Habilitation à diriger des recherches

université PARIS-SACLAY

Computing at the border of abstractions: the power of timed, non-binary, distributed circuits

Habilitation à diriger des recherches de l'Université Paris-Saclay

présentée et soutenue à Gif-sur-Yvette, le 6. Jan. 2022, par

Matthias FÜGGER

Composition du jury

Miloš KRSTIĆ Prof., University Potsdam, IHP, Germany	Rapporteur
Yoram MOSES Prof., Technion, Israel	Rapporteur
Alex YAKOVLEV Prof., Newcastle University, GB	Rapporteur
Stefan HAAR Senior Researcher, INRIA, LMF, ENS Paris-Saclay, Université Paris-Saclay, France	Examinateur
Maria POTOP-BUTUCARU Prof., Sorbonne Université, LIP6, France	Examinatrice

To Sophie

Abstract

The design and analysis of low-level computing devices directly implemented in hardware is commonly based on finite state machine models. In this work we review some of the assumptions made in these designs and discuss techniques for the cases where the assumptions fail to hold. The work concentrates on three such aspects: (i) computing under conditions where low-level timing effects cannot be neglected, (ii) when non-binary signals play a central role, and (iii) computing within a changing infrastructure, i.e., dynamic networks. While most of this work is devoted to implementations in silicon, microbiological circuits are discussed in the outlook.

Acknowledgements

I'd like to start this thesis with a thank you to the many who have made this not only happen, but an enjoyable and interesting journey together. It would be impossible to mention all who contributed and all the great experiences along this path and so I apologize upfront for those not mentioned in this really short summary – rest assured you are not forgotten.

I'd like to begin with the members of the habilitation committee, who foremost contributed by their kind willingness, effort, and time to review and assess this thesis and the defense. I am grateful to my reviewers Miloš Krstić, Yoram Moses, and Alex Yakovlev, as well as my defense members Stefan Haar and the president of the jury Maria Potop-Butucaru for their time and energy. I am well aware that this requires considerable effort besides the already heavily loaded days (and nights). Many, many thanks! I would additionally like to thank Stefan Haar, not only for the many helpful and nice discussion and coffees we had during my time at LSV and later LMF, but also for being so kind to be my habilitation patron. I'd also like to thank Benedikt Bollig for his advises, proofreading, and setting up the defense's technical environment and Nicole Bidoit-Tollu for her help.

Following up with my studies and PhD time at TU Wien, I'd first like to mention Andreas Steininger. He immediately sparked my interest in how circuits work and how to design good ones, not only with his catching lectures, but also with his open-minded and searching attitude. I was more than happy to write my bachelor thesis under his and Babak Rahbaran's supervision, which I greatly enjoyed. My wish to become a member of the ECS group, where he was working, was all the more clear, when I got to know Ulrich Schmid in his impressive course on distributed systems. I clearly remember the way he encouraged us to contribute to the current state-of-the-art in a homework. Working under his supervision on my Master's thesis and later on my PhD has highly influenced not only the way I see science, but also the way I see ideal teams – it was an open, kind, and fun atmosphere with lots of discussions among team members. Thank you also for all the mentoring chats! Besides the many fruitful and proving-until-late-night sessions I enjoyed so much at his group, I'd also like to mention the after-bouldering and photo sessions with Heinz Deinhart, Alex Koessler, and Martin Perner, the circuit design and helicopter sessions with Gottfried Fuchs and Peter Tummeltshammer, the philosophical and distributed computing sessions with Thomas Nowak and Josef Widder, the ski touring and blackboard sessions with Robert Najvirt, the biking and chats about asynchronous designs with Florian Huemer, the work and nice chats with Manfred Schwarz, Jürgen Maier, Thomas Polzer, and Kyrill Winkler, and Traude Sommer, who is always there, be it to solve something really urgent and seemingly impossible with the greatest ease or for a nice chat over cafe. The list is far from complete, though. All in all, I am greatly in dept to Ulrich, Andreas, Traude, and the many I met there during the years at ECS, for the inspiring and great time – it is always a pleasure to visit you!

I would also like to express my deep gratitude to Bernadette Charron-Bost, for all the enlightening discussions and the new view on distributed systems I learned during my stays with her at École polytechnique. It was also during this time in Paris, that I had the great pleasure to work with Thomas Nowak and Jennifer L. Welch. Many thanks Thomas, for the many amazing and interesting walks discussing distributed computing, circuits, and science! It was clearly this experience in Paris along with Bernadette Charron-Bost's mentoring and way to approach problems that made me want to apply for a position soon after.

I am also deeply thankful to Christoph Lenzen and Kurt Mehlhorn for my time at Max-Planck-Institut für Informatik. I had a great time at Saarbrücken with again new views to learn and the opportunity to work with a great team without boundaries between circuits and distributed algorithms. Many thanks to the MPI team! In particular I'd like to thank Christoph Lenzen for the great black-board and after-work sessions, Kurt Mehlhorn for his advises and his "all-is-possible" attitude to science without boundaries, Stephan Friedrichs, Attila Kinali, Moti Medina, Johannes Bund, and Michael Dirnberger for the great work sessions often followed by long walks, bouldering, guitar playing, discovering hidden places, and the fun long-distance chats and visits, as well as Christina Fries, Ingrid Finkler-Paul, and Alexandra Klasen-Schmitt for the great support and nice chats, during the time at MPI-INF and whenever I visit. Again, this list is far from complete and I'd like to thank all others I had the pleasure to work with.

I would like to thank Bernadette Charron-Bost for drawing my attention to CNRS, the members

of the former LSV at ENS Cachan for their warm welcome, and the members of the new LMF for the many great moments together. In particular I would like to thank Stéphane Demri, Stefan Haar, Hubert Comon-Lundh, Laurent Fribourg, and Serge Haddad for their help and advises; this really helped me a lot! I'd also like to thank Stefan Schwoon, Béatrice Bérard, and Benedikt Bollig with whom I had the great pleasure to work together in a room, as well as Stefan Göller for his enthusiasm on basic and difficult problems, and Dietmar Berwanger for the discussions and his aesthetic way to state problems. I am grateful to Patricia Bouyer-Decitre for all the energy and effort she devotes to creating and continuously improving the LMF, as well as Marie-France Grandisson and Eugénie Mery for their continuous help. It was several times that Marie-France wrote in the middle of a weekend to push a project further, thank you! I greatly enjoy to be part of this team – thank you all!

There are many colleagues whom I had the pleasure to discuss and work with at conferences, research visits, and common lectures – thanks a lot for the great moments! Additionally to those already mentioned, I'd like to mention Danny Dolev and Ian W. Jones, as well as the ASYNC community: I really enjoy the conferences and meetings with sharp criticism and insightful discussions from diverse scientific viewpoints within a nice atmosphere. I'd also like to mention two researchers from a domain I got fascinated with. Thank you Alfonso Jaramillo and Manish Kushwaha for the great and interesting discussions, and Manish for the fascinating problems I have the pleasure to work on with you and Thomas.

I'd like to dedicate a particular huge thanks to three researchers and friends from whom I have learned a lot, with lasting influence on my view on many scientific problems and research in general. I deeply want to thank Benedikt, Markus, and Thomas for the many occasions of inspiring chats at the university, on the route up to the university, during long walks (even in pouring rain), at cafes, and many other places. Thank you also for always being there, be it to help with a difficult problem or a walk or call in the middle of the night to discuss an urgent proof fix or a crazy idea – I am proud and very much looking forward to traveling to new destinations with you!

I'd also like to express my gratitude to my family: my parents Michaela and Reinhold, to Barbara and Tapas, Samuel and Rafael (it is a pleasure to watch their curiosity for biology), to Hanna and Simon, to Chri and Andrea, and to Klara and Josef. Thanks a lot for your never ending support throughout all the years at numerous occasions! I'd also like to thank Marie-Pierre and Jean-Marc for their warm support and help.

Equally, I'd like to thank my friends Andreas, Gottfried, Robert, Mario, Tanja, Wolfi, and Michi for their support during the years and for always being there.

The last paragraph – and the thesis – is dedicated to Sophie, whom I deeply thank for our journey together and also for her open-minded approach to science and life that I find truly inspiring! Merci!

Contents

Chapter 1

Introduction

A central aspect of computation is mechanically, in the sense that this process follows rules, transforming binary input words into binary output words, such that input and output are related via a predefined function. The rules of the mechanical transformation naturally depend on the target machinery. Two abstract machineries are typically studied within this context.

(i) The Turing machine, an abstract machine, reads from and writes to an infinite, linear, sequentially accessible tape, and is controlled by a finite state machine. The input is presented to the machine as the tape's initial state and the output contained on the tape when the state machine enters a dedicated final state.

(ii) Circuits that comprise of untimed Boolean gates. The input is represented as respective constant signals, and the output is obtained as the signal values of dedicated gate outputs. These Boolean circuits are restricted to be purely combinational, i.e., gates are stateless, and no cycles are allowed when connecting gates.

While a Turing machine specifies an input-output relation for a potentially infinite set of inputs, a circuit does so only for inputs of the same length. Computation of a function on an infinitely large set of inputs is modeled by a family of circuits; one per input length. Care has to be taken when defining how these circuits are generated and which constraints they have to fulfill as these decisions have effects on the resulting complexity hierarchy.

The problem of determining an output functionally related to an initially presented input, however falls short of capturing reactive problems where inputs are presented along an execution. The design of controllers falls into this category: it round-wise receives inputs and responds with outputs such that the potentially infinite execution is within a predefined set of correct executions.

Machine models considered for this class of problems are typically state machines with finite or infinite number of states. Notable special cases of finite state machines are *Moore machines*, where outputs only depend on the current state, and *Mealy machines*, where outputs are determined by the current state and current input. Both play a key role for physical circuit implementations discussed in the following.

Before we start with definitions and problems, the author would like to remark that we work with a broad definition of computation in this text. In particular, we will not distinguish between one-shot input-output problems and providing a continuously executing service, as the line of separation seems to be set rather for historically reasons than by other means. While one may object that with such a definition the specificity of computer science as opposed to computer engineering, control theory, to physics or biology disappears, the author of this text sees this as an opportunity rather than a deficit.

1.1 Basic Notation

We write $\mathbb{N} = \{0, 1, \dots\}$, $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$, and $\mathbb{R}_0^+ = \{x \in \mathbb{R} \mid x \geq 0\}$. Further, for $k \in \mathbb{N}^+$, $[k] = \{1, \dots, k\}$. For the set of Boolean values we write $\mathbb{B} = \{0, 1\}$.

For a directed graph $G = (V, E)$ with vertices V and edges E, we write $\mathrm{In}_v \subseteq V$, with $v \in V$, to denote the set of in-neighbors of v in G, and $\mathrm{Out}_v \subseteq V$ for v's set of out-neighbors. Instead of

vertices, we will sometimes speak of nodes when it is more natural, e.g., when vertices correspond to computing nodes. Likewise, we will sometimes refer to edges as links.

1.2 State Machines and Circuits

We will first make the two machine models, namely, state machines and circuits, more precise.

State machine. Consider a *deterministic, finite Moore state machine*, that operates in discrete rounds. Such a machine M is defined by a tuple $(I, O, S, s_0, \delta, o)$, where I is a finite input alphabet, O a finite output alphabet, S a finite set of states, $s_0 \in S$ the initial state, $\delta : S \times I \to S$ the transition function, and $o : S \to O$ the output function. An *execution of M* for a countably infinite input trace $\iota \in I^\omega$ is the sequence $(\iota_k, s_k, o_k)_{k \geq 0}$, where s_0 is the initial state, $o_0 = o(s_0)$, and for $k \geq 1$,

$$\begin{aligned} s_k &= \delta(s_{k-1}, \iota_{k-1}) \\ o_k &= o(s_k) \ . \end{aligned}$$

Execution (prefixes) of M for input traces $\iota \in I^*$ of finite length $k \geq 0$ are defined as the (unique) k-prefix of an execution of M for an input trace ι' that is obtained by extending ι arbitrarily to an infinite input trace.

With the above definitions, the notion of round-wise computation lies at hand. Given an execution, round k comprises of

(i) reading input ι_{k-1},

(ii) computing the current round's state s_k based on input ι_{k-1} and the previous round's state s_{k-1}, and

(iii) computing the current round's output o_k based on the current state s_k.

Circuit. Likewise, a *combinational feed-forward circuit*, in the following simply referred to as a *combinational circuit*, is a directed node-labeled graph $G = (V, E, g)$ with a finite set of nodes V partitioned into input nodes I, output nodes O, and internal nodes L, a set of signals E, i.e., edges between nodes, and a gate labeling function g that assigns to each internal or output node v, a Boolean formula with atoms in v's incoming edges. The graph is required to be acyclic, input nodes only have outgoing but no incoming edges, output nodes only incoming and no outgoing edges.

Example 1. *Figure 1.1 shows a circuit with input nodes $I = \{1, 2\}$, internal nodes $L = \{3, 4, 5\}$, and output nodes $O = \{6, 7\}$.*

Before defining an execution of a circuit, we need to define delay models for circuit components. We continue with some definitions towards this goal.

An *event* is a tuple $(t, x) \in \mathcal{E} = (\mathbb{R}_0^+ \cup \{-\infty\}) \times \mathbb{B}$. We call t the event's time and x its value. If $t = -\infty$, the event is called an *initial event*. A *signal* is a finite or countably infinite sequence of events, such that, there is exactly one initial event, event times are increasing, and within any finite interval of time only finitely many events occur. Intuitively, the list of events describes the signal's value changes over time, with an event (t, v) denoting that the signal takes value v at time t. The *history of signal s at time t*, denoted by $s[\to t]$, is s with all events with times larger than t removed. We denote the set of histories of signal s at time t by $\mathrm{Hist}_s(\to t)$. A *signal's value at a time $t \in \mathbb{R}$* is the value of the latest event in $s[\to t]$. Note that for convenience of later definitions we also allow a signal to change to, say 1, when it already has value 1. We will call such an event *redundant* as the signal's value stays the same upon such an event.

Let s_1 to s_k, with $k \geq 1$, be signals. A *delay function d* with inputs s_1 to s_k is a function from

$$\mathbb{R}_0^+ \times \bigcup_{t \geq 0} \left(\prod_{i=1}^{k} \mathrm{Hist}_{s_i}[\to t] \right) \to \mathbb{R} \ .$$

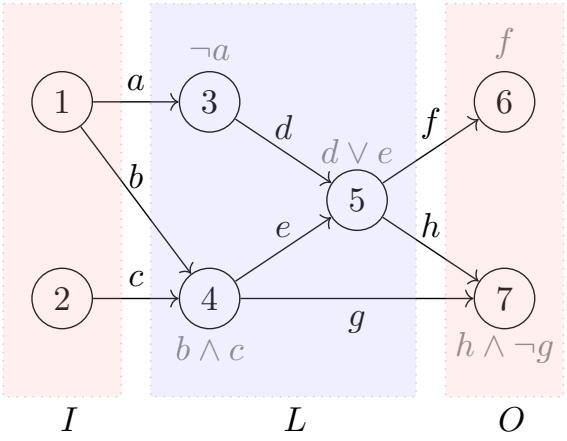


Figure 1.1: Example circuit with two inputs, two outputs, and internal nodes. Internal and output node labels are shown in gray.

Intuitively,

$$\delta(t; s) = d(t, s_1[\to t], \ldots, s_k[\to t]) \tag{1.1}$$

is the delay at time $t \geq 0$ of a channel with inputs s_1 to s_k. By slight abuse of notation we write $\delta(t)$ for $\delta(t; s)$ if s is clear from the context, and extend δ to $t = -\infty$ by setting $\delta(-\infty; s) = 0$ for all s.

Most widely adopted are *pure delay functions*, where $\delta(t) = c$ is constant in t with $c \in \mathbb{R}_0^+$. *Bounded delay functions* guarantee that $\delta(t) \in [\delta^-, \delta^+]$ for non-negative constants δ^- and δ^+.

A *channel* with inputs s_1 to s_k is a tuple (d, f) where d is a delay function with inputs s_1 to s_k, and f the channel's Boolean function from $\mathbb{B}^k \to \mathbb{B}$. A channel induces an output signal o that is defined by Algorithm 1 on page 4 as follows. Given a channel with input signals s_1 to s_k and time $t \geq 0$, the *induced output signal until time* t is the output o_t of Algorithm 1 with inputs $s_1, \ldots, s_k$ and t. The time t in the notation o_t refers to the input event time t. It does not imply stability of the output signal until time t: in particular the induced output until time t does not necessarily have the same events with times less than time t as the induced output until time $t' \geq t$.

We define the *induced output signal* o as the limit, with $\tau \to \infty$, of the common prefixes of signals o_t, where $t \geq \tau$. Note that a priori the existence of the limit is not guaranteed. Further, if the limit exists, it may not be a signal, as its event times may not approach ∞ while it contains an infinite number of events, or it may contain events at finite negative times. We will, however, see that for a large class of naturally occurring channels, the limit is a valid signal and an arbitrarily long prefix of it can be computed efficiently.

The algorithm. Algorithm 1 carries out the following steps:

- It generates an event sequence o, by looping over all input events, in increasing order of event times, until time t is reached. For each such input event, it adds an output event to o whose value is determined by the Boolean function f and whose time is the time τ of the input event plus the channel's delay $\delta(\tau)$.

- In case, o's events are increasing in time, o will be the returned signal (except for the removal of redundant events).

- However, event list o may contain events that are reversed in time, i.e., be of the form $o = x \cdot (t, v) \cdot y \cdot (t', v') \cdot z$ with $t' \leq t$, and sequences x, y, and z. Function doCancellations removes any such violating (sub)sequences, while looping over o's events. An example of two events in wrong order, with the result of an event being removed is shown in Figure 1.3.

- Finally, removeRedundant removes redundant events. Note that according to our definition of a signal this is not necessary for correctness.

Algorithm 1 Algorithm computing the induced output signal until time $t \geq 0$.

