_i = \{1\}, \\ \mathbb{B}_{\mathrm{M}} & \text{otherwise.} \end{cases} \tag{32}$$

We generalize (32) *to Boolean functions. For $f\colon \mathbb{B}^m \to \mathbb{B}^n$, we define $[f]_{\mathrm{M}}\colon \mathbb{B}_{\mathrm{M}}^m \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^n)$ as above, but require $f(x')_i = 0$ and $f(x')_i = 1$, respectively (instead of asking for $\{0\}$ and $\{1\}$).*

By construction, $[f]_{\mathrm{M}}$ is bit-wise, closed, specific, and hence natural.

Observation 25. *$[f]_{\mathrm{M}} \in \mathrm{Fun}_S$ for all $f\colon \mathbb{B}^m \to \mathbb{B}^n$ and for all $f\colon \mathbb{B}_{\mathrm{M}}^m \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^n)$.*

An immediate consequence of Observation 25 for the construction of circuits is that, given an arbitrary Boolean function $f\colon \mathbb{B}^m \to \mathbb{B}^n$, there is a circuit without masking registers that implements $[f]_{\mathrm{M}}$.

For $f\colon \mathbb{B}^m \to \mathbb{B}^n$, Theorem 23 shows that $[f]_{\mathrm{M}}$ is the minimum extension of f implementable with simple registers: by (30) any natural extension g of f must satisfy

$$\forall x \in \mathbb{B}_{\mathrm{M}}^m, \forall i \in [n]\colon \bigcup_{x' \in \mathrm{Res}(x)} f(x')_i \subseteq g(x)_i, \tag{33}$$

and thus $\exists x', x'' \in \mathrm{Res}(x)\colon f(x')_i \neq f(x'')_i \Rightarrow g(x)_i = \mathbb{B}_{\mathrm{M}}$ by (29).

To show that a function is not implementable with simple registers only, it suffices to show that it violates the precondition of Theorem 23, i.e., has no natural subfunction.

Example 26. *Consider $f\colon \mathbb{B}_{\mathrm{M}}^2 \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^2)$ with*

$$f(x) := \mathrm{Res}_{\mathrm{M}}(x) \setminus \{\mathrm{MM}\}. \tag{34}$$

This function specifies to copy a 2-bit input, allowing metastability to resolve to anything except MM*. No circuit without masking registers implements f: $f \notin \mathrm{Fun}_S$.*

The recipe to prove such a claim is: (1) For contradiction, assume $f \in \mathrm{Fun}_S$, i.e., that f has some natural subfunction $g \subseteq f$ by Theorem 23. (2) By specification of f, the individual output bits of g can become metastable for input MM*. (3) Since g is bit-wise, it follows that* $\mathrm{MM} \in g(\mathrm{MM})$*. (4) This contradicts the assumption that $g \subseteq f$.*

8 The Power of Masking Registers

As discussed in Section 6, circuits with masking registers are strictly more powerful than circuits restricted to simple registers. In this section, we show that they can also make a circuit significantly smaller. Let $f\colon \mathbb{B}^m \to \mathbb{B}^n$ be a Boolean function. We propose a generic implementation of its metastable closure $[f]_{\mathrm{M}}$ based on a *Boolean implementation of* f, i.e., a circuit disregarding metastability, but implementing f correctly for stable inputs.

First, recall that the implementation of $[f]_{\mathrm{M}}$ without masking registers implied by Theorem 23 relies on covering all prime implicants of f. The number of prime implicants, however, can be exponential in m, resulting in a circuit that is exponentially larger than one which implements f and is oblivious to metastability. Worse, in [20] unconditional exponential lower bounds were shown on the size of combinational circuits implementing $[f]_{\mathrm{M}}$ for functions f that allow for polynomially-sized circuit implementations. With one round of computation, a circuit in our model computes exactly the same function as its combinational logic, see Lemma 6 and, for a more detailed discussion, [14, Chapter 7]. As circuits without masking registers can be unrolled (Theorem 19), this implies exponential lower bounds on the product of circuit size and the number of rounds of computation for circuits without masking registers in our model. Recall also that masking registers make no difference if we use a single round of computation only, see Corollary 8.