Input: signals $s_1, \ldots, s_k$ and time t
Output: signal o

$\tau \leftarrow -\infty$ ▷ initial event time
$o \leftarrow [\,]$ ▷ initial event list
do
  $\delta' \leftarrow \delta(\tau; s)$ ▷ get delay
  $v \leftarrow f\big(s_1(\tau), \ldots, s_k(\tau)\big)$ ▷ new output value
  o.append$\big((\tau + \delta', v)\big)$ ▷ add event
  $\tau \leftarrow$ smallest time greater than τ of an event in $s_1, \ldots, s_k$; $+\infty$ if does not exist
while $\tau \leq t$
$o \leftarrow$ doCancellations(o) ▷ cancel events that violate order
$o \leftarrow$ removeRedundant(o) ▷ generate signal without redundant events

function doCancellations(s) ▷ defined for signals s with finite number of events
  Pending $\leftarrow [\,]$
  for (τ, v) in s **do**
    remove all events with times $\tau' \geq \tau$ from Pending ▷ we say (τ, v) cancels these events
    Pending.append$\big((\tau, v)\big)$. ▷ add event
  end for
  return Pending
end function

function removeRedundant(s) ▷ defined for signals s with finite number of events
  if no events in s **then**
    return s
  end if
  Pending $\leftarrow [\,]$
  Pending.append$\big($first event in $s)$. ▷ add event
  $v' \leftarrow$ value of first event in s
  for (τ, v) in s **do**
    if $v \neq v'$ **then** ▷ ensure that event is not redundant
      Pending.append$\big((\tau, v)\big)$. ▷ add event
      $v' \leftarrow v$
    end if
  end for
  return Pending
end function

- The final finite event list o, by construction, is a signal if the delay function guarantees that o does not contain negative finite event times.

Cancellations. Let e be an event that causes the removal of an event e', i.e., e comes after e' in the event sequence o, but e has time less or equal to the time of event e'. Such an event e' will be removed within doCancellations in Algorithm 1. We then say that e *cancels* e'.

Example 2. *Figure 1.2 shows an example of a channel with inputs s_1 and s_2, and the induced output signal o. The delay $\delta(\tau, s)$ at time τ, is depicted, with signal histories* $\mathrm{Hist}_{s_1}(\to \tau)$ *and* $\mathrm{Hist}_{s_1}(\to \tau)$ *shown in red.*

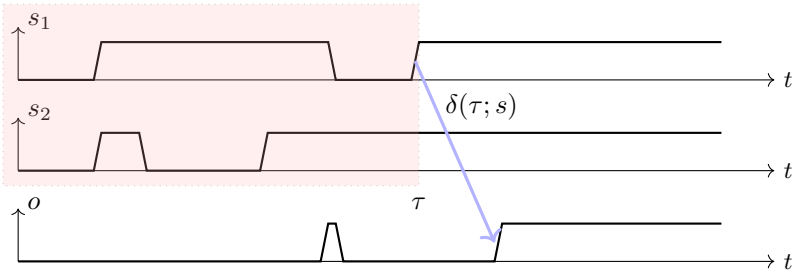


Figure 1.2: Example of a channel (d, f) with input signals s_1 and s_2 and $f \equiv$ And. The channel induces the output signal o.

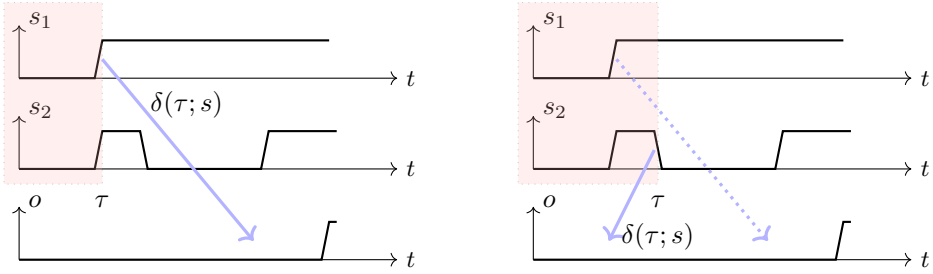


Figure 1.3: Same channel as in Figure 1.2. Suppression of an output pulse. (left) Delay of the first transition. (right) The first transition does not become effective at the output o since it is canceled by the second transition which happens to be before the first one. The output thus remains at value 0.

Example 3 (Channel with single input)**.** *Let $s = \big[(-\infty, 0), (1, 1), (2, 0)\big]$ and (f, d) be the channel with input signal s, Boolean function $f \equiv$ Id, and delay function $d(\tau; s) = 1$. The scenario is depicted in Figure 1.4.*

To determine the induced output signal o until time 2, we execute Algorithm 1 with inputs s and $t = 2$.

1. *Initially, τ is set to $-\infty$ and $o = \big[\,\big]$.*
2. *Next, the initial event $(-\infty + \delta(-\infty; s), f(0)) = (-\infty, 0)$ is added to o, resulting in $o = \big[(-\infty, 0)\big]$. Time τ is updated to the next input event time 1.*
3. *Since the condition of the **while**-loop is fulfilled, δ' is set to $\delta(1; s) = 1$ and v to $f(s(1)) = 1$.*
4. *This results in $o = \big[(-\infty, 0), (2, 1)\big]$ and $\tau = 2$.*
5. *Again the loop's condition is fulfilled, resulting in $o = \big[(-\infty, 0), (2, 1), (3, 0)\big]$ and $\tau = \infty$.*
6. *Function* doCancellations *returns o, since there are no events violating order.*

7. *Function* removeRedundant *returns o, since there are no redundant events.*

Since, execution of the algorithm for all $t \geq 2$ leads to the same output, the output signal induced by the channel is $o = [(-\infty, 0), (2, 1), (3, 0)]$.

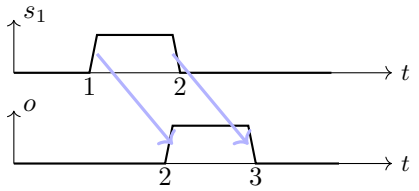


Figure 1.4: Channel from Example 3.

Example 4 (Channel with two inputs)**.** *Let s_1 and s_2 be signals as shown in Figure 1.5 (left) and (right). Further, let (f, d) be a channel with input signals s_1 and s_2, Boolean function $f \equiv$ Or, and a delay function $d(\tau; s)$.*

First, consider the case shown in Figure 1.5 (left). Here a reordering of events with identical value occurs. A reason for this, e.g., is that the presence of an active second input leads to quicker charging of the Or *gate's output, and thus shorter rising transition times.*

1. *Initially the output has value* 0.
2. *The rising transition of s_2 at time* 1 *results in an event $(1 + \delta', 1) = (4, 1)$ being added to o.*
3. *The following rising transition of s_1 at time* 2 *results in an event $(2 + \delta', 1) = (3, 1)$ being added to o.*
4. *Thus $o = [(-\infty, 0), (4, 1), (3, 1)]$.*
5. *Function* doCancellations *returns $o = [(-\infty, 0), (3, 1)]$, canceling the event at time* 4.
6. *Function* removeRedundant *returns o, since there are no redundant events.*

Second, consider the case shown in Figure 1.5 (right) where a reordering of events with distinct values occurs. A reason for such a scenario may be a short input glitch that is suppressed at the output.

1. *Initially the output has value* 0.
2. *The rising transition of s_2 at time* 1 *results in an event $(1 + \delta', 1) = (4, 1)$ being added to o.*
3. *The following rising transition of s_2 at time* 2 *results in an event $(2 + \delta', 0) = (3, 0)$ being added to o.*
4. *Thus $o = [(-\infty, 0), (4, 1), (3, 0)]$.*
5. *Function* doCancellations *returns $o = [(-\infty, 0), (3, 0)]$, canceling the event at time* 4.
6. *Function* removeRedundant *returns $o = [(-\infty, 0)]$, removing the second redundant transition.*

Delay model and execution of a circuit. Returning to circuits, we define: A *delay model* for a circuit is a collection $(d_v)_{v \in V}$ of delay functions. An *execution* of a circuit is a collection of signals, one per node $v \in V$, that fulfills the following properties.

- The signals assigned to input nodes are arbitrary.
- Let $u \in L \cup O$ be an internal or output node. Further let $v_1, \ldots, v_k$ be the incoming edges of u. Let d be the circuit's delay model. Then the signal assigned to u is the output signal induced by the channel with input signals s_1 to s_k assigned to v_1 to v_k, and delay function d_u.

As such, the definition is not constructive and thus does not allow to generate output signals from input signals. We will, however, later on discuss constructive solutions for several practically relevant delay models.

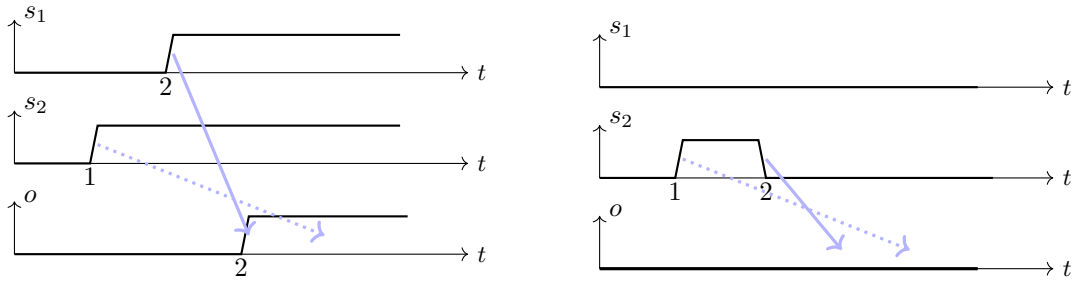


Figure 1.5: Channel from Example 4. Two types of order violations are depicted in the two scenarios. (left) An order violation between events with the same value; here 1. The event with the dotted arrow is canceled. (right) An order violation between events with opposing values. Again, the event with the dotted arrow is canceled. The second event is later removed by the algorithm, since it is redundant.

1.3 State Machine Implementations

We next discuss a well-known and practically relevant relation between state machines and circuits: state machines are implementable by circuits.

Implementation of synchronous Moore state machines. A *register*, or flip-flop, is a state-holding circuit element with a data input D, a clock input clk, and a data output Q. We only provide a high-level specification of a register here, as this will suffice for our purposes. Inputs and outputs are single line, and, by slight abuse of notation, we write D for the input (port) D and its signal in an execution; and similarly for the other ports. Given that the clock respects a certain minimum time between transitions, and that D does not change value within the setup time before a rising (active) clock transition at time t and the hold time after t, output Q is set to $D(t)$ at latest after some clock-to-Q delay $\delta_{\text{reg}} > 0$.

Example 5. *Figure 1.6 shows the behavior of a register sampling* 1 *and then* 0*.*

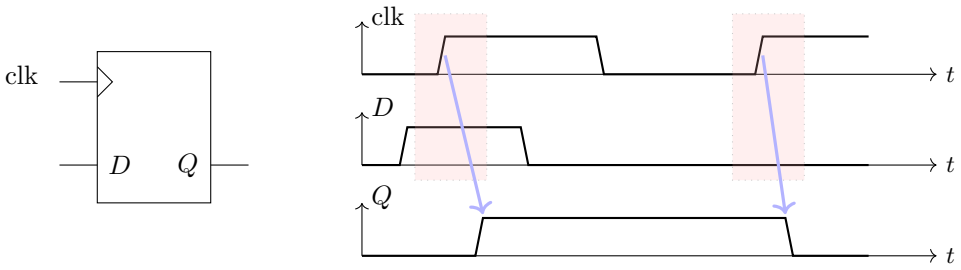


Figure 1.6: Signals of a register initialized to 0, first sampling 1 and then 0. Red areas indicate setup-hold windows where D is required to be stable.

We are now in the position to combine registers and (combinational) circuits and describe the classic, clocked Moore state machine implementation shown in Figure 1.7. Input, local, and output registers have bit width large enough to hold values in the input alphabet I, state space S, and output alphabet O, respectively. With each active clock transition, input and local registers sample new values, and the (combinational) circuitry computes the next state and output. Under practically relevant delay models, the next local state and output signals will stabilize within bounded time, and are then (after an additional setup time) ready to be sampled again at the arrival of the next active clock transition.

Figure 1.8 shows the phases within a clock cycle. The red, unstable phases are conservative over-approximations of actual unstable periods, since stability depends on the combinational circuitry and register states among others. The maximum register clock-to-Q delay δ_{reg} and the maximum

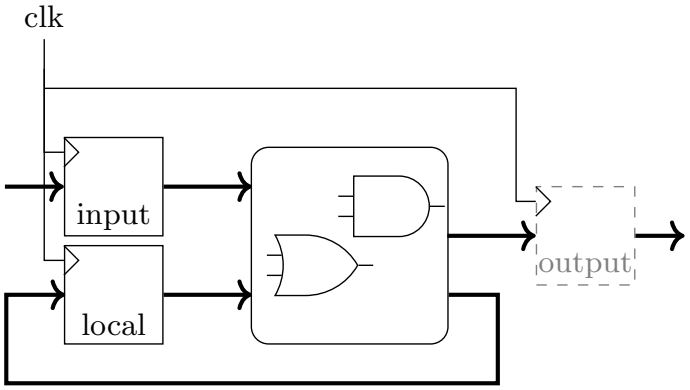


Figure 1.7: Clocked Moore state machine implementation. The dashed output register is not considered to be part of the implementation, and represents the input register of a potentially successor state machine or output logic.

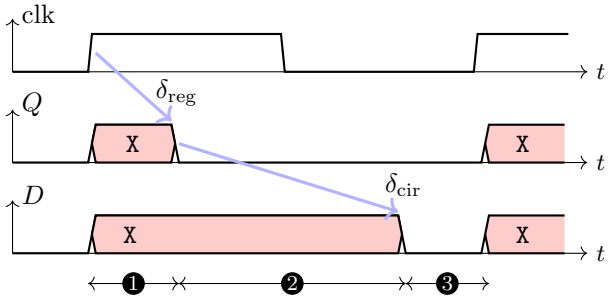


Figure 1.8: Phases of a clock cycle. Red (X) indicates unstable signals. (1) register output stabilization, (2) propagation of outputs through combinational logic to register inputs, and (3) stable register inputs.

combinational circuit's propagation time δ_{cir} determine the duration of these phases. The minimum propagation time through the circuit, here for simplicity, is lower bounded by the trivial bound 0, requiring that the hold time is 0. Generally, designs require minimum propagation times of at least the registers' setup time.

One easily verifies that any finite Moore state machine is implementable by a circuit in Figure 1.7, with round-wise presented inputs and round-wise generated outputs. While this approach is used extensively, it has its limitations discussed in the following section.

1.4 Abstractions Made so Far

Tempting for its simplicity, the classical synchronous state machine implementation has been widely adopted. However, limitations become apparent in certain applications. Three major foci, based on implicitly made assumptions, are:

Delay bounds and steady state behavior. Only maximum and minimum circuit timing has been taken into consideration to validate clock cycle and setup/hold constraints. This allows to consider only the steady state of the combinational circuit and ignore transients.

Precise latency of gates, however, may be essential for several reasons. (i) Working with the steady state may be too conservative to obtain the intended performance (input-output latency and throughput). (ii) The steady state behavior abstracts away from internal glitches[1]. Since glitches contribute to the gates' dynamic power components, their presence or absence influences the circuit's overall power consumption.

[1] A glitch is a short pulse that typically occurs as unintended transient behavior.

For applications where this matters, a timing analysis with refined delay models beyond just the maximum latency may become relevant. This results in potentially less conservative bounds on clock cycle lengths and power consumption. Chapter 2 discusses this path in greater detail.

In some cases, a refined analysis and design of the above scheme is still insufficient, or prohibitively costly, to meet the design constraints. Historically, alternatives have been studied within the asynchronous circuit's community with an initial emphasis on designs that operate in presence of arbitrary delays [1, 2]. A clear, formal model for such circuits and an analysis of their expressiveness was given by Manohar and Moses [3]. For an overview on asynchronous circuits we refer the reader to [4, 5]. While a general discussion of such solutions would go beyond the scope of this work, we deviate from the classical state machine implementation at several places throughout this work.

Boolean signals. So far, circuits have been defined to compute with Boolean values $\mathbb{B} = \{0, 1\}$. In our model, a signal value s is a non-continuous function from time t to its value $s(t) \in \mathbb{B}$ at time t. Clearly, though, this is a convenient, but simplifying, abstraction of physical signal values. Classically, digital gates in electronic circuits are viewed as transforming input voltages into output voltages. In design styles like the widely-used Complementary Metal Oxide Semiconductor (CMOS) technique, signal values are continuous functions from the time to the voltage between a certain probing point and ground.

While, signals are meant to reside within clearly separated regions of low and high voltage, encoding 0 and 1, respectively, transient phases with in-between voltages are inevitable when switching between low and high. These, typically, short transient switches are normally neglected in an analysis. The simplification, however, becomes problematic if switching times become a dominant part. This may be due to (i) aggressive timing of the circuitry with only short stable phases, (ii) faults such as charge induced by ionizing particles [6], or failing transistors or interconnect [7, 8], and (iii) metastability[2] of state-holding gates [9, 10].

While some of these effects can be covered by more advanced delay models as discussed in Chapter 2 or by non-deterministic over-approximation of switching times, we will discuss a different approach in Chapter 3: to compute with non-binary signal values.

Stable architecture. Finally, we assumed that gates and interconnections between gates behave according to their Boolean gate and channel specifications. This neglects changes, or deviation, of intended component behavior and interconnect. While, arguably, from a modeling perspective there is no difference between modeling correct or faulty behavior, solutions that have to take into account faulty components will be different from the simple state-machine implementation in Section 1.3. Clearly, though, problems are different, too.

Depending on the problem to solve, available components, failure models, and the behavior of the environment, available solutions differ considerably. However, key to many solutions is redundant execution of computations over space and/or time. One of the fundamental problems when replicating components is to maintain the component states synchronized to a certain extent. Examples of such states are component outputs, internal component states, and a common notion of time among components.

We discuss such homogenization techniques in Chapters 4 and 5.

1.5 Algorithms as Circuits

Before discussing means of computing in absence of some of the convenient assumptions of delay bounds and steady state behavior, Boolean signals, and stable architectures, the author of this work would like to emphasize on a general view that is taken throughout the work.

For solutions to problems, algorithms are given in terms of circuits rather than pseudo-code. If pseudo-code is given, it is with the intent to comprise of simple computational and communication statements, that are amenable to implementation within circuits. Clearly though, this criterion is informal, with success or failure becoming visible only when systems are indeed mapped to target

[2] Any state-holding gate with at least two stable internal states was shown to contain a third metastable internal state in which it may reside for an unbounded time [9].

technologies. Further, the criterion depends on the intended target technology and size of the circuit. Taking the average of two numbers may be too expensive as a solution in small digital circuits, but be valid for larger digital circuits, or easy to obtain by mixing two equal volumes with a certain protein concentration, and taking the resulting protein concentration. We will thus comment on the intended target technology when it is not clear from the context.

Algorithms and circuits are thus treated interchangeably, instead of seeing circuits as implementations of algorithms. For previous work on gate-level algorithms, we refer the reader to distributed algorithms that have been shown to solve *fault-tolerant distributed pulse generation*. Consider a distributed system of $n > 1$ nodes, that can communicate with each other via binary signals whose delay is within bounds $[\delta^-, \delta^+]$. Each node has a dedicated output signal; its clock output. Some of the nodes are correct, i.e., they follow the algorithm, and some are Byzantine faulty, i.e., they produce arbitrary binary output signals and communicate arbitrary binary signals to the other nodes. Communicated signals may be even different for different nodes in case the sending node is faulty.

We say a distributed algorithm, i.e., a family of circuits indexed by the set of nodes, solves fault-tolerant distributed pulse generation in presence of up to $f \in \mathbb{N}$ Byzantine nodes, if

- each correct node produces a clock signal with a cycle length, i.e., the time between two successive active transitions, within some strictly positive bounds $[T^-, T^+]$, and
- the clock skew, i.e., the maximum of the difference of time of the k^{th} active clock transition of two nodes, over all $k \geq 1$ and all pairs of correct nodes, is within some strictly positive bounds $[\sigma^-, \sigma^+]$,

given that at most f nodes are Byzantine in an execution.

The DARTS algorithm [11, 12] is an example of an algorithm that solves the problem if $f < n/3$ and if certain constraints on node internal and external communication delays are fulfilled. The FATAL algorithm [13] is another example of a fault-tolerant distributed pulse generation algorithm targeted at simple hardware components. In addition to being fault-tolerant, it is self-stabilizing: irrespective of the past until a time t, given that there are no more than $f < n/3$ Byzantine faults[3] during some $[t, t + T]$, then the system solves fault-tolerant distributed pulse generation within $[t + T_s, T]$ with a certain probability $p(T_s)$ that converges to 1 as $T_s < T$ approach infinity.

A discussion on gate-level models for such algorithms is given in [14]. Distributed gate-level algorithms in more general settings are discussed in Dolev *et al.* [15].

1.6 Organization of this Work

The following three chapters are devoted to computing in machine models with relaxed assumptions. Chapter 2 discusses work on delay models. Chapter 3 is on computing with non-binary, but still digital, signals. Chapter 4 discusses algorithms that provide a homogeneous state despite highly dynamic architectures. Finally, Chapter 5 provides an outlook to currently ongoing research and future directions.

At the beginning of a chapter we provide a blue box that lists, and briefly sketches, the main research papers this chapter is based on. Central results of a chapter are also highlighted in green. At the end of a chapter, own work that is related to the chapter is provided in a green box.

[3] Definitions of correct and faulty have to be adapted to correct and faulty within a certain time interval. The same holds for the definition of an algorithm solving fault-tolerant distributed pulse generation.

Chapter 2

Timing

This chapter discusses results from the following research articles.

- [16] Függer, Nowak, Schmid. *Unfaithful glitch propagation in existing binary circuit models.* IEEE Transactions on Computers 65.3 (2015): 964-978, and

  [17] Függer, Nowak, Schmid. *Unfaithful glitch propagation in existing binary circuit models.* In IEEE 19th International Symposium on Asynchronous Circuits and Systems (ASYNC), pages 191–199, 2013.

  We show deficiencies in previous, widely used delay models for digital circuits. In particular the propagation of glitches through circuits is not captured correctly: problems solvable in Netwonian physics are not solvable in such models and vice versa.

- [18] Függer, Najvirt, Nowak, Schmid. *A Faithful Binary Circuit Model.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2019), and

  [19] Függer, Najvirt, Nowak, Schmid. *Towards binary circuit models that faithfully capture physical solvability.* In Design, Automation & Test in Europe (DATE), pages 1455–1460, 2015.

  Based on our previous finding of problems in delay models, we propose a new class of delay models: involution delay models. We show that variants of the short pulse filtration problem, related to glitch propagation, are solvable in these new models if and only if they are solvable in Newtonian physics.

- [20] Najvirt, Függer, Nowak, Schmid, Hofbauer, Schweiger. *Experimental validation of a faithful binary circuit model.* Proceedings of the 25th edition on Great Lakes Symposium on VLSI (GLSVLSI). 2015.

  We challenge the involution delay model by experiments on inverter chains in an ASIC circuit. Paremetrizations of the model are obtained by these experiments.

- [21] Függer, Maier, Najvirt, Nowak, Schmid. *A faithful binary circuit model with adversarial noise.* Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2018.

  The involution delay model is a deterministic delay model: a function from the history of signals to a delay. In this work we weaken this assumption, adding bounded non-determinism to the delay and show that the model is equivalent to the deterministic model with respect to solvability of the short pulse filtration problem variants.

2.1 Glitch Propagation

Consider a chain of combinational gates, say buffers, and a glitch propagating through the chain. An example scenario is depicted in Figure 2.1. Following the convention of delay models and circuit

executions from Chapter 1, transitions that occur in reverse order (here, falling before rising in a low signal), cancel each other and the signal value remains unchanged. In the example, the pulse shrinks in length when propagated from a to b and cancellation occurs when propagating the pulse from b to c.

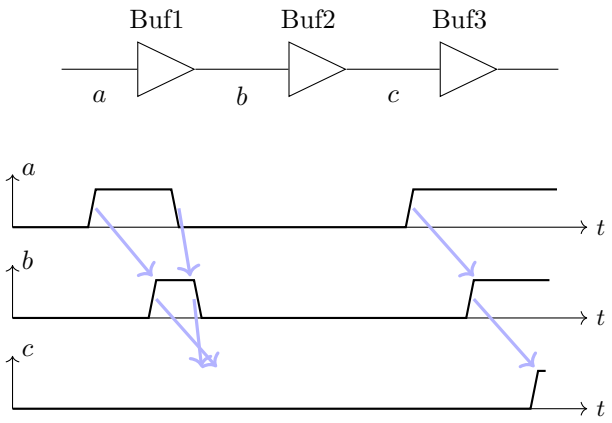


Figure 2.1: A glitch propagating through a buffer chain. The glitch shrinks in length and is suppressed after the second buffer. Blue arrows indicate combined gate and interconnect delays.

Such behavior is typically observed if gates or interconnect form capacitors of non-negligible capacitance that needs to be charged or discharged when the gate drives a different output value. Charging or discharging is not only responsible for the delay until sufficient voltage is available across the capacitor, but this delay also depends on charge already present in the capacitor.

Figure 2.2 on page 13 shows a Spice simulation of a chain of 7 buffers, g10 to g16, with an input pulse applied at the first buffer input. The chain was synthesized for the 15 nm FinFET NanGate open cell library [22], layed out and routed, and its parasitics extracted for simulation (all within the Cadence suite).

While the pulse in Figure 2.2a propagates through all buffer stages, Figure 2.2b shows a pulse that shrinks in length until it is fully canceled by the pre-last stage.

Pulses, and their propagation through a circuit will play an important role when discussing channel models. Formally, in our binary circuit model, a *pulse* of length $\Delta > 0$ at time T is a signal p with

$$p(t) = \begin{cases} 1 & \text{if } T \leq t < T + \Delta \\ 0 & \text{otherwise} \end{cases} . \tag{2.1}$$

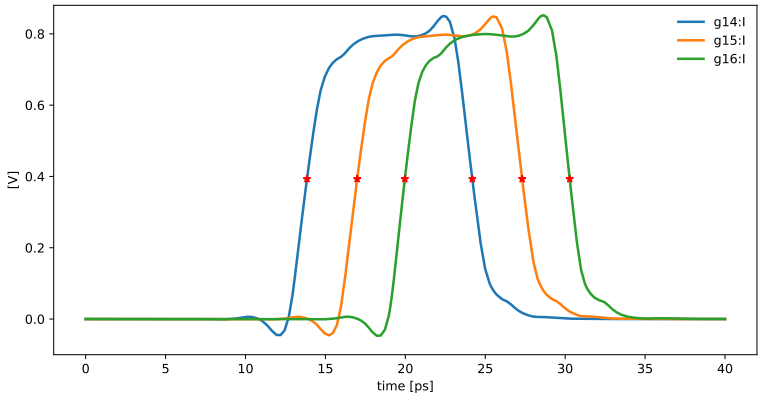
A signal *contains a pulse* of length $\Delta > 0$ at time T if its event list contains the two consecutive events $(T, 1)$ and $(T + \Delta, 0)$.

2.2 Bounded Single-history Delay Models

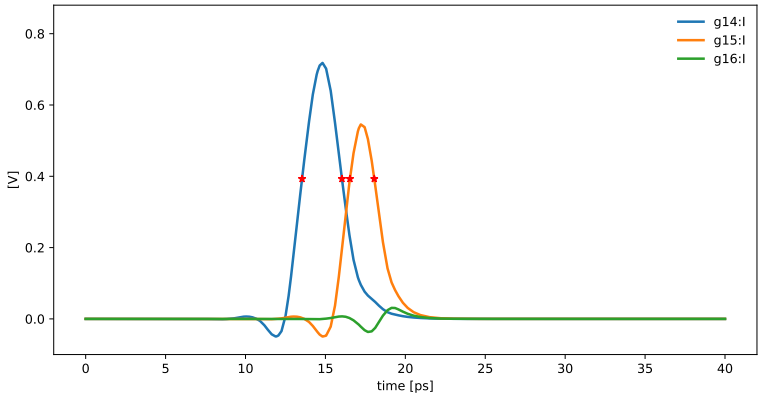
To facilitate a comparison between delay models we follow an approach proposed in Függer *et al.* [16, 17], unifying several delay models. For that purpose we define bounded single-history channels. While "single-history" refers to the fact that its delay depends on the current and previous transition only, the term "bounded" will be discussed in Section 2.8.1, when we introduce unbounded single-history channels.

Definition 6 ([16])**.** *A* bounded single-history channel *is a tuple* (F, f, δ)*, where* $F \in \mathbb{B}$ *specifies whether the channel is non-forgetful* ($F = 0$) *or forgetful* ($F = 1$)*,* f *is the channel's Boolean function, and* $\delta : \mathbb{R} \to \mathbb{R}$ *determines the channel's delay function* d *as specified in the following. The channel's function* δ *must fulfill:*

- δ *is nondecreasing.*



(a) The pulse propagates through the buffer chain.



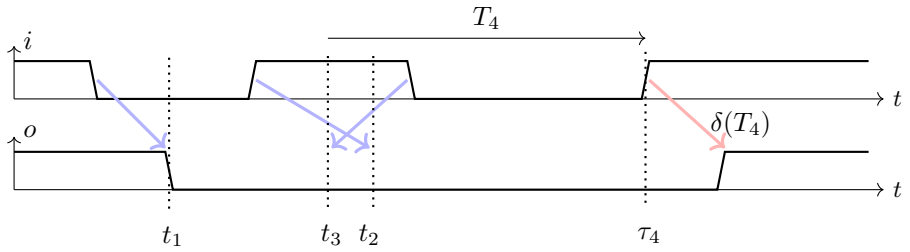
(b) The pulse at the input of buffer g14 is attenuated by buffer g14 and filtered out by g15.

Figure 2.2: Two scenarios of a buffer chain with a short input pulse. The first 4 buffers, g10 to g13, are used to shape the input signal. Input signals of the buffers g14, g15, g16 are shown in the figure. Threshold crossings are shown as red markings. Layout and routing was done in Cadence Encounter, Spice simulations in Cadence Spectre. $V_{\mathrm{DD}} = 0.8\,\mathrm{V}$.

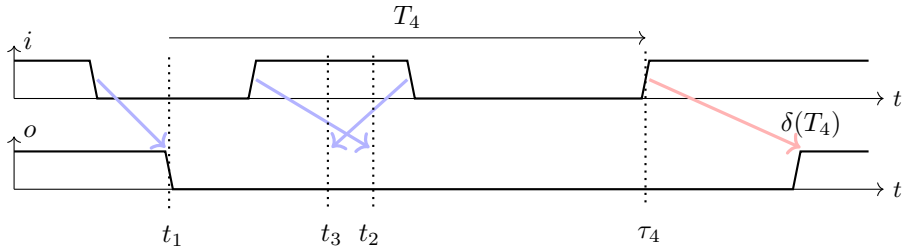
- $\delta(\infty) = \lim_{T\to\infty} \delta(T)$ *is finite and positive.*

The two variants of bounded single-history delay channels, non-forgetful and forgetful, differ in whether the time of a canceled output event has an effect on the delay for the next output event (non-forgetful) or not (forgetful). The value $d(t, s[\to t])$ *of their delay function is given by the following inductive definition. For* $n \geq 1$*,*

- *Let* τ_i*, with* $i \in [n]$ *and* $-\infty < \tau_1 < \tau_2 < \cdots < \tau_n \leq t$*, be the times where* $f(s(\cdot))$ *changes value.*
- *Let* $t_i = \tau_i + d(\tau_i, s[\to \tau_i])$ *for* $i \in [n-1]$*. Further, set* $t_0 = -\infty$*.*
- *Let* $T_n = \tau_n - t'$*, where* t' *is determined as follows:*
  - *If the channel is non-forgetful, then* $t' = t_{n-1}$*.*
  - *If the channel is forgetful, then* $t' = t_{n'}$*, with* $n' \leq n-1$ *maximal, given that the output event at time* $t_{n'}$ *was not canceled by output events at times* $t_1, \ldots, t_{n-1}$*.*
- *Set* $d(t, s[\to t]) = d(\tau_n, s[\to \tau_n]) = \delta(T_n)$*.*



(a) Non-forgetful bounded single-history channel.



(b) Forgetful bounded single-history channel.

Figure 2.3: Behavior of a non-forgetful and a forgetful bounded single-history channel with $f \equiv \mathrm{id}$ and identical δ. The output events at times t_2 and t_3 are canceled. This leads to different T_4 for the two channel variants.

By slight abuse of notation we will refer to δ as the channel's delay function.

The difference between the two bounded single-history channel types is demonstrated in Figure 2.3. Figure 2.3a shows the input and output signals of a non-forgetful channel and Figure 2.3b of a forgetful channel in presence of the same input signal. Otherwise, both channels have identical Boolean functions $f \equiv \mathrm{id}$ and delay functions δ.

The delay of a bounded single-history channel, depends only on the time since the last (non-forgetful channel) or last non-canceled (forgetful channel) output event; thus the name single-history. The channel is further called bounded, because of the following property:

Lemma 7. *The delay of a bounded single-history channel with delay function δ, is bounded. In particular it is within $[\delta(-\delta(\infty)), \delta(\infty)]$.*

Proof. From the definition of T, it is easily verified that the parameter T for a bounded single-history channel is always larger than $-\delta(\infty)$. In fact such a T occurs as the infimum of parameter T for the delay of the second transition in pulses of duration $\varepsilon > 0$, with the first transition being delayed by $\delta(\infty) < \infty$. The lemma's statement then follows from the fact that δ is nondecreasing on $\mathbb{R}$. □

Recall from Section 1.2 that the induced output signal until a time t does not necessarily exist. In particular the list of output events generated by Algorithm 1 may not necessarily have common prefixes for increasing times t. This may be the case if output event $k \geq 1$ can cancel output event $1 \leq \ell < k$ for arbitrarily large $k - \ell \geq 1$. One can, however, show that bounded single-history channels are well-behaved, and their induced output signal indeed exists.

Indeed, we will see in Lemma 17 on page 21 that bounded single-history channels, as their name suggests, have bounded delays within $[-\delta_{\inf}, \delta_\infty]$. Combining this property with the fact that for a bounded single-history channel, an event can at most cancel the previous event, but never more, we obtained:

Lemma 8 (Lemma 4 and 5 in [16])**.** *Denote by S_n the output event list generated by Algorithm 1 for a prefix of the input event list of length n. Further, denote by $S_n \mid t$ its restriction to events at times at most t.*

Then, for the non-forgetful and the forgetful single-history channel, for all $t \geq 0$, there exists an $N \in \mathbb{N}$ such that $S_n \mid t$ is constant for all $n \geq N$.

2.3 Conventional Delay Models

Two widely adapted delay models are the pure delay model, also referred to as transport or constant delay model, and the inertial delay model. Both are implemented in standard circuit design frameworks. Example 9 discusses their use in the VHDL language.

Example 9. *In the following a VHDL listing for a simple buffer gate with delay is given. Depending on the buffer's architecture, it uses a pure delay channel with parameter* 5 *ps (BUF_transport), or an inertial delay channel with parameters* 4 *ps and* 5 *ps (BUF_inertial) or* 5 *ps and* 5 *ps (BUF_after).*

```
library ieee;
use ieee.std_logic_1164.all;

entity BUF is
  Port ( X: in  std_logic;
         Y: out std_logic );
end BUF;

architecture BUF_transport of BUF is
begin
  Y <= transport X after 5 ps;
end BUF_transport;

architecture BUF_inertial of BUF is
begin
  Y <= reject 4 ps inertial X after 5 ps;
end BUF_inertial;

architecture BUF_after of BUF is
begin
  Y <= X after 5 ps;
end BUF_after;
```

In terms of our model, a pure delay channel is a single-history channel with delay function

$$\delta(T) = \delta_\infty \ , \tag{2.2}$$

where δ_∞ is its delay.[1] The behavior of pure delay channels is identical, no matter whether they are forgetful or non-forgetful: they simply never cancel transitions. Figure 2.4 shows two pure delay functions with different δ_∞.

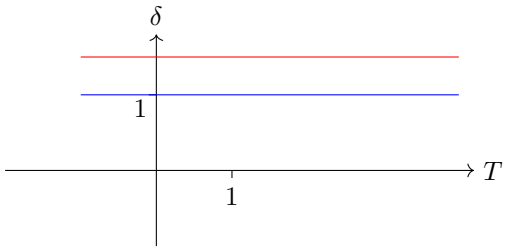


Figure 2.4: Pure delay function with parameters $\delta_\infty = 1$ (blue) and $\delta_\infty = 1.5$ (red).

The advantage of the slightly more refined inertial delay channels is not immediate for standard synchronous designs. While correctness of classical synchronous state machine implementations indeed only rely on minimum and maximum delay, i.e., ranges for δ_∞, inertial delays play a

[1] We use the notation δ_∞ since $\delta(\infty) = \delta_\infty$. While this may seem heavy in notation for a constant delay channel, we will follow this scheme also for more refined channels. We call δ_∞ the long-term delay since it is the delay for a channel that has not seen an input for increasingly long durations.

role for power estimates and when leaving the single-clock synchronous design path. Indeed, Unger [23] proposed a design technique for asynchronous sequential switching circuits that relies on the availability of inertial delay channels.