Using masking registers and multiple rounds enables to break this hardness barrier. We propose a clocked implementation based on masking registers that requires only additive linear and logarithmic overheads in the gate count and depth of the combinational logic, respectively; the number of required rounds is $2m + 1$. Thus, this approach is provably exponentially more efficient than any solution without masking registers.

The idea underlying the proposed circuit is to make sufficiently many copies of the input so that the majority of copies is completely stable. As masking registers guarantee that their reads are stable in all but one round, $2m+1$ such copies suffice. As the metastable closure guarantees a stable output only if *all* stabilizations of the input yield the same result (for a given output bit), either evaluating f for each of the stable copies yields the same resulting bit $b \in \mathbb{B}$ or arbitrary output is valid. In the former case, sorting (separately for each output bit) the $2m+1$ computed bits with respect to the order $0 < \mathrm{M} < 1$ and returning the $(m+1)$-th bit yields the correct output. This is exactly what a metastability-containing sorting network does, i.e., a circuit that implements the metastable closure of the sorting function. We discuss this in depth in Section 9, see Lemma 28. However, we only require to sort single bits here; hence simple AND and OR gates implement the $[\min]_{\mathrm{M}}$ and $[\max]_{\mathrm{M}}$, respectively, enabling to directly use standard sorting networks with these gates implementing the 2-sort subcircuits.

Theorem 27. *Let $f\colon \mathbb{B}^m \to \mathbb{B}^n$ be a Boolean function and G a combinational logic DAG with $f^G(x) = f(x)$ for all $x \in \mathbb{B}^m$. Then there is a circuit C that implements $[f]_{\mathrm{M}}$ in $2m+1$ rounds that uses m masking registers, $(2m+1)n$ simple local registers, and n simple output registers. The additive overhead in complexity w.r.t. G is $\mathrm{O}(nm \log m)$ in gate count and $\mathrm{O}(\log m)$ in depth.*

Proof sketch. First suppose that $n = 1$. We propose the following circuit C, which implements $[f]_{\mathrm{M}}$ in $2m+1$ rounds. C has m mask-0 registers $I_1, \dots, I_m$ as input registers, $2m+1$ simple local registers $L_1, \dots, L_{2m+1}$, and one simple output register O. In the r-th round, C performs the following operations:

- It copies, for all $1 \le i \le 2m$, the content of L_i to L_{i+1}.
- It reads the input registers, yielding $x^{(r)} \in \mathrm{Res}_{\mathrm{M}}(\iota)$, where ι is the input, and feeds the result into G, yielding $f^G(x^{(r)})$, and stores it in L_1.
- If feeds the values stored in $L_1, \dots, L_{2m+1}$ into a metastability-containing sorting network (for single bit inputs) and writes the median, i.e., the $(m+1)$-th output bit of the sorting network, to O.

We claim that C implements $[f]_{\mathrm{M}}$ in $2m+1$ rounds. First observe that if $[f]_{\mathrm{M}}(\iota) = \mathbb{B}_{\mathrm{M}}$, any output of C is feasible. Hence, suppose $[f]_{\mathrm{M}}(\iota) = \{b\}$ for some $b \in \mathbb{B}$ in the following.

Recall that each masking register can be read as M at most once in any $(2m+1)$-round execution, cf. Figure 3(b). Hence, there must be $m+1$ rounds in which all reads are stable, i.e., where $x^{(r)} \in \mathbb{B}^m$. As $[f]_{\mathrm{M}}(\iota) = \{b\}$, $f(x^{(r)}) = b$. As $f^G(x) = f(x)$ for $x \in \mathbb{B}^m$ by assumption, in these rounds we have $f^G(x^{(r)}) = b$. It follows that at least $m+1$ of the local registers L_i hold the bit b after $2m+1$ rounds. We claim that — after sorting — the $(m+1)$-th bit is b and is copied to O. To see this, consider the closure $[s]_{\mathrm{M}}$ of the (1-bit) sorting function $s\colon \mathbb{B}^m \to \mathbb{B}^n$. The i-th output is 0 if there are at least i inputs that are 0, it is 1 if there are $n-i-1$ inputs that are 1, and M otherwise — in the latter case, stabilizing all M inputs to either 0 or 1 would result in different values of output i, in the first two cases it would not. Hence, having at least $m+1$ of the registers L_i hold value b implies that the $(m+1)$-th output bit of the sorting network is b. We conclude that C behaves as claimed.

It remains to bound the number of additional gates and the increase in depth of the combinational logic. Using a pair of AND and OR gates as a 2-sort element and optimal sorting networks [1], the sorting network contains $O(m \log m)$ gates and has depth $O(\log m)$. This completes the proof for the special case of $n = 1$. To conclude the proof, we observe that the above construction directly generalizes to arbitrary n — Definition 24 allows us to treat the output bits independently. □

Observe that the above construction can be specialized to implementing $[f]_{\mathrm{M}}$ only for inputs satisfying that at most $k \le m$ bits in the input are metastable. In this case, it is sufficient to read the input $2k+1$ times only, meaning that m is replaced by k in all of the above complexity bounds, i.e., the overheads become $\mathrm{O}(nk \log k)$ in the gate count and of $\mathrm{O}(\log k)$ in depth. This is to be contrasted with the construction from [20], which implements $[f]_{\mathrm{M}}$ for inputs with at most k metastable bits without masking registers and in a single round, but at the cost of increasing the number of gates by a factor exceeding $\binom{ne}{k}^{2k}$, i.e., a *multiplicative* blow-up in circuit size that is *exponential* in k.

9 Components for Clock Synchronization

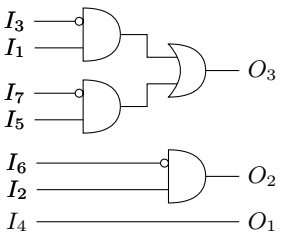
This section demonstrates the power of our techniques: We establish that a variety of metastability-containing components are a reality. Due to the machinery established in the previous sections, this is possible with simple checks (usually using Observation 25). The list of components is by no means complete, but already allows implementing a highly non-trivial application.

We are the first to demonstrate the implementability of the fault-tolerant clock synchronization algorithm by Lundelius Welch and Lynch [27] in hardware, with deterministic correctness guarantee, despite the unavoidable presence of metastable upsets. This algorithm is widely applied, e.g., applied in the Time-Triggered Protocol (TTP) [25] and in FlexRay [6]. While the software–hardware based implementations of TTP and FlexRay achieve a precision in the order of microseconds, higher operating frequencies ultimately require a pure hardware implementation. Recently, an implementation based on an FPGA has been presented by Kinali et al. [22]. All known implementations, however, synchronize potentially metastable inputs *before* computations — a technique that becomes less reliable with increasing operating frequencies, since less time is available for metastability resolution.

Moreover, classical bounds for the MTBF for metastable upsets assume a uniform distribution of input transitions; this is not guaranteed to be the case in clock synchronization, since the goal is to align clock ticks. Either way, synchronizers do not deterministically guarantee stabilization, and errors are bound to happen eventually when n clocks take $n(n-1)$ samples at, e.g., 1 GHz. Ever-increasing operating frequencies and the inevitability [28] of metastable upsets when measuring relative timing deviations lead us to a fundamental question: Does the unavoidable presence of

dec	O_1	O_2	O_3
0	0	0	0
1	0	0	1
2	0	1	1
3	0	1	0
4	1	1	0
5	1	1	1
6	1	0	1
7	1	0	0

(a) 3-bit BRGC



(b) Transformation circuit

Figure 8. Efficient TC-to-BRGC conversion.

metastable upsets pose a principal limit on the operating frequency? We show that this is not the case.