Cast in our model, the delay function of an inertial delay channel is,

$$\delta(T) = \begin{cases} \delta_\infty & \text{if } T \geq T_0 \\ -T_0 & \text{otherwise} \end{cases} . \tag{2.3}$$

with parameters T_0 and long-term delay δ_∞. Figure 2.5 shows two delay functions for different long-term delays δ_∞. Observe, the discontinuity at $T = T_0$ that will play a central in its analysis later on.

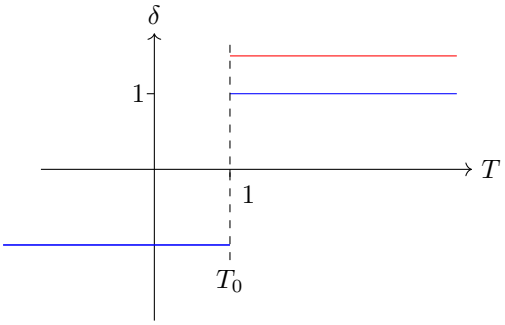


Figure 2.5: Inertial delay function with parameters $T_0 = 1$ and $\delta_\infty = 1$ (blue) and $\delta_\infty = 1.5$ (red).

Bellido-Diaz *et al.* [24] proposed a more refined model, the Delay Degradation Model (DDM). The model overcomes shortcomings of the inertial delay model in that it captures gradual degradation of pulse widths by channel, rather than sudden cancellation at a certain threshold. Its delay function is,

$$\delta(T) = \delta_\infty \left(1 - e^{-\frac{T - T_0}{\tau}}\right) . \tag{2.4}$$

Figure 2.6 shows two delay functions that differ in their long-term delay δ_∞. By contrast to the inertial delay model, the delay function is continuous over $\mathbb{R}$. The function was derived by analytic arguments of charging effects and its accuracy compared against Spice simulations. Indeed DDM achieves very good accuracy in challenging scenarios with significant degradation effects.

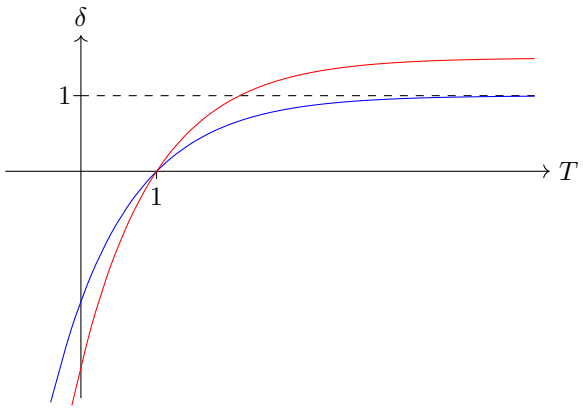


Figure 2.6: Two DDM delay functions with parameters $\tau = 1$, $T_0 = 1$, and $\delta_\infty = 1$ (blue) and $\delta_\infty = 1.5$ (red).

2.4 Glitch Propagation in Models

Starting from the three models with their goals to refine on glitch propagation in circuits, the question arises, which of these models indeed captures these phenomena. While we would intuitively rank the three models from pure delay, followed by inertial delay, and DDM for their capability to model real glitching behavior, a formal answer to this question has several advantages:

1. Accuracy of simulations. We clearly aim for more accurate simulation results with more refined models. Accuracy is measured as deviation from physical measurements, or detailed simulation models that are considered sufficiently close to real measurements. In our setting these are typically (post-layout) Spice models with extracted parasitics.

2. Impact of correctness proofs on real circuits. A correctness proof of a circuit within a model that deviates significantly from reality clearly does not imply correctness of the circuit in real-world settings. While all proposed delay models clearly deviate from reality in one or the other way, a more coarse question of whether a model deviates from reality would be to ask if an (important) problem P is solvable in this model if and only if it is solvable in reality. We termed such models *faithful with respect to* P [16].

3. Validity of circuit synthesis. The above observation also has implications on automated circuit synthesis. By analogous arguments as for correct proofs, a circuit synthesized within a model that is not faithful with respect to P may lead to circuits that fail in practice for problems that are related to P.

2.5 Short Pulse Filtration in Physical Models

Before discussing adequate problems P to prove faithfulness or non-faithfulness with respect to P, let us specify a model that we consider to capture reality. In doing so we follow Marino's work on metastability [9], who defined a *realistic circuit* as one whose voltage signals can be described in a time and value continuous system of differential equations with restrictions that, e.g., imply causality.[2] For a detailed description of the model we refer the reader to [9].

In our case, (normalized) voltage signals are assumed to be within $[0, 1]$. To map these signals to binary values, we assume the existence of thresholds $l_0, l_1 \in (0, 1)$ with $l_0 < l_1$ and will speak of the signal being logical $b \in \mathbb{B}$ if the signal's value is within L_b, where $L_0 = [0, l_0]$ and $L_1 = [l_1, 1]$.

It remains to chose a problem P for which to assess whether the model is faithful with respect to P. In our case, the informal goal is to correctly model propagation of glitches, i.e., short pulses, through circuitry. Note that this question is closely related to correctly modeling metastability — although, the latter is not straightforward to state in a binary-valued model. We will, see, however, that both — glitches and metastability — are indeed closely related.

Several problems lie at hand when it comes to finding typical problems where correct handling of glitches supposedly plays a role in whether a circuit solves this problem, or not. Although care must be taken on the precise specification of such problems, a fundamental component that would help solving such problems, is a single input single output circuit that filters away too short pulses at the output. In its simplest form we require that this device only works for a single positive input pulse. We termed this problem the Short Pulse Filtration (SPF) problem [16, 17]. It remains to specify what filtering away means in this context — too strict definitions inevitable will lead to unsolvability in real systems. The SPF problem is specified as follows and has two variants:

Definition 10 (Unbounded SPF and bounded SPF in the physical model [16])**.** *Consider a circuit with a single input and a single output in the physical model. It* solves unbounded Short-Pulse Filtration (SPF) *if:*

(i) If the input signal is constantly logical 0*, then so is the output signal.*

[2]While one may argue that such a continuous model neglects quantum effects with discrete energy potentials, note that these potentials are steady-state solutions. A more severe limitation, is the lack of probabilistic effects in our model. We will discuss probabilistic, or in this case, nondeterministic, extensions in Section 2.10, however, a relation to quantum effects is open at this point.

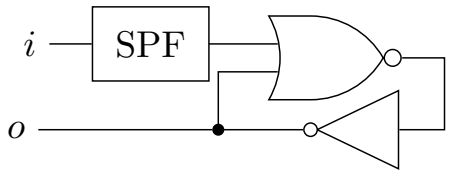


Figure 2.7: Reducing a single-input bistable element stabilizing within bounded time to a bounded SPF circuit.

(ii) There exists an input signal such that the output signal attains logical 1 *at some point in time.*

(iii) There exists some fixed $\varepsilon > 0$ *such that, if the output signal is not interpreted as logical* 1 *at two points in time* t *and* t' *with* $t' - t < \varepsilon$*, then it is not logical* 1 *at any time in between* t *and* t'*.*

It solves bounded SPF *if additionally:*

(iv) There exists a time $T > 0$ *such that, if the input signal switches to logical* 1 *by time* t*, then the output signal value is either logical* 0 *or logical* 1 *at time* $t + T$ *and remains logical* 0 *respectively logical* 1 *thereafter.*

Conditions (i) and (ii) outrule trivial solutions to the problem that simply set their output to constant 0 or 1. Condition (iii) prevents a solution from producing an output pulse of length less than a predefined ε; thus the problem's name. Condition (iv) further requires the circuit output to stabilize within bounded time.

We will next discuss both SPF variants in Marino's model of physical circuits.

2.5.1 Impossibility of Bounded SPF

In [16] we showed by reducing the problem of building a bistable storage element that stabilize in bounded time to building a circuit that solves bounded SPF:

Theorem 11 ([16])**.** *No physical circuit solves bounded SPF.*

For the proof, let us define a *single-input bistable element* in the physical circuit model as a circuit with a single input and a single output that fulfills properties (i) and (ii) of SPF, and additionally:

(iii') If the output is logical 1 at some time t, it also remains logical 1 at all times larger than t.

Analogous to the bounded SPF problem, a circuit solves the *single-input bistable element stabilizing within bounded time*, if additionally (iv) of SPF holds.

Indeed, Marino's Theorem 3 in [9] then directly implies:

Corollary 12 ([16])**.** *In the physical circuit model, there is no single-input bistable element stabilizing within bounded time.*

To show Theorem 11, it remains to reduce a single-input bistable element stabilizing within bounded time to a bounded SPF circuit. Figure 2.7 shows such a reduction by a circuit that uses a bounded SPF circuit to build a respective single-input bistable element. Its idea is to use a storage loop, formed by the NOR with the INV feedback, to capture a 0-1-0 pulse, and permanently output 1. The storage loop does not fail, due to too short input pulses, because of property (iii) of the SPF. Finally, boundedness carries over from the bounded SPF to the single-input bistable element.

2.5.2 Possibility of Unbounded SPF

For the weaker unbounded SPF problem, we showed:

Theorem 13 ([16])**.** *There is a physical circuit that solves unbounded SPF.*

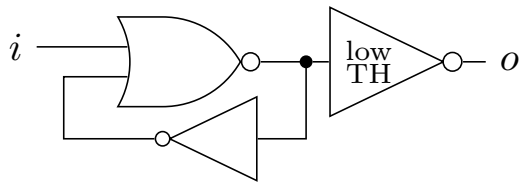


Figure 2.8: A circuit that solves unbounded SPF. The inverter driving the output is a low threshold inverter.

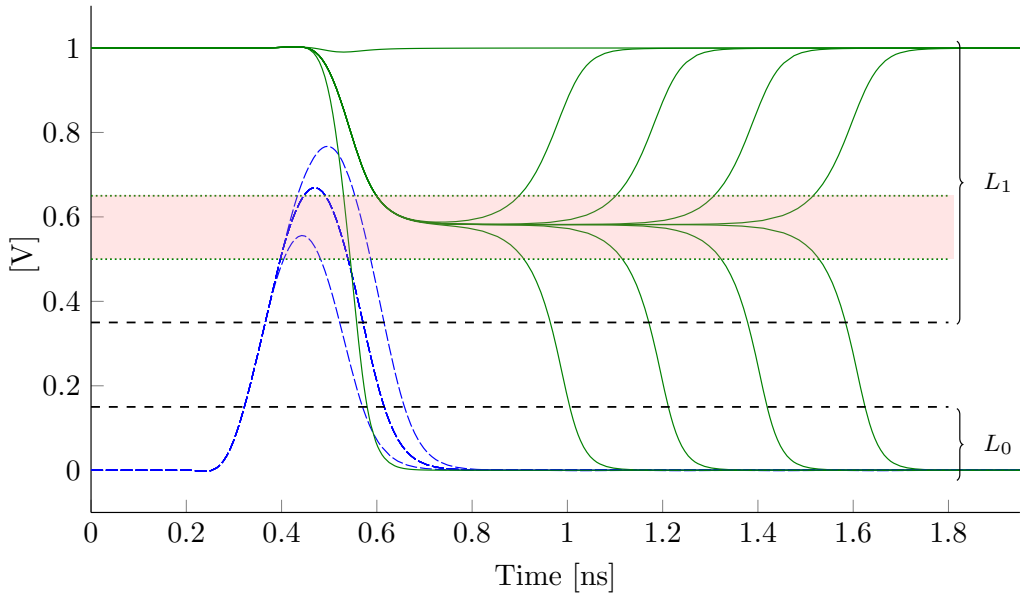


Figure 2.9: Spice simulations of a storage loop in presence of short input pulses (blue). The storage loop gets metastable, resulting in its internal signal staying arbitrarily long near the metastability point (bundled green), before it resolves to stable 0 or 1. Once the signal leaves the metastable region (red area), it resolves quickly, however. Regions L_0 and L_1 for voltages that are interpreted as logical 0 and logical 1 by the successive low threshold inverter are marked in the figure.

Figure 2.8 depicts the circuit that solve unbounded SPF. It comprises of a storage loop formed by the NOR and Inv gates, and a successory low threshold Inv gate. Such an inverter has a shifted input-output function that maps a larger (upper) voltage range of input values to logical 1 and a smaller (lower) voltage range to logical 1.

Figure 2.9 provides an intuition why the circuit indeed solves the unbounded SPF problem: The (blue) input signal may lead to a metastable upset of the storage loop, resulting in the (green) signal at the low threshold inverter input. However, the low-threshold inverter interprets a sufficiently large voltage range, including the metastable region of the storage loop, as a logical 1 and thus maps it to 1. Metastable upsets thus lead to arbitrarily late, but not spurious, transitions at the inverter output. One easily verifies that the circuit thus fulfills all properties required for solving unbounded SPF.

2.6 Short Pulse Filtration in Binary-valued Models

We are ready to state unbounded SPF and the bounded SPF from Definition 10 in binary-valued circuit models. Further, we introduce a third problem, the eventual SPF problem.

Definition 14 (Unbounded SPF, bounded SPF, and eventual SPF in binary-valued models [16])**.** *Consider a circuit with input i and output o in the binary-valued model. The circuit* solves unbounded Short-Pulse Filtration (SPF) *if:*

(w) For every pulse p, there exists an execution that has p as the input signal. (Well-formedness)

(i) In all executions, if the input signal is constant zero, then so is the output signal. (No generation)

(ii) There exist a pulse p such that, in all executions with p as the input signal, the output signal is not the constant zero signal. (Nontriviality)

(iii) There exists an $\varepsilon > 0$ such that, in all executions, the output signal does not contain a pulse of length less than ε. (No short pulses)

The circuit solves bounded SPF *if additionally:*

(iv) There exists a $K > 0$ such that, in all executions with a pulse as the input signal whose last event is at time T, the output signal does not change anymore after time $T + K$. (Bounded stabilization time)

The circuit solves eventual SPF *if conditions (w), (i)–(iii), and the following condition hold:*

(iv') There exists an $\varepsilon > 0$ and a $K > 0$ such that, in all executions with a pulse at time T as the input signal, the output signal does not contain a pulse of length less than ε after time $T + K$. (Eventually no short pulses)

One easily verifies that the problems are ordered as eventual SPF $\leq$ unbounded SPF $\leq$ bounded SPF with respect to their difficulty: bounded SPF implies unbounded SPF, which itself implies eventual SPF.

We say a circuit model is *faithful with respect to glitch propagation* if within the model bounded SPF is not solvable, and bounded SPF is solvable.

2.7 Unfaithful Delay Models

While the non-faithfulness of the discontinuous inertial delay channels may not come as a surprise, a much larger class of channels has difficulties capturing glitch propagation effects adequately:

Theorem 15 ([16])**.** *No circuit model with bounded single-history channels is faithful with respect to glitch propagation.*

The proof is separated into three cases of bounded single-history channels, each of which is sketched in the following sections.

2.7.1 Constant Delay Channels

Constant delay channels are widely used in circuit design. The following theorem shows that they are not faithful, however.

Theorem 16 ([16])**.** *No circuit solves unbounded SPF with only constant delay bounded single history channels.*

The proof is based on the observation that a circuit that comprises only of constant delays "samples" the input signals at certain, discrete, times, only. Assume a circuit that solves SPF and let i be its input and o its output. Then for every $t \geq 0$, $o(t)$ depends only on a finite set of input values $i(t-\tau_1)$, $i(t-\tau_2)$, $\ldots$, $i(t-\tau_k)$, with $\tau_1, \ldots, \tau_k > 0$. Changes of the input signal that happen between these times thus get unnoticed by the current output value. However, at certain times t, the set of input values grows. The proof now relies on choosing a time t where the set is guaranteed to not grow within an environment of length $\varepsilon > 0$, and changing the input signal to produce an arbitrarily short pulse at the output — making use of the previously established indistinguishably argument for inputs.

2.7.2 Non-Constant Forgetful Channels

For a bounded single-history delay function δ, let

$$\delta_\infty = \lim_{t\to\infty} \delta(t) \tag{2.5}$$

$$\delta_{\text{inf}} = \lim_{t\to 0^+} \delta(-\delta_\infty + t) \ . \tag{2.6}$$

The notation δ_{inf}, for infimum delay, as well as the notation of a bounded single-history channel is due to the following result:

Lemma 17 ([16]). *All events produced by a bounded single-history channel are delayed by times within* $[\delta_{inf}, \delta_\infty]$.

For a bounded single-history channel c, that is either forgetful or non-forgetful, let

$$\gamma(c) = \inf\{\Delta > 0 \mid \delta - \delta_\infty + \delta(\Delta - \delta_\infty)\} \ . \tag{2.7}$$

We then showed that:

Lemma 18 ([16]). *Let c be a non-forgetful or a forgetful bounded single-history channel with initial value 0. The following statements are equivalent: (i) $\gamma(c) > 0$, (ii) c is not a constant-delay channel, (iii) There exists an input pulse, such that the output of c is the zero signal.*

One can then show for forgetful bounded single-history channels:

Lemma 19 ([16]). *Let c be a non-constant-delay forgetful bounded single-history channel with initial value 0. Let s be a signal that does not contain pulses of length greater or equal to $\gamma(c)$ and that is not eventually equal to 1. Then the output of c with input s is the zero signal.*

The above lemma can finally be applied to show that the circuit shown in Figure 2.8 on page 19, except with a standard inverter with the non-constant forgetful bounded single-history channels instead of the low-threshold inverter, can be used to solve bounded SPF. We thus obtain:

Theorem 20 ([16]). *Let c^* be a non-constant-delay forgetful bounded single-history channel. Then there exists a circuit solving bounded SPF whose channels are either constant-delay channels or c^*.*

2.7.3 Non-Constant Non-forgetful Channels

The proof for the non-constant non-forgetful case is by dissecting into three subcases for the delay function δ. The cases differ in how the function behaves at $-\delta_{\text{inf}}$.

- There exists a $t > -\delta_{\text{inf}}$ such that $\delta(t) < \delta_\infty$. From the fact that $\delta(t)$ goes to δ_∞ with $t \to \infty$, we have that δ is non-constant within the subdomain $(-\delta_{\text{inf}}, \infty)$.
- Otherwise, for all $t > -\delta_{\text{inf}}$, it is $\delta(t) = \delta_\infty$. Either,
  - δ is continuous at $-\delta_{\text{inf}}$, i.e., at $-\delta_{\text{inf}}$ its left limit equals its right limit, or
  - δ is non-continuous at $-\delta_{\text{inf}}$.