We prepare by arguing that the right encoding is crucial in Section 9.1 and present the algorithm and the required components in Section 9.2.

9.1 Encoding and Precision

An appropriate encoding is key to designing metastability-containing arithmetic components. If, for example, a control bit u indicating whether to increase $x = 7$ by 1 is metastable, and x is encoded in binary, the result must be a metastable superposition of 00111 and 01000, i.e., anything in $\mathrm{Res}(0\mathrm{MMMM})$ and thus an encoding of any number $x' \in [16]$ — even after resolving metastability! The original uncertainty between 7 and 8 is massively amplified; a good encoding should *contain* the uncertainty imposed by $u = \mathrm{M}$.

Formally, a *code* is an injective function $\gamma\colon [n] \to \mathbb{B}^k$ mapping a natural number $x \in [n]$ to its encoded representation. For $y = \gamma(x)$, we define $\gamma^{-1}(y) := x$, and for sets X, $\gamma(X) := \{\gamma(x) \mid x \in X\}$ and $\gamma^{-1}(X) := \{x \mid \gamma(x) \in X\}$. In this work, we consider two encodings for input and output: TC and BRGC. For the 4-bit (unary) TC we use $\mathrm{un}\colon [5] \to \mathbb{B}^4$ with $\mathrm{un}(1) = 0001$ and $\mathrm{un}^{-1}(0111) = 3$; $\mathrm{un}^{-1}(0101)$ does not exist. BRGC, compare Figure 8(a), is represented by $\mathrm{rg}(x)$, and is much more efficient, using only $\lceil \log_2 n \rceil$ bits. In fact, $\mathrm{rg}\colon [2^k] \to \mathbb{B}^k$ is bijective.

We choose un and rg due to the property that in both encodings, for $x \in [k-1]$, $\gamma(x)$ and $\gamma(x+1)$ differ in a single bit only. This renders them suitable for metastability-containing operations. We revisit the above example with the metastable control bit u indicating whether to increase $x = 7$ by 1. In BRGC, 7 is encoded as 00100 and 8 as 01100, so their metastable superposition resolves to $\mathrm{Res}(0\mathrm{M}100)$, i.e., only to 7 or 8. Since the original uncertainty was whether or not to increase $x = 7$ by 1, the uncertainty is perfectly contained instead of amplified as above. We formalize the notion of the amount of uncertainty in a partially metastable code word: $x \in \mathbb{B}_{\mathrm{M}}^k$ has *precision-p (w.r.t. the code γ)* if

$$\max\left\{y - \bar{y} \mid y, \bar{y} \in \gamma^{-1}(\mathrm{Res}(x))\right\} \leq p, \tag{35}$$

i.e., if the largest possible difference between resolutions of x is bounded by p. The precision of x w.r.t. γ is undefined if some $y \in \mathrm{Res}(x)$ is no code word, which is not the case in our application.

Note that the arithmetic components presented below make heavy use of BRGC. This makes them more involved, but they are exponentially more efficient than their TC counterparts in terms of memory and avoid the amplification of uncertainties incurred by standard binary encoding. As a matter of fact, recently proposed efficient implementations for metastability-containing sorting networks [9], [26] and metastability-containing TDC [16] use BRGC.

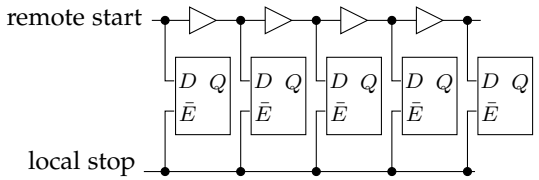


Figure 9. Tapped delay line TDC. Outputs 1^k0^{n-k} or $1^k\mathrm{M}0^{n-k-1}$, i.e., at most one metastable bit, and hence has precision-1.

9.2 Algorithm and Components

Our core strategy is the *separation of concerns* outlined in Section 1 and Figure 1. The key is that the digital part of the circuit can become metastable, but that metastability is *contained* and ultimately translated into *bounded* fluctuations in the analog world, not contradicting Marino.

We propose an implementation for n clock synchronization nodes with at most $f < n/3$ faulty nodes, in which each node does the following.

Step 1: Analog to Digital. First, we step from the analog into the digital world: Delays between $n-1$ remote pulses and the local pulse are measured with TDC. The measurement can be realized such that at most one of the output bits, accounting for the difference between x and $x+1$ ticks, becomes metastable, i.e., has precision-1.

TDC can be implemented using tapped delay lines or Vernier delay line TDC [18], [31], [32]; see Figure 9: A line of delay elements is tapped in between each two consecutive elements, driving the data input port of initially enabled latches initialized to 0. The rising transition of the remote clock signal fed into the delay line input then passes through the line, and sequentially sets the latches to 1; the rising transition of the local clock signal is used to disable all latches at once. After that, the delay line's latches contain the time difference as unary TC. Choosing the propagation delays between the latches larger than their setup/hold times, we ensure that at most one bit is metastable, i.e., their status is of the form 1^*0^* or $1^*\mathrm{M}0^*$. The output is hence a precision-1 TC-encoded time difference.

A traditional implementation would use synchronizers on the TDC outputs. This delays the computation and encourages stabilization, but does not enforce it. However, clock synchronization cannot afford to wait. Furthermore, we prefer guaranteed correctness over a probabilistic statement: Four nodes, each sampling at 1 GHz, sample $1.2 \cdot 10^{10}$ incoming clock pulses per second; synchronizers cannot provide sufficiently small error probabilities when allocating 1 ns or less for metastability resolution [5].

Step 2: Encoding. We translate the time differences into BRGC, making storage and subsequent components much more efficient. The results are BRGC-encoded time differences with at most one metastable bit of precision-1.

With the example circuit in Figure 8, we show how precision-1 TC-encoded data can be efficiently translated into precision-1 BRGC-encoded data. Figure 8(b) depicts the

circuit that translates a 7-bit TC into a 3-bit BRGC; note that gate count and depth are optimal for a fan-in of 2. The circuit can be easily generalized to n-bit inputs, having a gate depth of $\lfloor \log_2 n \rfloor$. While such translation circuits are well-known, it is important to check that the given circuit fulfills the required property of preserving precision-1: This holds as each input bit influences exactly one output bit, and, due to the nature of BRGC, this bit makes exactly the difference between $\mathrm{rg}(x)$ and $\mathrm{rg}(x+1)$ given a TC-encoded input of $1^x\mathrm{M}0^{7-x-1}$.

A more efficient way is to use a metastability-containing TDC which directly produces BRGC of precision-1; such a component is presented in [16].

Step 3: Sorting Network. A sorting network selects the $(f+1)$-th and $(n-f)$-th largest remote-to-local clock differences (tolerating f faults requires to discard the smallest and largest f values).

This requires 2-sort building blocks that pick the minimum and maximum of two precision-1 BRGC-encoded inputs preserving precision-1, which can then be combined using well-known sorting networks [1], [2]. We show that $\max$ (analogously $\min$ and hence a 2-sort) of two precision-1 k-bit BRGC numbers is implementable without masking registers, such that each output has precision-1. Observe that this is straightforward for TC-encoded inputs with bitwise AND and OR for $\min$ and $\max$, respectively. We show, however, that this is possible for BRGC inputs as well; efficient implementations of the proposed 2-sort building blocks are presented in [9], [26].

Lemma 28. *We define the function* $\max_{BRGC}\colon \mathbb{B}^k \times \mathbb{B}^k \to \mathbb{B}^k$ *by* $\max_{BRGC}(x,y) := \mathrm{rg}(\max\{\mathrm{rg}^{-1}(x), \mathrm{rg}^{-1}(y)\})$. *Then* $[\max_{BRGC}]_{\mathrm{M}} \in \mathrm{Fun}_S$ *and it determines precision-*1 *output from precision-*1 *inputs* x *and* y.