For all three cases, one can construct (three different) circuits that solve SPF. The idea underlying all circuits is to produce pulses that flank the actual input pulse, and that are shifted by a uniformly lower bounded $\varepsilon > 0$ in time by the presence of an input pulse. This shift is then transformed into a steady state change via a storage loop. We refer the reader to [16] and the extended arxiv version for a description of the circuits and their correctness proofs. We thus obtain:

Theorem 21 ([16]). *Let c^* be a non-constant-delay non-forgetful bounded single history channel with initial value 0. Then there exists a circuit solving SPF whose channels are all either constant-delay channels or c^*.*

2.8 A Faithful Delay Model

The following section is based on [18, 19], where we show that faithful delay models exist.

Central to the new model is a modification of bounded single-history channels from Section 2.2 to so called unbounded single-history channels. We will give a definition of these channels in Section 2.8.1. In search for faithful subclasses of unbound single-history channels, we start with a simple physical model of the effects responsible for pulse shortenings in Section 2.8.2 and work our way towards delay functions from there.

2.8.1 Unbounded Single-history Channel

Definition 6 on page 12 formalized bounded single-history channels, with δ's domain being $\mathbb{R}$. Like bounded single-history channels, unbounded single-history channels are defined via their delay function δ (respectively two delay functions $\delta_\downarrow$ and $\delta_\uparrow$) and an initial value. The parameter T of the delay function is defined analogously to the parameter T for bounded single-history channels. We consider only non-forgetful channels, here, however.

The major difference of unbounded to bounded channels is that the delay function's domain is not $\mathbb{R}$, but a subset. Along the work in [19], we define:

Definition 22. *An* unbounded single-history channel *is defined analogously to a bounded single-history channel, except that its two delay functions (up and down) are strictly increasing concave functions of the form*

$$\delta_\uparrow : (-\delta_\infty^\downarrow, \infty) \to (-\infty, \delta_\infty^\uparrow) \text{ and}$$
$$\delta_\downarrow : (-\delta_\infty^\uparrow, \infty) \to (-\infty, \delta_\infty^\downarrow)$$

with finite $\delta_\infty^\uparrow = \lim_{T\to\infty} \delta_\uparrow(T)$ *and* $\delta_\infty^\downarrow = \lim_{T\to\infty} \delta_\downarrow(T)$.

2.8.2 Analog Channel Model

Consider a channel with a successive identity gate — the generalization to other Boolean gates is straightforward by modifying the Boolean behavior. The channel itself is driven by a predecessor gate. Assume that $V_{\text{gnd}} = 0\,V$ and $V_{\text{dd}} = 1\,V$. Figure 2.10 on page 23 depicts the schematic of a simple equivalent circuit.

The predecessor gate output is modeled as an amplifier with infinite amplification and a prespecified discretization voltage V_{th}. Its output is a single bit digital signal u_i. The amplifier itself is not part of this channel, however, and is only shown to depict the input environment. The binary signal u_i is then delayed by constant time T_p, obtaining the binary signal u_d. With the occurrence of a rising transition of u_d, the following analog switching element follows its rising switching waveform $f_\uparrow$ from its current (voltage) state onwards, until a falling transition occurs, and it reacts by following the falling switching waveform $f_\downarrow$ from its current voltage onwards, etc. The resulting signal is an analog signal u_r. The signal is finally discretized again by an amplifier, resulting in the channel–gate output signal u_o. This signal potentially is the input of a downstream gate, or a circuit output. Figure 2.10 (below subfigure) also shows the delays $\delta_\uparrow(T_1)$ and $\delta_\downarrow(T_2)$ that result for the first and second transition of the input signal u_i to the output signal u_o.

In [18] we have shown how the switching waveforms can be translated into delay functions that result in the same digital behavior. We start from the ansatz,

$$\begin{aligned} \delta_\uparrow(T) &= -f_\uparrow^{-1}(f_\downarrow(T)) \\ \delta_\downarrow(T) &= -f_\downarrow^{-1}(f_\uparrow(T)) \ , \end{aligned} \tag{2.8}$$

where $f_\uparrow$ is continuous and strictly increasing and $f_\downarrow$ is continuous and strictly decreasing.

In the above formula, e.g., at a rising transition, $\delta_\uparrow(T)$ returns the time by which $f_\uparrow$ has to be shifted so that the analog output signal u_r remains continuous with respect to the output signal caused by the previous falling transition. We further need that rising and falling switching waveforms start at analog voltages 0 and 1 respectively, i.e.,

$$f_\uparrow(0) = 0 \quad \text{and} \quad f_\downarrow(0) = 1 \ .$$

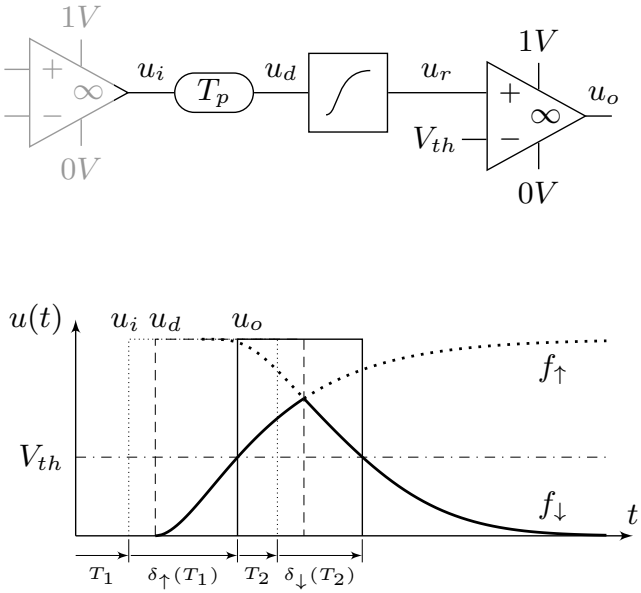


Figure 2.10: Physical, analog channel model based on continuous rising $f_\uparrow$ and falling $f_\downarrow$ switching waveforms. From [18].

Further, they are required to be full-swing, i.e.,

$$\lim_{t\to\infty} f_\uparrow(t) = 1 \quad \text{and} \quad \lim_{t\to\infty} f_\downarrow(t) = 0 \ .$$

Incorporating both into our ansatz, yields a revised ansatz:

$$\begin{aligned} \delta_\uparrow(T) &= -f_\uparrow^{-1}(f_\downarrow(T + \delta_\infty^\downarrow)) + \delta_\infty^\uparrow \\ \delta_\downarrow(T) &= -f_\downarrow^{-1}(f_\uparrow(T + \delta_\infty^\uparrow)) + \delta_\infty^\downarrow \ , \end{aligned} \tag{2.9}$$

where

$$\begin{aligned} \delta_\infty^\uparrow &= \lim_{T\to\infty} \delta_\uparrow(T) \\ \delta_\infty^\downarrow &= \lim_{T\to\infty} \delta_\downarrow(T) \ . \end{aligned}$$

From Figure 2.10 one readily observes for the above two quantities,

$$\begin{aligned} \delta_\infty^\uparrow &= T_p + f_\uparrow^{-1}(V_{th}) \text{ and} \\ \delta_\infty^\downarrow &= T_p + f_\downarrow^{-1}(V_{th}) \end{aligned}$$

First order low pass filter. As an example, consider $f_\uparrow$ and $f_\downarrow$ to result from a first-order RC low pass filter behavior. Then,

$$f_\downarrow(t) = 1 - f_\uparrow(t) = e^{-t/\tau} \ ,$$

where τ is the RC time constant. Using (2.9), we obtain the delay function of a so-called *exp-channel* as being

$$\begin{aligned} \delta_\uparrow(T) &= \tau \ln(1 - e^{-(T+T_p-\tau\ln(V_{th}))/\tau}) + T_p - \tau\ln(1 - V_{th}) \\ \delta_\downarrow(T) &= \tau \ln(1 - e^{-(T+T_p-\tau\ln(1-V_{th}))/\tau}) + T_p - \tau\ln(V_{th}) \ . \end{aligned} \tag{2.10}$$

Figure 2.11 on page 24 shows examples for delay functions $\delta_\uparrow$ for an exp-channel with two different parameters.

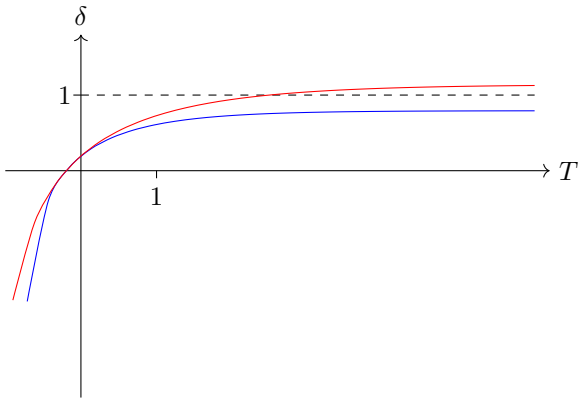


Figure 2.11: Two exp-channel delay functions with parameters $T_p = 0.1$, $V_{\text{th}} = 0.5$, and $\tau = 1.0$ (blue) and $\tau = 1.5$ (red). The resulting long-term delays are $\delta_\infty = 0.1 + 1.0\ln(2)$ (blue) and $\delta_\infty = 0.1 + 1.5\ln(2)$ (red).

2.8.3 Involution Channels

In [18] we introduced a new channel type, the so called involution channels, that are defined as follows:

Definition 23 ([18])**.** *An* involution channel *is an unbounded single-history channel with*

$$-\delta_\uparrow\big(-\delta_\downarrow(T)\big) = T \text{ and } -\delta_\downarrow\big(-\delta_\uparrow(T)\big) = T \tag{2.11}$$

for all applicable T.

The name involution channel is due the fact that, in case $\delta_\uparrow = \delta_\downarrow = \delta$, function $-\delta$ is an involution, i.e., is self-inverse.

All delay functions from involution channels are necessarily continuous. For simplicity, we also assumed them to be differentiable — δ being concave thus implies that its derivative δ' is continuous and monotonically decreasing. While this is a restriction of the general case, the underlying motivation was that delay functions in physical circuits will most likely be continuous in their behavior.

In the following we will discuss our main result from [18] that render involution channels interesting candidates for faithful delay channels:

Theorem 24 ([18])**.** *The binary circuit model with involution channels is faithful with respect to glitch propagation.*

2.8.4 Central Properties of Involution Channels

As a first observation, note that $\delta_\uparrow$ and $\delta_\downarrow$ are not independent of each other. The timing behavior of involution channels is fully determined by either one of the delay functions, as $\delta_\uparrow(T) = -\delta_\downarrow^{-1}(-T)$ (and similarly for $\delta_\downarrow$).

Further, so called *strictly causal channels* will play a role to show uniqueness of executions. As we will see from Lemma 27, strictly causal channels have a minimum positive delay which enables constructive generation of executions; a central property for the existence of algorithms that simulate circuits with such channels.

Definition 25 (Strict causality [18])**.** *An involution channel is* strictly causal *if* $\delta_\uparrow(0) > 0$.

One observes from (2.11) and the functions being strictly increasing that a channel is strictly causal if and only if $\delta_\downarrow(0) > 0$,

For the special case of an exp-channel, strict causality is solely determined by T_p as follows:

Lemma 26 ([18]). *An exp-channel is strictly causal if and only if* $T_p > 0$.

The next lemma identifies an important parameter $\delta_{\min}$ of a strictly causal involution channel — the minimum delay of any transition that will not be canceled, as we will show in Lemma 28. By contrast, transitions that will be canceled very well can (and in fact we show that they will) have delays less than $\delta_{\min}$.

Lemma 27 ([18]). *A strictly causal involution channel has a unique* $\delta_{\min}$ *defined by*

$$\delta_{\min} = \delta_{\uparrow}(-\delta_{\min}) = \delta_{\downarrow}(-\delta_{\min}) \ ,$$

which is positive. For exp-channels, $\delta_{\min} = T_p$. *For the derivative, we have*

$$\begin{aligned} \delta'_{\uparrow}(-\delta_{\downarrow}(T)) = 1/\delta'_{\downarrow}(T) \quad &\Rightarrow \\ \delta'_{\uparrow}(-\delta_{\min}) = 1/\delta'_{\downarrow}(-\delta_{\min}) \ . \end{aligned}$$

From this result, we finally established a central relation of $\delta_{\min}$ to the minimum channel delay of non-canceled transitions:

Lemma 28 ([18]). *Let* t_n *and* t_{n+1} *be the times of the* n*th and* $(n+1)$*th input transitions. The following are equivalent:*

1. *The* n*th and* $(n+1)$*th pending output transitions cancel.*
2. $t_{n+1} \leq t_n + \delta_n - \delta_{\min}$
3. $\delta_{n+1} \leq \delta_{\min}$

2.8.5 Possibility of Unbounded SPF

To show faithfulness of the involution channels with respect to glitch propagation, we need to show that, within this delay model: (i) there exists a circuit that solves unbounded SPF, and (ii) that there is no circuit that solves bounded SPF. We start by showing the former:

Theorem 29 ([18]). *There is a circuit that solves unbounded SPF.*

Its proof is constructive in the sense that it provides the circuit that solves unbounded SPF. It is shown in Figure 2.12. Interestingly, the circuit is in accordance with the circuit used to show that unbounded SPF is solvable in physical circuits.

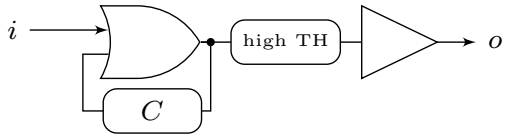


Figure 2.12: A circuit solving unbounded SPF, consisting of an OR gate fed-back by channel C, and an exp-channel HT implementing a high-threshold buffer. From [18].

For the proof, we proceed to show the following theorem, that characterizes the circuit's OR gate output behavior with respect to the input pulse. It shows that positive input pulses that have length less than Δ_0 result in output pulse trains that degrade to eventually being constant 0, longer input pulses lead to pulse trains whose pulses grow to eventually being constant 1, and input pulses of length exactly Δ_0 result in an output that oscillates forever.

Theorem 30 ([18]). *The fed-back* OR *gate with a strictly causal involution channel has the following output when the input pulse has length* Δ_0*:*

- *If* $\Delta_0 > \tilde{\Delta}_0$, *then the output is eventually constant* 1.

- *If* $\Delta_0 < \tilde{\Delta}_0$*, then the output is eventually constant* 0*.*
- *If* $\Delta_0 = \tilde{\Delta}_0$*, then the output after the initial pulse* Δ_0 *is a periodic pulse train with uptime* $\tilde{\Delta}_1$*, period* κ *and duty cycle*[3] $\gamma = \tilde{\Delta}_1/\kappa < 1$*.*

Furthermore, the stabilization time in the first two cases is in the order of $\log_a(1/|\Delta_0 - \tilde{\Delta}_0|)$ *with* $a = 1 + \delta'_\uparrow(0)$*.*

To show Theorem 29, it remains to show that the signal generated by the feedback-OR gate is properly filtered by the high-threshold INV. In particular all eventually vanishing pulse trains must be filtered out and produce a constant 0 signal at the output. Towards this goal, we showed that exp-channels have such filtering properties:

Lemma 31 ([18])**.** *Let* $\Theta > 0$ *and* $\Gamma \in [0, 1)$*. Then, there exists an exp-channel* C *such that every finite or infinite pulse train with pulse lengths* $(\Theta_i)_{i\geq 0}$ *and duty cycles* $(\Gamma_i)_{i\geq 0}$*, that fulfills*

- $\Theta_i \leq \Theta$*, for* $i \geq 0$*, and*
- $\Gamma_i \leq \Gamma$*, for* $i \geq 1$

is mapped to the zero signal by channel C*.*

Theorem 30 and Lemma 31 finally show Theorem 29.

2.8.6 Impossibility of Bounded SPF

The idea to show that no circuit solves bounded SPF has similarities to Marion's work on metastability [9], showing that no bistable circuit is metastability-free. It first defines a metric space in which (i) circuits are shown to be continuous functions from inputs to outputs, and (ii) argues that the problem to be solved requires a non-continuous function. While in Marino's work this problem is the problem of building a bistable it is solving bounded SPF in our case. Metric spaces are fundamentally different though since our signals are binary as opposed to the continuous valued signals in [9].

Continuity of Channels. In [18] we showed that strictly causal unbounded single-history channels are continuous within a certain metric space. Towards this goal, we define $s_1 \leq s_2$ for two signals s_1, s_2, if the relation holds point-wise. Similarly functions like $|s_1 - s_2|$ that denote the inequality signal of s_1, s_2, are defined point-wise.

As a first result, monotonicity of unbounded single-history channels is established via:

Lemma 32 ([18])**.** *Let* s_1 *and* s_2 *be signals such that* $s_1 \leq s_2$ *and let* C *be a channel. Then,* C *is monotone in the sense that* $f_C(s_1) \leq f_C(s_2)$*.*

The proof is by induction on the input transitions of s_2 and uses the fact that delay functions are non-increasing in parameter T. Monotonicity of T for two corresponding transitions in s_1 and s_2 is due to the assumption that $s_1 \leq s_2$. Care has to be taken in that s_2 may contain pulses that do not exist in s_1: if so, we simply create ones (for analysis) in s_1.

One may next define a measure for signals, and from this, a distance between signals:

Definition 33 ([18])**.** *Let time* $T \geq 0$*. For a signal* s *denote by* $\mu_T(s)$ *the* measure *of the set* $\{t \in [0, T] \mid s(t) = 1\}$*. For any two signals* s_1 *and* s_2*, we define their* distance up to time T *by* $d_T(s_1, s_2) = \mu_T(|s_1 - s_2|)$*.*

Given a strictly causal unbounded single-history channel, it remains to show how the distance between output pulses changes if input pulses are transformed. We state one of these bounds below. For a detailed analysis we refer the reader to [18].

[3]The duty cycle of a periodic signal is the ratio of high-time and period. For a non-periodic pulse train, we define the duty cycle of pulse $i \geq 1$ as the ratio of the high-time of the i^{th} cycle and its total length.

Lemma 34 ([18]). *Let $T \geq 0$ be a time, s be a signal that is eventually constant 0 and, C be a strictly causal unbounded single-history channel. Then, adding a 0-1-0 pulse of length $\varepsilon_1 \geq 0$ at the last transition of s (thus extending the signal) or after it (thus resulting in a new pulse), and calling the new signal s', we have*

$$\mu_T(f_C(s')) \leq \mu_T(f_C(s)) + \big(1 + \delta'_{\downarrow}(-\delta_{\min})\big)\varepsilon_1 \ .$$

From the above lemma, showing that small extensions have small effects on the measure of a signal, and similar lemmas that study measure and the distance changes in presence of input signal changes, we finally obtain:

Theorem 35 ([18]). *Let C be a strictly causal unbounded single-history channel and let $T \geq 0$. Then, the mapping $s \mapsto f_C(s)$ is continuous with respect to the distance d_T.*

Impossibility in combinational feed-forward circuits. Combinational feed-forward circuits by definition do not contain storage loops. Any output signal of such a circuit is obtained by a finite application of channel functions and Boolean functions to the input signals. Boolean functions are easily seen to be continuous in the metric space with distance d_T, where $T \geq 0$. Theorem 35 showed, that channel function, are, too. Since the composition of continuous functions is continuous, and by choosing T arbitrarily large, we can show that a combinational feed-forward circuit cannot solve bounded SPF until time T since solving this problem would require a circuit with a discontinuous input-to-output function. It follows that:

Theorem 36 ([18]). *No forward circuit solves bounded SPF.*

Impossibility in general circuits. The restriction to purely combinational circuits is strong. However, we showed that the case of general circuits can be reduced to combinational feed-forward circuits. The principal idea is to simulate a general circuit by a combinational circuit. Central to the simulation argument is that the existence of a combinational circuit that produces the same output events as the original general circuit can be shown. However, the simulation relation *a priori* only holds until a certain event number $N \geq 0$. Making use of the strict causality, and thus a strictly positive $\delta_{\min} > 0$, one can relate N to a time $T \geq 0$ until when the combinational circuit simulates output signals correctly. Choosing T arbitrarily large, we finally obtained:

Theorem 37 ([18]). *No circuit solves bounded SPF.*

2.9 Experimental Evaluation

While faithfulness as previously discussed is a primary requirement when it comes to correctness of circuits, it does not yet separate quantitatively good models from those that do not provide reliable quantitative results. Clearly, though, a tradeoff between (time, space, conceptual) complexity and accuracy has to be taken into account for any model that strives for accuracy. While detailed Spice models are typically considered the golden standard in circuit models, they arguably are too coarse for detailed predictions that include non-electrical properties, e.g., behavior under radiation and mechanical stress, and too slow for larger circuit simulations and formal verification. In [20], we studied how close involution models come to Spice models and physical implementations in terms of predicting transition times in digital circuits. In a recent work in [25] we compared the involution model to other delay models and Spice simulations. The simulations are made with a toolsuite that was designed to integrate the involution model into a standard design flow.

As a test circuit for the comparison in [20] we used a custom UMC-90 ASIC [26] that comprises an inverter line, tapped with low-intrusive high-speed on-chip analog amplifiers to observe the signal traveling along the line. By applying fast, successive input transitions to the line's input, we were then able to trace potential degradation effects of the traversing pulses. Figure 2.13 shows the

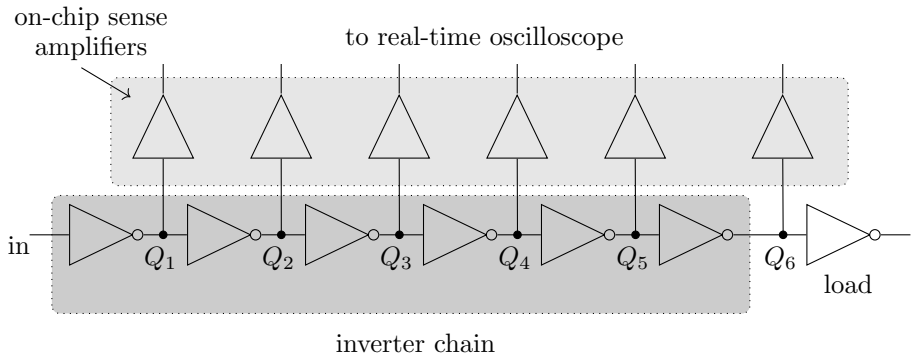


Figure 2.13: Schematics of the ASIC used for experimental setup. It comprises of an inverter chain with analog high-speed sense amplifiers for measurements along the chain. From [20].

experimental setup. The inverters comprise of 700 nm × 80 nm (WxL) pMOS and a 360 nm × 80 nm nMOS transistors with threshold voltages of 0.29 V and 0.26 V. Nominal $V_{\text{dd}} = 1\,\text{V}$.

The probing amplifiers are attached to a high-speed real-time oscilloscope (50 Ω input) to log traces for later offline analysis in custom Matlab scripts. The measured amplifier gain is 0.15, with an overall −3 dB cutoff frequency of approximately 8.5 GHz. Their input load is equivalent to 3 Inv. Independent power supplies and grounds for inverters and amplifiers was used to reduce measurement noise and simplify differential voltage readout. To reduce temperature effects, the PCB with the directly bonded die has been mounted on a Peltier-cooled copper heat sink.

Figure 2.14 shows the readout at the oscilloscope (12 GHz Tektronix TDS 6154B real-time oscilloscope, 4-channel sampling rate of 20 GS/s). The input was generated with a 3.35 GHz Agilent 81134A pulse/pattern generator. The figure also shows corresponding simulated traces from Spice models that have been extracted from the synthesized design and that were calibrated to the analogue behavior of the amplifiers. Observe that the measured and simulated results match well. This became important for high frequency inputs near nominal voltages of $V_{\text{dd}} = 1\,\text{V}$, since the analog amplifiers were too slow to observe these undistorted by measurements. We thus had to resort to simulations for these scenarios. In the scenario in Figure 2.14, $V_{\text{dd}} = 0.6\,\text{V}$ for this reason.

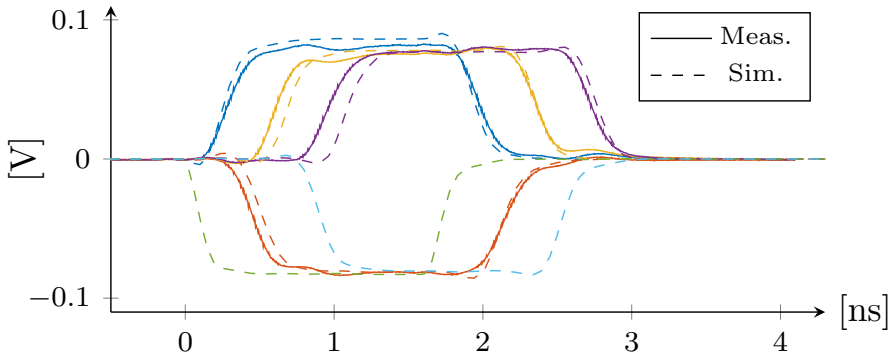


Figure 2.14: Measured (solid) and simulated (dashed) waveforms at taps Q_1, Q_3, Q_5 (bottom, shifted by $-V_{\text{dd}}$) and Q_2, Q_4, Q_6 (top), with $V_{\text{DD}} = 0.6\,\text{V}$. The lower output voltage is due to the amplifier gain of 0.15. From [20].

By applying an input pulse with varying width, and logging the transition times, one may extract $\delta(\infty)$ and $\delta(T)$ for varying $T \in \mathbb{R}$. Depending on whether the pulse is positive or negative, the falling or rising delay function can be extracted. Figure 2.15 shows the resulting delay function. The positive arm has been measured, and the negative arm obtained by mirroring along $y = -x$. Observe the smooth transition at this diagonal as predicted from the involution delay model. We also draw a fitted DDM delay function for comparison.

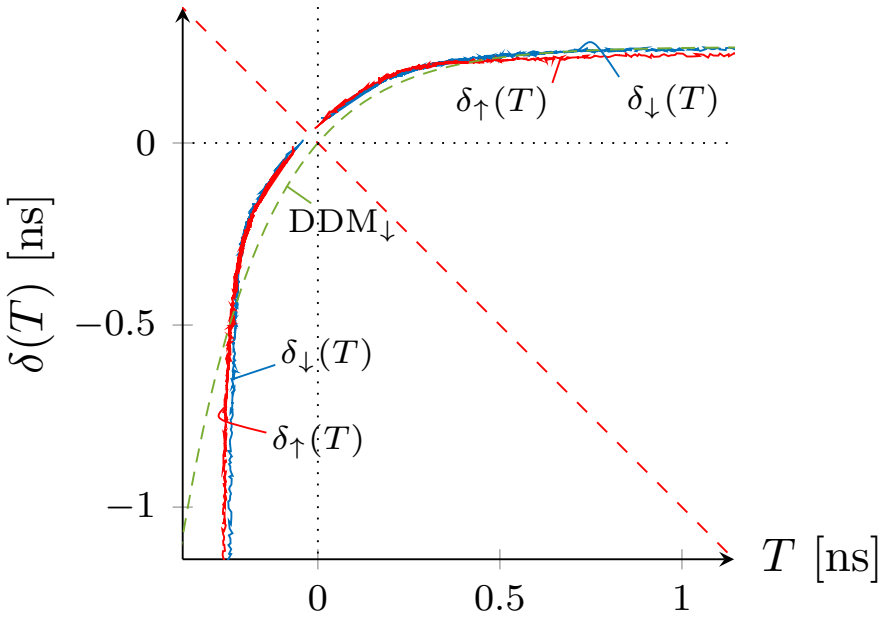


Figure 2.15: Measured $\delta_\downarrow$ (blue) and $\delta_\uparrow$ (red) for UMC-90 inverter chain for $V_{DD} = 0.6\ V$, which support the involution hypothesis. By contrast, there is no perfect fit for the exponential DDM delay function (dashed green). From [20].

Finally, Figure 2.16 shows a longer input trace with high-frequency pulses to ensure that charging effects play a role in the propagation delay. Observe that some of the pulses are already filtered away at Inv4, while others are filtered at Inv6. This effect cannot be obtained with classical pure delay models. While the solid blue lines show the traced voltage signal, blue and red arrows indicate signal transitions according to the involution delay model (blue) and the DDM (red). Both the involution delay model and DDM match the measured signal transitions with great accuracy.

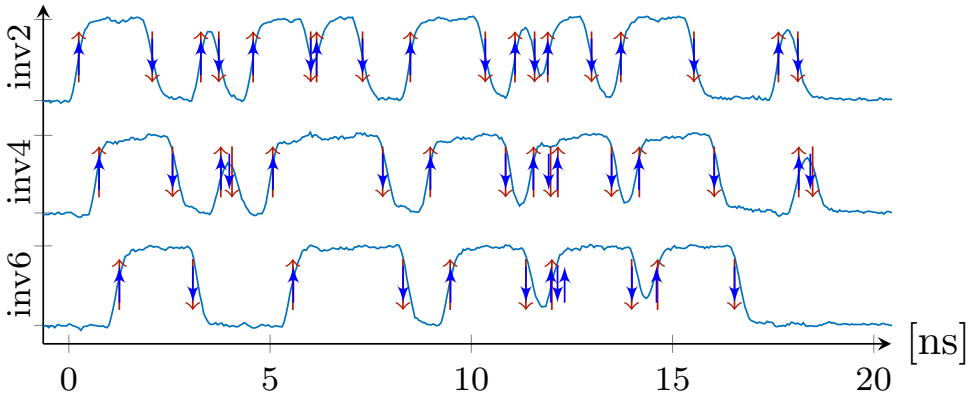


Figure 2.16: Measured waveform (solid) for the UMC-90 inverter chain, with the predictions according to the involution model (red long up/down-arrows) and the DDM (blue short up/down arrows). From [20].

While measurements had to be run at reduced V_{dd} to slow down inverter delays, we used the measurements to also calibrate a Spice model. In the Spice simulations nominal V_{dd} runs could then be generated to validate if the circuit also behaves according ot the involution delay model at nominal operation conditions. Our delay model showed great accuracy at nominal voltage, too.

Finally, the same inverter line was synthesized in the UMC-65 nm process and Spice simulations used to check accuracy of the involution delay model. Our delay model again showed great accuracy. Figure 2.17 shows the inverters' delay functions obtained for different V_{dd}.

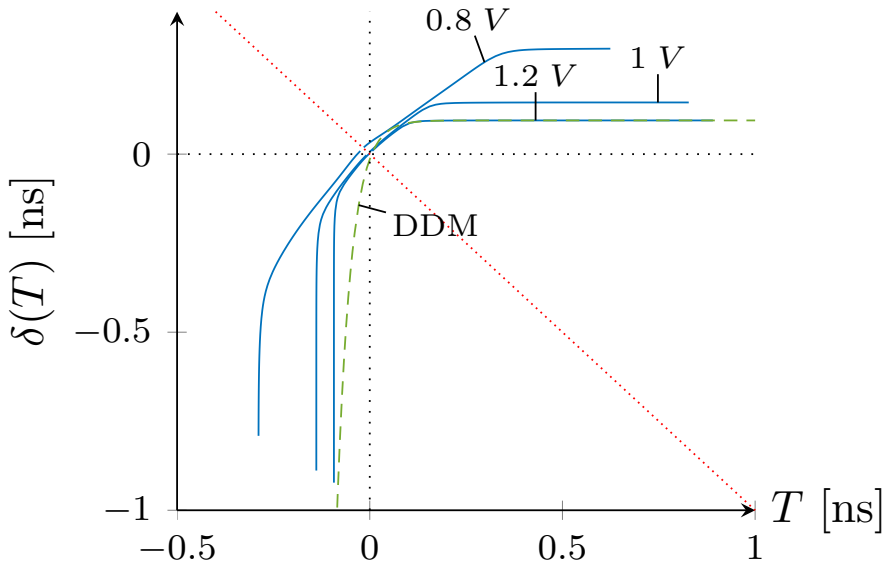


Figure 2.17: Simulated $\delta_\downarrow$ for UMC-65 inverter chain for $V_{\text{dd}} \in \{0.8, 1, 1.2\}$ V. DDM fitting for $V_{\text{dd}} = 1.2\ V$.

2.10 Noise

All delay models presented in this chapter are deterministic: given an input trace, there is a unique output trace that is compliant with this input. While non-determinism can be obtained by varying channel parameters from execution to execution, such models rather account for variations in production than for dynamic noise sources like changes in voltage, temperature, wear-out, etc. In fact we will discuss high-frequency voltage changes, so called voltage droops, in greater detail in Chapter 3. Voltage droops result in rapid changes in gate propagation delays and — as such — are not covered by the current model.

A first path to broaden the model to include varying delay functions, is to introduce additional parameters. For example, a gate with delay function $\delta : \mathbb{R} \to \mathbb{R}$ may be generalized to a family of delay functions $(\delta_v)_{v\in[0,1]}$ with voltage $v \in [0, 1]$ as an additional parameter. Care has to be taken to ensure consistent switching between the delay functions for this approach.

Another approach is to allow a limited additive noise term, resulting in a delay function $\delta + \eta$, for $\eta \in [\eta_{\min}, \eta_{\max}]$. It is this path that was followed in [21]. Rather than assuming a certain probabilistic noise model, we let the adversary arbitrarily chose $\eta \in [\eta_{\min}, \eta_{\max}]$ for each transition. While, we conjectured that the involution delay model does not allow for any additive noise, it turned out that a significant amount of noise could be handled by the model without losing its faithfulness property.

This is in particular surprising as additive noise gives the adversary not only the possibility to shrink and increase pulse sizes, but also to cancel pulses that would otherwise not be canceled and de-cancel pulses that would be canceled.

To show that the involution channel with noise is faithful with respect to glitch propagation, one has to show that (i) bounded SPF cannot be solved in this model, and (ii) unbounded SPF can be solved in the model.

Direction (i) immediately follows from the result for the involution model: Whatever circuit we design, the adversary may choose $\eta = 0$, resulting in executions in the involution delay model. From Theorem 37 on page 27 we, however, have that this set of executions does not fulfill the requirements of bounded SPF.

It remains to show the more involved direction (ii). Without going into details on the proofs we mention that the possibility result of showing that unbounded SPF is solvable within such a channel model relies on the same circuit, shown in Figure 2.12, that also solves unbounded SPF in the involution channel model and in the Newtonian physical model. While not mathematically relevant, this is reassuring as it increases confidence that the possibility results do not rely on

peculiarities of the models that could not be used to build real physical circuits.

As expected, faithfulness does not hold for arbitrary noise. For a strictly causal involution channel, we showed in [21] that faithfulness holds if the noise interval η fulfills

$$\eta_{\max} + \eta_{\min} < \delta_{\downarrow}(-\eta_{\max}) - \delta_{\min} \ ,$$

where $\delta_{\min}$ is the unique value for which $\delta_{\uparrow}(\delta_{\min}) = \delta_{\min} = \delta_{\downarrow}(\delta_{\min}) > 0$; see Lemma 27 on page 25.

Further reading. The work by Öhlinger *et al.* [25] discusses a tool that integrates the involution delay model into circuits specified in VHDL, and carry out simulations within existing, commercial digital simulation frameworks. The work by Maier *et al.* [27] discusses properties of involution delay functions as emerging from transistor-level properties as opposed to the simplified first-order RC charging considerations for exp-channels presented in this chapter.

Chapter 3

Non-binary Signals

This chapter discusses results from the following research articles.

- [28] Friedrichs, Függer, Lenzen. *Metastability-containing circuits.* IEEE Transactions on Computers, 67(8), 2018.

  Traditionally, synchronous designs lower the probability of metastable setups by waiting long enough. Since waiting costs time, latency is traded for reliability in this case. We show that this is not a necessity, and that rather than to wait, one can compute with metastable values.

- [29] Függer, Kinali, Lenzen, Polzer. *Metastability-aware memory-efficient time-to-digital converter.* In IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), pages 49–56, 2017.

  The work applies the principles of metastability-containing circuit design from [28] and discusses three digital time-to-digital converters (TDCs). The designs provide means to reuse TDCs for high-frequency measurements, are memory efficient, and are particularly well-suited for downstream metastability-containing logic.

- [30] Tarawneh, Függer, Lenzen. *Metastability tolerant computing.* In IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), pages 25–32, 2017.

  This work applies the principles of metastability-containing circuits to state machines. At the example of a Network-on-Chip (NoC) router, we show that such state machines may lead to higher performance while maintaining reliability constraints.

- [31] Függer, Kinali, Lenzen, Wiederhake. *Fast all-digital clock frequency adaptation circuit for voltage droop tolerance.* In IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), pages 68–77, 2018.

  Particularly interesting for applications of metastability-containing circuits are those that involve control with very low latencies. An example is the problem of controlling the clock frequency of a synchronous circuit with respect to high-frequency voltage droops. We present a design based on the techniques developed in [28] and prove it correct.