An analogous statement holds for $\min_{\mathrm{BRGC}}(x,y) := \mathrm{rg}(\min\{\mathrm{rg}^{-1}(x), \mathrm{rg}^{-1}(y)\})$. Efficient implementations are given in [9], [26] and improved in [15].

Step 4: Decoding and Digital to Analog. The BRGC-encoded $(f+1)$-th and $(n-f)$-th largest remote-to-local clock differences are translated back to TC-encoded numbers. This can be done preserving precision-1: A BRGC-encoded number of precision-1 has at most one metastable bit: For any up-count from (an encoding of) $x \in [2^k-1]$ to $x+1$, a single bit changes, which thus can become metastable if it has precision-1. It is possible to preserve this guarantee when converting to TC.

Lemma 29. *Define* $\mathrm{rg2un}\colon \mathbb{B}^k \to \mathbb{B}^{(2^k-1)}$ *as* $\mathrm{rg2un}(x) := \mathrm{un}(\mathrm{rg}^{-1}(x))$. *Then* $[\mathrm{rg2un}]_{\mathrm{M}} \in \mathrm{Fun}_S$ *converts its parameter to TC, preserving precision-*1.

Finally, we step back into the analog world, again without losing precision: The two values are used to control the local clock frequency via a Digitally Controlled Oscillator (DCO). However, the DCO design must be chosen with care. Designs that switch between inverter chains of different length to modify the frequency of a ring oscillator cannot be used, as metastable switches may occur exactly when a pulse passes. Instead, we use a ring oscillator whose frequency is controlled by analog effects such as changes in inverter load or bias current, see e.g. [11], [30], [39]. While the at most two metastable control bits may dynamically change the load of two inverters, this has a limited effect on the overall frequency change and does not lead to glitches within the ring oscillator.

Carefully note that this gives a *guaranteed end-to-end uncertainty of a single bit* through all digital computations.

10 Conclusion

No digital circuit can reliably avoid, detect, or resolve metastable upsets [28]. So far, the only known counter strategy has been to use synchronizers — trading time for an increased probability of resolving metastability. We propose a fundamentally different method: It is possible to design efficient digital circuits that tolerate a certain degree of metastability in the input. This technique features critical advantages:

1) Where synchronizers decrease the odds of failure, our techniques provide deterministic guarantees. A synchronizer may or may not stabilize in the allotted time frame. Our model, on the other hand, guarantees to return one of a specific set of known values — like the metastable closure, but this depends on the application — without relying on probabilities.

2) Our approach avoids synchronization delay and, in principle, allows higher operating frequencies. If the required functions can be implemented in a metastability-containing way, there is no need to use a synchronizer, i.e., to wait a fixed amount of clock cycles before starting the computation.

3) Even if metastability needs to be resolved eventually, one can still save time by allowing for stabilization *during* the metastability-containing computations. In light of these properties, we expect our techniques to prove useful for a variety of applications, especially in time- and mission-critical scenarios.

As a consequence of our techniques, we are the first to establish the implementability of the fault-tolerant clock synchronization algorithm by Lundelius Welch and Lynch [27] with a deterministic correctness guarantee, despite the unavoidable presence of metastable upsets.

Furthermore, we fully classify the functions computable with circuits restricted to standard registers. Finally, we show that circuits with masking registers become computationally more powerful with each round, resulting in a non-trivial hierarchy of computable functions.

Future Work. In this work, we focus on computability under metastable inputs. There are many open questions regarding circuit complexity in our model of computation. It is of interest to reduce the gate complexity and latency of circuits, as well as to determine the complexity overhead of metastability-containment in general. While [20] proved the existence of functions with exponential overhead in case of simple registers, efficient circuits have been obtained for sorting networks [9], [26], a TDC that directly produces precision-1 BRGC [16], and network-on-chip routers [33].

Acknowledgments. The authors thank Christian Ikenmeyer, Attila Kinali, Ulrich Schmid, and Andreas Steininger for many fruitful discussions. Matthias Függer was affiliated with Max Planck Institute for Informatics during the research regarding this paper. This project has received

funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 716562), and from the SIC project (grant P26436-N30) of the Austrian Science Fund.

REFERENCES

[1] M. Ajtai, J. Komlós, and E. Szemerédi. An O(n log n) sorting network. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 1–9, 1983.

[2] K. E. Batcher. Sorting networks and their applications. In *American Federation of Information Processing Societies*, pages 307–314, 1968.

[3] S. Beer and R. Ginosar. Eleven ways to boost your synchronizer. *IEEE Transactions on VLSI Systems*, 23(6):1040–1049, 2015.

[4] S. Beer, R. Ginosar, J. Cox, T. Chaney, and D. M. Zar. Metastability challenges for 65nm and beyond: simulation and measurements. In *Design, Automation and Test in Europe*, pages 1297–1302, 2013.

[5] S. Beer, R. Ginosar, M. Priel, R. R. Dobkin, and A. Kolodny. The devolution of synchronizers. In *IEEE International Symposium on Asynchronous Circuits and Systems*, pages 94–103, 2010.

[6] 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, 2003. EP Patent App. EP20,020,008,171.

[7] J. A. Brzozowski, Z. Ésik, and Y. Iland. Algebras for hazard detection. In *ISMVL*, page 3, 2001.

[8] J. A. Brzozowski and M. Yoeli. On a ternary model of gate networks. *IEEE Transactions on Computers*, C-28(3):178–184, 1979.

[9] J. Bund, C. Lenzen, and M. Medina. Near-optimal metastability-containing sorting networks. In *Design, Automation, and Test in Europe*, pages 226–231, 2017.

[10] A. K. Chandra and G. Markowsky. On the number of prime implicants. *Discrete Mathematics*, 24(1):7–11, 1978.

[11] V. De Heyn, G. Van der Plas, J. Ryckaert, and J. Craninckx. A fast start-up 3ghz–10ghz digitally controlled oscillator for uwb impulse radio in 90nm cmos. In *European Solid State Circuits Conference*, pages 484–487, 2007.

[12] E. B. Eichelberger. Hazard detection in combinational and sequential switching circuits. *IBM Journal of Research and Development*, 9(2):90–99, 1965.

[13] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[14] S. Friedrichs. *Metastability-containing circuits, parallel distance problems, and terrain guarding*. PhD thesis, Universität des Saarlandes, Postfach 151141, 66041 Saarbrücken, 2017.

[15] S. Friedrichs and A. Kinali. Efficient metastability-containing multiplexer. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 332–337, 2017.

[16] M. Függer, A. Kinali, C. Lenzen, and T. Polzer. Metastability-aware memory-efficient time-to-digital converters. In *23rd IEEE International Symposium on Asynchronous Circuits and Systems*, 2017.

[17] R. Ginosar. Fourteen ways to fool your synchronizer. In *International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 89–97, 2003.

[18] C. Gray, W. Liu, W. Van Noije, J. Hughes, T.A., and R. Cavin. A sampling technique and its cmos implementation with 1 gb/s bandwidth and 25 ps resolution. *IEEE Journal of Solid-State Circuits*, 29(3):340–349, 1994.

[19] D. A. Huffman. The design and use of hazard-free switching networks. *Journal of the ACM*, 4(1):47–62, 1957.

[20] C. Ikenmeyer, B. Komarath, C. Lenzen, V. Lysikov, A. Mokhov, and K. Sreenivasaiah. On the complexity of hazard-free circuits. *arXiv preprint arXiv:1711.01904*, 2017.

[21] International technology roadmap for semiconductors. http://www.itrs2.net/, 2013.

[22] A. Kinali, F. Huemer, and C. Lenzen. Fault-tolerant clock synchronization with high precision. In *IEEE Symposium on VLSI (ISVLSI)*, pages 490–495, 2016.