- [32] Bund, Függer, Lenzen, Medina. *Synchronizer-free digital link controller.* IEEE Transactions on Circuits and Systems I: Regular Papers, 67(10), pages 3562–3573, 2020.

  We show how a metastability-containing link controller between a sender and a receiver can be used to tightly control receiver and sender clocks with low controller latencies.

3.1 Metastability

The existence of an intermediate, so called metastable, state in bistable elements like latches and flip-flops, already played a role in Chapter 2: in fact, for the circuit depicted in Figure 2.8 on page 19 that solves unbounded SPF, we explicitly used the storage loop's behavior near the metastable point when resolving to one of the stable states 0 or 1. Our solution, however, did not guarantee when this will happen — and indeed, we showed that a bounded version of Short Pulse Filtration is not solvable by any circuit. The proof is by reduction to a fundamental result from Marino [9] on bistable storage elements: the existence of a metastable state and an input signal that drives the storage element arbitrarily close to the metastable state, where it resides for an unbounded amount of time before it resolves to stable 0 or 1. Externally, at the output of the storage element, a metastable state may manifest as an intermediate voltage level between a clear 1 and 0, a clear signal followed by a late transition when it resolves, glitches, oscillation, etc. [33, 34]; depending on the implementation [35].

The phenomenon of metastability has a large impact on the design of circuits. Consider the standard clocked Moore implementation in Figure 3.1.

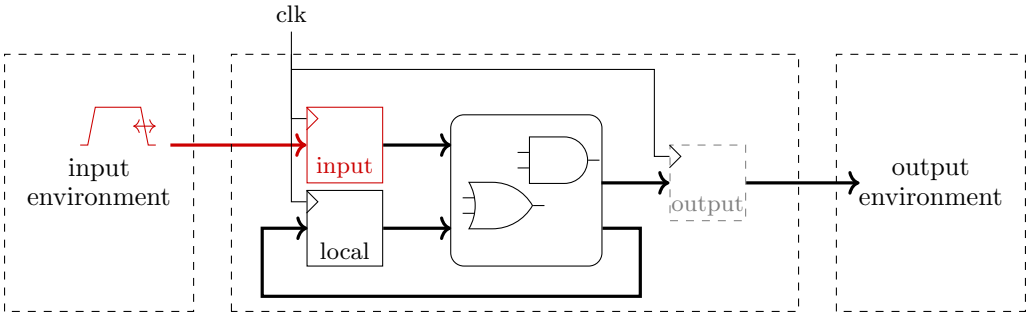


Figure 3.1: Crossing clock domains in in a Moore state machine implementation. The dashed output register is not necessarily part of the implementation. On the left and right circuits that are not under the control of the clock clk, e.g., either unclocked or clocked by a different clock signal.

A realistic design will interact with an input and output environment — on its left, respectively, right, in the figure. Environments may be untimed or clocked with a clock with unknown phase offset. The former, e.g., either because they are implemented without a clock or since they are simply driven by physical phenomena or circuitry that is outside of our control. Reasons for the latter are typically either circuitry that is outside of the designer's control such as IP blocks, or the circuit may have been deliberately partitioned into independent clock domains for performance reasons.

In both cases, input changes may violate setup/hold constraints of the input registers with severe consequences:

1. If the input register becomes metastable, no upper bound on when its output stabilizes exists.
2. By consequence, a priori, there is no bound on when the combinational logic will stabilize.
3. Thus, the next clock transition may arrive when the combinational circuitry has not stabilized yet — with the possibility for a wrong, or even inconsistent, state stored in the local registers. In the worst case, the local registers may become metastable, leading to similar problems as described here in downstream logic.

3.2 Coping with Metastability

Weather a design suffers from metastable upsets, and if so, with which probability, depends on several factors. First, note that metastability, is not a stable state and the slightest distance of the internal state to the metastable point will amplify exponentially over time until the internal state

saturates at a stable state [36–38]. While odds to generate an input pulse that drives the bistable element into metastability for long enough to interfere with the next clock transition may be low for a particular circuit, several factors may very well render such upsets a real problem:

1. *A large number of independent clock domains.* Designs with tens and recently thousands of clock regions on a chip with significant interfacing to the external world and among each other face a proportionally higher upset frequency.

2. *High clock frequencies.* Since, a priori, an upset may occur at every clock cycle, upset frequencies are proportional to clock frequencies.

3. *Aggressive timing, i.e., small clock margins.* This reduces the tolerance to even slightly longer propagation delays through the combinational logic.

4. *Low supply voltage.* Trends to decrease the supply voltage to meet power constraints, lead to increased metastability decay times [39].

Given that there provably is no mitigation technique that completely prevents bistable elements from having upsets, techniques to diminish effects have been developed and are in place in many designs.

3.2.1 Synchronizers

The most widely adopted mitigation technique is to use chains of flip-flops as synchronizers [36, 40, 41]. Each of the flip-flops has preferable metastability decay characteristics compared to a standard data flip-flop [42, 43]. Other known mitigation techniques are Schmitt triggers [44]. But also C-elements, being widely used in asynchronous designs, show synchronizing behavior [45, 46].

The principal idea of a synchronizer is simple and effective: Each synchronizer stage is clocked by the recipient circuit's clock and provides extra decay time for the signal before it is passed to the recipient circuitry, aligned to its clock domain. Since signal amplification is exponential around the metastability point of a storage loop, and given uniform input transition arrival times at input of the first synchronizer stage, the probability of the first stage to be metastable after one clock period decreases exponentially with the length of the clock period. Care must be taken, though, in the analysis since the flip-flop comprises of a master and slave latch: the uniform arrival times, e.g., only (potentially) hold for the first latch. The interested reader is referred to the work by Jones *et al.* [47]. A refined analysis is particularly important when analyzing synchronizers with multiple stages. As a rule of thumb, the probability that the last input stage becomes metastable is exponentially small in the number of stages due to an exponential decay happening in all stages.

While synchronizers may decrease the probability of metastable upsets at the cost of additional input latency, a deterministic guarantee of the absence of metastability is impossible.

3.2.2 Asynchronous Circuits

An orthogonal approach is to abolish clock domain crossings altogether. If the timing in the clock domain is dictated by the arriving signal rather than the region's clock, upsets of the circuit can be prevented. The circuit then waits until it has received a handshake from a yet to stabilize component, using either of:

- The fact that timing constraints are met and thus no metastable upsets can occur. This approach is similar to designing either closed-systems where the controller signals to the external environment when it is ready for a new input, or requires respective timing constraints on the environment. As such it is often not feasible.

- The fact that certain components may get metastable, but that do not produce a valid handshake until metastability is resolved internally. This approach allows for deterministically correct solutions, however, with potentially unbounded timing. A discussion of such designs in our context is given in Section 3.3.3 — for the moment we just note that this does not violate Marino's impossibility result of the existence of a bounded-time, deterministic metastability detector as the response may occur after an unbounded time or may even never

occur. Nonetheless it renders an interesting alternative to clocked designs if fast enough decay times are guaranteed by the target technology.

In both approaches handshaking is a central concept. Typically, handshaking between components in asynchronous circuits is used to wait for all predecessors of a component to complete, before starting computation. Yakovlev *et al.* [48] introduced the concept of OR-causality as an extension to the classical AND-causality that many asynchronous circuits rely on. With classical AND-causality, given that its two predecessors compute a and b, the component then carries on to compute $f(a, b)$ for some function f. In certain scenarios, however, waiting for all predecessors is overly conservative — for example if $f(a, b) = a \vee b$ and $a = 1$. No matter of b's value, the component will produce $f(a, b) = 1$. Thus, the circuit may start computation right after receiving input $a = 1$.

OR-causality, and more generally the concept of ignoring certain inputs, has the potential to also ignore metastable inputs, and not wait arbitrarily long for them to resolve: by this it may tolerate unbounded delays, or even, complete absence of handshakes in data that it is no longer waiting for. However, this is only guaranteed in certain scenarios. A similar example as above, but with $a = 0$ forces the component to wait for its second input. Further, a circuit with OR-causality does not necessarily handle metastability in a correct way. Denote a metastable bit with M. Then, $f(1, 0) = f(1, 1) = 1$ holds, but $f(1, \mathrm{M}) = 1$ may not necessarily hold for an implementation of $f(a, b) = a \vee b$ with OR-causality.

3.2.3 Speculative Computing

Tarawneh *et al.* [49, 50] proposed the concept of speculative computing. Given a single-argument function $f : \mathbb{B} \to \mathbb{B}$, with a potentially metastable input a, they compute $f(0)$ and $f(1)$, and only later on, when a has stabilized, decide whether to output $f(0)$ or $f(1)$. This in fact allows to hide a part of the overall synchronization delay within the computation delay, since both, computation and synchronization, are carried out in parallel. The approach, also works for multiple arguments, by recursively applying it to the arguments. Further, if only the first argument of a function $f(a, b)$ may become metastable, the scheme can be applied to the partial evaluation $b \mapsto f(a, b)$ as well.

3.3 Metastability-Containing Circuits

Both, the synchronizer and the asynchronous approach have in common that they wait before computing. In case of the synchronizers for a fixed time T, and in asynchronous circuits until metastability has decayed. Besides the fact that waiting in high-speed circuits defeats its primary goal, the designer is left to make a decision between:

- *Being time-safe.* Timing guarantees with the downside of potentially metastable values when using synchronizers.
- *Being value-safe.* Value guarantees with the possibility of unbounded or infinite delays when using asynchronous circuits.

With metastability-containing circuits [28] we showed that this tradeoff is not obligatory, however.

3.3.1 Topological View

Marino's proof on the impossibility of bounded-time stabilization [9] is based on the topological statement that, given a connected input space $\mathcal{I}$ and a continuous function $s : \mathcal{I} \to \mathcal{O}$, the output space $\mathcal{O}$ is connected. Let us consider a bistable element, like a synchronizers, with an input port and an output port.[1] To apply the topological statement, the input space is defined as the set of valid input pulses, and the output space is set to $\mathcal{O} = [0, 1]$, i.e., the output's signal value at a certain time $T > 0$ — the time until which we expect the output signal to be stabilized. With standard metrics in place, the input space is connected. The metric on the output space, is simply the Euclidean distance. The major difficulty then is to prove that the function o is indeed

[1] For ease of presentation the clock signal is supposed to be internal to the synchronizer.

continuous under assumptions that capture realistic physical circuits, like causality. Given that there exists an input pulse i_0 for which $s(i_0)$ is arbitrarily close to 0 and an input pulse i_1 with $s(i_1)$ being arbitrarily close to 1, we may apply the topological argument to obtain that there exist input pulses i_M for which $s(i_M)$ is arbitrarily close to, say, 0.5. Figure 3.2 visualizes this argument.

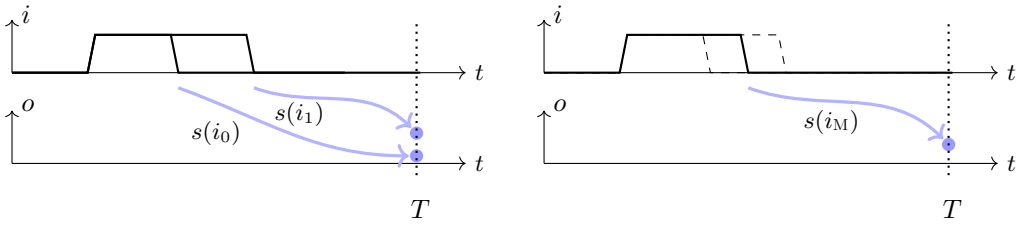


Figure 3.2: Topological argument on the impossibility to stabilize within bounded time $T > 0$.

Synchronizer followed by computation. Figure 3.3 sketches a synchronizer and the subsequent synchronous computation upon the synchronized data from this point of view. The input environment applies input signals that are from a connected space. The input is then synchronized to the new clock domain, e.g., by passing it through a 3 stage synchronizers, which lasts until some time $T > 0$. In fact this synchronization process can be understood as computing the identity function during, say, $T = 3$ clock cycles. While the output space is shown to be connected, with a certain probability, the output signal will be within a disconnected subspace — corresponding to the binary-valued representation of the input signal. Assuming that this is the case, the circuit then computes a function f that finally yields the binary-valued output after another, e.g., 4 clock cycles. The end-to-end latency is thus the sum of both delays; here 7 clock cycles.

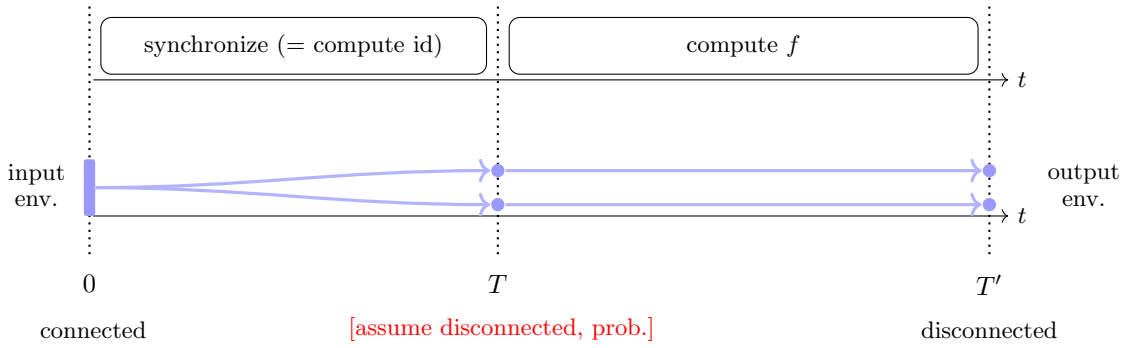


Figure 3.3: Synchronous circuit computing f: synchronize followed by compute.

Metastability-containing computation. The idea of metastability-containing computing is to study computation in a larger, connected, space that includes metastable signals. As such it treats metastability as a valid effect that may occur during the computation, rather than its absence as an assumption external to the model.

While Boolean computation studies computation of functions from input values in some discrete input space $\mathbb{B}^i$, with $i \in \mathbb{N}^+$, to a discrete output space $\mathbb{B}^j$, with $j \in \mathbb{N}^+$, we consider functions from tuples on $\mathbb{B}_M$ to tuples on $\mathbb{B}_M$, where $\mathbb{B}_M = \mathbb{B} \cup \{M\}$. Here, the third value M encodes a *metastable* signal. In our work we do not make any assumption on such a signal — in particular, a metastable signal may have an arbitrary intermediate voltage between 0 and 1, may glitch, oscillate, etc.

From a computational point of view, a synchronizer, if restricted to values in $\mathbb{B}$, computes the identity. So why first compute the identity, before the actual computation of a function f is started? Two properties of the identity function may be responsible for this intuitive choice:

- It preserves precision from the input to the output, no matter if the input/output is encoded in binary, unary, or otherwise. We will later formalize this property of a function, and call such functions metastability-containing.

- It is easily implemented in a way that metastability at the inputs introduces only as much metastability at the outputs as theoretically necessary for such a circuit (in presence of standard registers). In particular, it does output a metastable bit whose corresponding input bit was not metastable. Again, we will formalize this property later on for functions and their circuit implementations and call such circuits metastability-containing.

The question arises, if there are other functions and circuit implementations of these functions that have similar advantageous properties and that additionally already perform (part of) the circuit's intended computation. Designs that make use of these techniques will be referred to as *metastability-containing computations*.

Figure 3.4 sketches the idea of metastability-containing computing from a topological perspective. The difference to the synchronizer is that function f is now computed in a metastability-containing way. The advantage is not only that computing time is spared by removing the synchronization process — see Figure 3.4 (left) — but also that the assumption of stabilization in time is remove — see Figure 3.4 (right). The latter, however, is only applicable if the output environment allows for potentially unstable outputs. Examples where this is the case, are downstream metastability-containing circuitry and applications with a continuous output space, e.g., shifting a signal transition in time, controlling a frequency, voltage, etc. The latter is particularly interesting for controller applications; see Section 3.9.

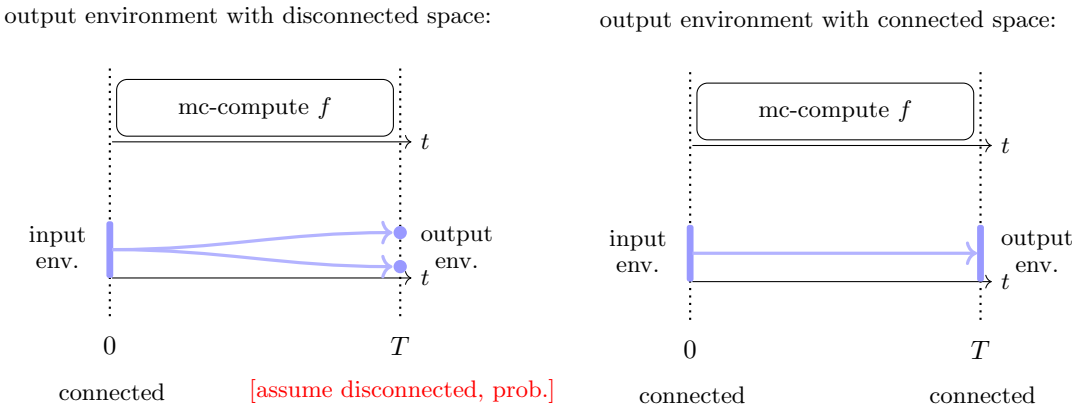


Figure 3.4: Metastability-containing computation of f: compute during synchronization.

3.3.2 Relation to Previous Work

To the best of the author's knowledge, the work on speculative computing [49, 50] is the most closely related to our work: it also aims at parallelizing stabilization and computation and thus fundamentally differs from the synchronizer approach. The differences to our work on metastability-containing circuits [28] are: (i) Speculative computing runs two parallel tracks — synchronization and computation — while the idea in metastability-containing computing is to tightly interweave both to obtain synchronization during computation. (ii) It does not aim for deterministic end-to-end guarantees, since it uses synchronizers that may fail with a certain probability. (iii) The designer must know in advance which bits face the risk of becoming metastable. Speculative computing is then applied to these. By its recursive nature, designs with several metastable candidates quickly become large: k potentially metastable bits result in designs that are larger than by a factor 2^k (duplication and merging). Because of the tight interweaving of computation and synchronization, smaller circuits can be obtained by metastability-containing circuits if the function allows for it. Indeed in this chapter we will encounter several problems for which small circuit solutions exist.