[23] D. J. Kinniment. *Synchronization and Arbitration in Digital Systems*. Wiley, 2008.

[24] D. J. Kinniment, A. Bystrov, and A. V. Yakovlev. Synchronization circuit performance. *IEEE Journal of Solid-State Circuits*, 37(2):202–209, 2002.

[25] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

[26] C. Lenzen and M. Medina. Efficient metastability-containing gray code 2-sort. In *IEEE International Symposium on Asynchronous Circuits and Systems*, 2016.

[27] J. Lundelius Welch and N. A. Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and Computation*, 77(1):1–36, 1988.

[28] L. R. Marino. General theory of metastable operation. *IEEE Transactions on Computers*, 30(2):107–115, 1981.

[29] M. Mendler, T. R. Shiple, and G. Berry. Constructive boolean circuits and the exactness of timed ternary simulation. *Formal Methods in System Design*, 40(3):283–329, 2012.

[30] T. Olsson and P. Nilsson. A digitally controlled pll for soc applications. *IEEE Journal of Solid-State Circuits*, 39(5):751–760, 2004.

[31] T. Rahkonen and J. T. Kostamovaara. The use of stabilized cmos delay lines for the digitization of short time intervals. *IEEE Journal of Solid-State Circuits*, 28(8):887–894, 1993.

[32] G. W. Roberts and M. Ali-Bakhshian. A brief introduction to time-to-digital and digital-to-time converters. *IEEE Transactions on Circuits and Systems*, 57-II(3):153–157, 2010.

[33] G. Tarawneh, M. Függer, and C. Lenzen. Metastability tolerant computing. In *23rd IEEE International Symposium on Asynchronous Circuits and Systems*, 2017.

[34] G. Tarawneh and A. Yakovlev. An RTL method for hiding clock domain crossing latency. In *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 540–543, 2012.

[35] G. Tarawneh, A. Yakovlev, and T. S. T. Mak. Eliminating synchronization latency using sequenced latching. *IEEE Transactions on VLSI Systems*, 22(2):408–419, 2014.

[36] S. H. Unger. Hazards and delays in asynchronous sequential switching circuits. *IRE Transactions on Circuit Theory*, 6(1):12–25, 1959.

[37] H. J. Veendrick. The behaviour of flip-flops used as synchronizers and prediction of their failure rate. *IEEE Journal of Solid-State Circuits*, 15(2):169–176, 1980.

[38] A. V. Yakovlev, M. Kishinevsky, A. Kondratyev, and L. Lavagno. OR causality: Modelling and hardware implementation. In *15th International Conference on Application and Theory of Petri Nets*, pages 568–587, 1994.

[39] J. Zhao and Y.-B. Kim. A 12-bit digitally controlled oscillator with low power consumption. In *51st Midwest Symposium on Circuits and Systems*, pages 370–373, 2008.

Stephan Friedrichs completed his bachelor's and master's degrees in computer science at the TU Braunschweig, Germany, in 2008 and 2012, respectively. As a Ph. D. student at the Max-Planck-Institute for Informatics he received his doctorate from the Faculty of Mathematics and Computer Science of the Saarland University, Germany, in 2017.

Matthias Függer received his M.Sc. (2006) and his PhD (2010) in computer engineering from TU Wien, Austria. He worked as an assistant professor at TU Wien, and as a post-doctoral researcher at LIX, Ecole polytechnique and at Max-Planck-Institut für Informatik. Currently, he is a CNRS researcher at LSV, ENS Paris-Saclay, and co-leading the Digicosme group HicDies-Meus on Highly Constrained Discrete Agents for Modeling Natural Systems.

Christoph Lenzen received a diploma degree in mathematics from the University of Bonn in 2007 and a Ph. D. degree from ETH Zurich in 2011. After postdoc positions at the Hebrew University of Jerusalem, the Weizmann Institute of Science, and MIT, he became group leader at MPI for Informatics in 2014. He received the best paper award at PODC 2009, the ETH medal for his dissertation, and in 2017 an ERC starting grant.