Glitches and hazards are unintended transitions; typically of an output signals. Their systematic study has been pioneered by Huffman [51] and Unger [52]. Different types of hazards have been studied, ranging from static hazards where a multi-bit input x switches to $y \neq x$, and although $f_i(x) = f_i(y)$ for output bit i, two spurious transitions — $f_i(x)$ to $\neg f_i(x)$, and $\neg f_i(x)$ to $f_i(x)$ are observed at the output. While glitches and hazards are spurious, but well-formed, transitions in signals and thus differ from our metastable signals, the methods used to study metastable signals have commonalities with those to study glitches and hazards. For example, static hazards [51] have been addressed with covering prime implicants. In fact we will show that this technique, among others, can also be used to design metastability-containing circuits.

While, initially, two such inputs x, y were restricted to differ only in a single bit, work by Eichelberger [53] extended this to multiple bits and studied dynamic hazards with spurious transitions in switching outputs. Brzozowski and Yoeli [54] extended the simulation algorithm, Brzozowski *et al.* [55] surveyed techniques using higher-valued logics — including Kleene's 3-valued logic — and Mendler *et al.* [56] studied delay requirements needed to achieve consistency with simulated results.

Besides the different assumptions on signal integrity in glitching signals and metastable signals, key differences are:

- *Timing.* Circuits studied with respect to the occurrence of glitches require delay models. Typically pure delay channels are assumed. One can thus bound the time until when outputs are glitch-free. This is no longer true for metastable signals, since metastability may last forever.

- *Glitches.* Metastability-containing circuits are not designed to be free of static, dynamic, or any other glitches. Their outputs may thus contain glitches.

- *Multi-cycle.* Glitches and hazards are typically studied in combinational logic, only — and thus for one clock cycle. Our circuits operate over multiple clock cycles. In fact we will show that such circuits result in a strictly larger class of functions that can be implemented; see Section 3.7.3.

- *Register types.* We studied different register types, that behave differently with respect to internal metastability.

3.3.3 Computational Model

Consider a setup of a synchronous finite state machine implementation, similar to the implementation of the Moore machine in Section 1.3 and the circuit in Figure 3.1 on page 34, expect that the input is not newly latched in every clock cycle. The circuit is shown in Figure 3.5 — note the missing clock input at the input register. While a realistic circuit will have circuitry to load data into the input register, for simplicity we abstract from this and assume that the register is initialized and not written anymore during computation. We next discuss differences between the computational model proposed in [28] and the classical Boolean circuit model.

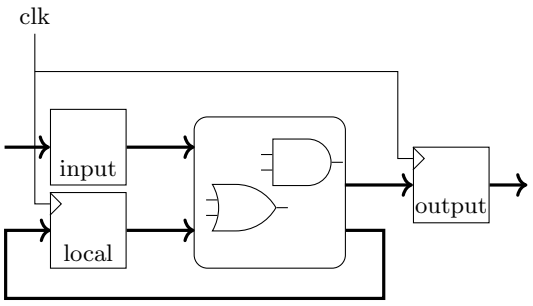


Figure 3.5: Clocked Moore state machine implementation with pre-initialized input registers.

Round-wise model. We do not explicitly model continuous evolution of signals, but assume that the clock period is chosen such that non-metastable data has stabilized at the occurrence of the next active (rising) clock transition. Time thus progresses in discrete rounds $r \in \mathbb{N}^+$, where round r comprises of:

1. *Read phase:* read all input and local registers. We will define register behavior in the following. The read phase models the time right after clock transition $r-1$ for $r > 1$, and the initial values for $r = 1$.

2. *Evaluation phase:* the read values are propagated through the combinational logic. We will define the gates' behavior in the following. This phase makes up the largest part of round r.

3. *Write phase:* the values at the output of the combinational logic finally have the possibility to resolve: An output that is stable, i.e., in $\mathbb{B}$, is not changes, and an output that is M, is allowed to take any value in $\mathbb{B}_{\mathrm{M}}$. The value is then the new state of the local or output register. This phase corresponds to the arrival of the r^{th} rising clock transition, ending round r.

Instead of considering the detailed timed executions of such circuits, we will work with a discrete-time abstraction. A *configuration* of a circuit is a tuple that contains the states of the input, local, and output registers. An *execution* of a circuit is a sequence $(s_r)_{r\in\mathbb{N}}$ of configurations, with s_0 being the initial configuration, and s_r, with $r \in \mathbb{N}^+$, obtained from s_{r-1} by application of the read, evaluation, and write phase. Note that a circuit has to define the initial local and output register states. Given a circuit C, the set of output register states reachable via an execution in $r \in \mathbb{N}$ rounds from input register states ι is denoted by $C_r(\iota)$.

A *specification* is a function $\mathbb{B}_{\mathrm{M}}^m \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^n)$ that maps a vector of input values to the set of allowed output values. We say r *rounds of circuit* C *implement a specification* $s : \mathbb{B}_{\mathrm{M}}^m \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^n)$, if for all inputs $\iota \in \mathbb{B}_{\mathrm{M}}^m$, and all executions of circuit C with inputs ι, after r rounds, the vector of circuit outputs o in this execution is in $s(\iota)$, i.e.,

$$\forall \iota \in \mathbb{B}_{\mathrm{M}}^m \,:\, C_r(\iota) \subseteq s(\iota) \ .$$

We say C *implements* s, if there exists an $r \in \mathbb{N}$ such that r rounds of C implement s. Instead of C implementing s, we also speak of C *computing* s if the focus is on the functional aspect of s.

It remains to specify the behavior of circuit gates and registers.

Gates. Similar to their classical Boolean counterparts, a gate with $k > 0$ inputs is a function $\mathbb{B}_{\mathrm{M}}^k \to \mathbb{B}_{\mathrm{M}}$. However, care has to be taken in which functions we allow in order to capture real word circuits. For example, a gate that maps M to 1, and all other values to 0 would be a bounded-time metastability detector; a device that does not exist.

For that purpose, we define resolutions and extensions of Boolean gates based on resolutions as follows. Let $x \in \mathbb{B}_{\mathrm{M}}^k$. The set of *partial resolutions of* x, denoted by $\mathrm{Res}_{\mathrm{M}}(x)$, is defined as

$$\mathrm{Res}_{\mathrm{M}}(x) = \left\{ y \in \mathbb{B}_{\mathrm{M}}^k \mid \forall i \in [k] \colon x_i = y_i \vee x_i = \mathrm{M} \right\} \ . \tag{3.1}$$

The set of *resolutions*, denoted by $\mathrm{Res}(x)$, is defined as

$$\mathrm{Res}(x) = \mathrm{Res}_{\mathrm{M}}(x) \cap \mathbb{B}^k \ . \tag{3.2}$$

For a Boolean function $f : \mathbb{B}^k \to \mathbb{B}$, we define its *metastable extension* $f_{\mathrm{M}} : \mathbb{B}_{\mathrm{M}}^k \to \mathbb{B}_{\mathrm{M}}$ as

$$f_{\mathrm{M}}(x) = \begin{cases} 0 & \text{if } \forall x' \in \mathrm{Res}_{\mathrm{M}}(x) \colon \ f(x') = 0 \\ 1 & \text{if } \forall x' \in \mathrm{Res}_{\mathrm{M}}(x) \colon \ f(x') = 1 \\ \mathrm{M} & \text{otherwise} \end{cases} \ . \tag{3.3}$$

We only allow gates that that are metastable extensions of Boolean functions. The reason is that standard Boolean CMOS gates like the 2 input NAND and NOR gates, if implemented correctly, readily compute the metastable extension of their Boolean functions.

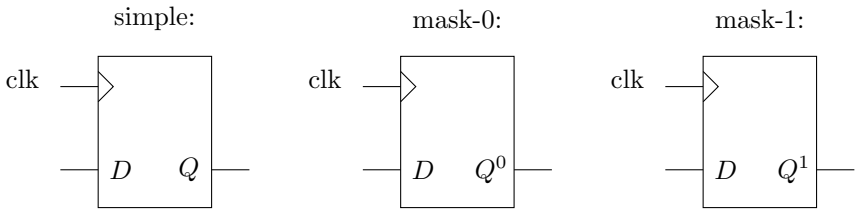


Figure 3.6: Three register types.

Registers. We consider three register types. Figure 3.6 shows their gate symbols.

- *Simple registers.* These are the classical used registers without any additional assumption on their behavior on metastable states. They simply output their internal state. We assume that input registers never change their internal state. While this is obviously no restriction for a stable internal state in $\mathbb{B}$, we claim that it is also not a restriction for a metastable internal state: any stabilization would actually diminish the reachable executions; and thus this captures the worst-case.
- *Masking registers.* There are two types of masking registers — *mask*-0 and *mask*-1. With $b \in \mathbb{B}$, a mask-b register maps an internal metastable state to b at its output. By contrast to simple registers, masking input registers may change their state during the read phase. If they are internally metastable, they may transition to any state in $\mathbb{B}_{\mathrm{M}}$. If they were in state M and they transition to b, then b is their new state, and we read b at their output during the read phase. If they were in state M and they transition to $\neg b$, then $\neg b$ is their new state, and we read M at their output during the read phase.

While simple registers are sufficiently unconstrained to arguably capture most existing register implementations, the behavior of masking registers is non-standard. Indeed, noise-tolerant implementations are a topic of current research. We will refer to an implementation based on metastability filters as discussed by Kinniment [36] (Section 3.1), here. Recall the unbounded SPF circuit from Section 2.8.5. It relied on a low-threshold inverter after a storage loop with a high metastability point; see Figure 2.12 on page 25. We may use the same idea and let the flip-flop output drive high- or low-threshold inverters, amplifying an internal metastable signal to 1 or 0.

Such filters are also used in asynchronous designs: for example the Blade design style [57], based on bundled data communication between modules, makes use of metastability filtering techniques. Aggressive timing of the request/acknowledge signals with respect to the data signals may result in metastable Blade controller states. In this case, metastability filters are used to locally halt computation (by delaying req/ack) until metastability is resolved. This allows for more aggressive timing than in synchronous pipelines, where large clock margins have to account for all cases.

Another interesting recent applications is within a mutex to create a (local) pausible clock that halts until internal metastability is resolved [58].

3.4 Metastability-Containing Multiplexers

Let us demonstrate the key questions at the example of a small, but widely-used circuit, a multiplexer. The choice of the multiplexer as a running example is motivated by a second advantageous fact: it will play a central role in several metastable-containing circuits.

3.4.1 Standard Multiplexer

Definition 38 (Multiplexer)**.** *A* (k-bit) multiplexer (MUX) *is a circuit with* $2k+1$ *inputs and* k *outputs, that implements* $f_{\mathrm{MUX}} \colon \mathbb{B}_{\mathrm{M}}^k \times \mathbb{B}_{\mathrm{M}}^k \times \mathbb{B}_{\mathrm{M}} \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^k)$ *with*

$$f_{\mathrm{MUX}}(a,b,s) = \begin{cases} \mathrm{Res}_{\mathrm{M}}(a) & \text{if } s = 0 \\ \mathrm{Res}_{\mathrm{M}}(b) & \text{if } s = 1 \\ \mathbb{B}_{\mathrm{M}}^k & \text{if } s = \mathrm{M} \end{cases} .$$

When we speak of a multiplexer, we refer to the case $k = 1$, unless noted otherwise. Its behavior, for binary signals is as expected: the select bit s, determines whether to output a or b. In presence of metastable inputs a and b, it only guarantees that a solution of the output takes on a resolution of the selected input. If the select bit is metastable, an arbitrary output may be produced.

Example 39. *For the single bit multiplexer, with $k = 1$, we have $f_{\text{MUX}}(1, \text{M}, 0) = \text{Res}_\text{M}(1) = \{1\}$, $f_{\text{MUX}}(1, \text{M}, 1) = \text{Res}_\text{M}(\text{M}) = \mathbb{B}_\text{M}$, and $f_{\text{MUX}}(1, \text{M}, \text{M}) = \mathbb{B}_\text{M}$.*

Figure 3.7 shows the schematics of the combinational circuit SMUX that resembles the standard gate-level circuit implementation of a multiplexer. Indeed the circuit implements MUX in our model, i.e., is a multiplexer. Note that technically, simple input and output registers are required by our model. Since this is a single round computation, we omit them.

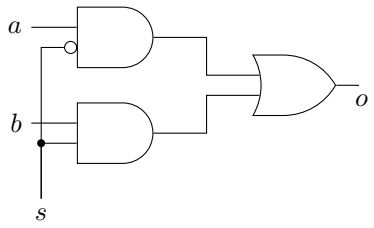


Figure 3.7: Gate-level multiplexer circuit SMUX that implements MUX.

3.4.2 Stronger Guarantees

In case of Boolean input values, and if $a = b$, the specification of a multiplexer implies that its output is a. Care must be taken in presence of metastable inputs. The specification of a multiplexer MUX does not constrain the output if $s = \text{M}$. Indeed, the circuit SMUX sets its output to M if $s = \text{M}$ and $a = b = 1$.

The question arises, if we can ask for more from a multiplexer-type circuit. Intuitively, a multiplexer that is unsure which input to forward to the output, but whose both inputs are identical and stable, should output the stable, identical, input value. Cast into a specification, we require:

Definition 40 (Metastability-Containing Multiplexer. From [28])**.** *A (k-bit) Metastability-Containing Multiplexer (CMUX) is a circuit with $2k+1$ inputs and k outputs, that implements $f_{\text{CMUX}} \colon \mathbb{B}_\text{M}^k \times \mathbb{B}_\text{M}^k \times \mathbb{B}_\text{M} \to \mathcal{P}(\mathbb{B}_\text{M}^k)$ with*

$$f_{\text{CMUX}}(a, b, s) = \begin{cases} \text{Res}_\text{M}(a) & \text{if } s = 0 \vee a = b \\ \text{Res}_\text{M}(b) & \text{if } s = 1 \\ \mathbb{B}_\text{M}^k & \text{if } s = \text{M} \wedge a \neq b \end{cases}.$$

Indeed, metastability-containing multiplexer exists. We showed in [28] that 1 round of CMUX1, depicted in Figure 3.8 on page 43, implements f_{CMUX}. Thus:

Lemma 41 ([28])**.** *Circuit* CMUX1 *is a* CMUX*.*

We will next discuss a two round implementation. The reader may wonder why we continue to find a slower implementation after solving the problem in a single round. Further, the two round implementation requires additional local registers to store the circuit's state. Our motivation is twofold: First, the two round solution demonstrates how masking registers can be used to solve problems, and second the proposed technique can be generalized and can lead to potentially efficient circuits for more complex problems.

For simplicity, we state the circuit in terms of pseudocode in Algorithm 2 — note that this can be readily translated to a synchronous circuit. The algorithm uses the fact that out of two reads from a masking register, only one can be metastable. For the notorious case of $s = \text{M}$ and $a = b = 1$

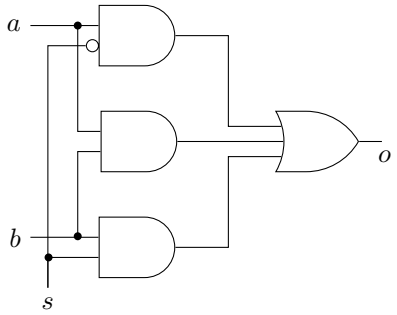


Figure 3.8: The metastability-containing multiplexer CMUX1.

Algorithm 2 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

where circuit MMUX failed to implement CMUX, we obtain the following behavior: Since we are using a mask-1 register to store s, if $s = \mathrm{M}$, the two sequential reads of s will yield $1, 1$ or $\mathrm{M}, 0$ or $1, \mathrm{M}$. Accordingly, these are also the states of s', s in round 2 — assuming s' did not resolve. One finally observes that in all these scenarios at least one AND clause is stable 1, setting the output to 1. We thus have that two rounds of the circuit corresponding to Algorithm 2 implement f_{CMUX}. Thus:

Corollary 42 ([28])**.** *The circuit corresponding to Algorithm 2 is a* CMUX.

An optimized version of this circuit is obtained by inserting a delay between signals s and s' instead of explicitly separating them by a clock cycle. By doing so, one obtains the circuit in Figure 3.9. The technique of delaying outputs from masking registers rather than sampling them in separate rounds has wider applications not detailed here.

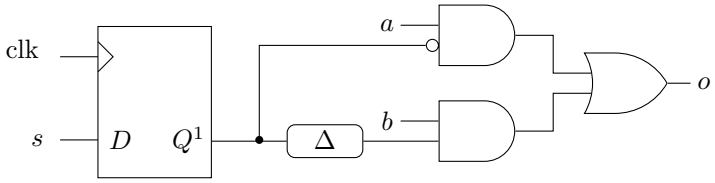


Figure 3.9: Metastability-containing multiplexer using a 1-masking register.

3.5 Properties of Synchronous Circuits under Metastability

We readily obtain several properties of circuits in our circuit model. The properties are not only useful in proving results about computability later on, but also to show that the model captures physical circuits faithfully. Like the SPF problem in Chapter 2 was used to determine the faithfulness of delay models with respect to glitch propagation, we will discuss analogous problems here to determine the faithfulness of our model with respect to propagation of metastability.

We start with a property of purely combinational circuits. For such circuits, stabilizing inputs do not introduce additional metastability at the outputs. Rather, unstable outputs remain unstable, or they stabilize. The following lemma states this in terms of our model:

Lemma 43 ([28])**.** *Let C be a purely combination circuit with m inputs and let $f^C : \mathbb{B}_M^m \to \mathbb{B}_M^n$ be the function that maps C's input to its (unique, since C only comprises of gates) output with n bits. Then for all inputs $\iota \in \mathbb{B}_M^m$,*

$$\iota' \in \mathrm{Res}_M(\iota) \Rightarrow f^C(\iota') \in \mathrm{Res}_M\left(f^C(\iota)\right) \quad . \tag{3.4}$$

The property also demonstrates that our assumption of simple registers remaining metastable throughout an execution is indeed the worst-case. Any resolution of them cannot lead to more metastability at the circuit outputs. The situation is different for masking registers, as we will see later on — this is also why resolution is explicitly modeled in masking registers.

A further basic property is that output registers, written at the end of the first round, resolve independently of each other:

Corollary 44 ([28])**.** *For any circuit C with m input and n non-input registers,*

$$C_1 = g_0 \times \cdots \times g_{n-1} \quad ,$$

where

$$g_i \colon \mathbb{B}_M^m \to \{\{0\}, \{1\}, \mathbb{B}_M\}$$

is the function that corresponds to non-input register $i \in \{0, \ldots, n-1\}$ and maps the input to its possible outputs.

The result implied that we can treat non-input register states independently of each other, at least in the first round. We will later see that this generalizes to multiple rounds for simple registers. Again masking registers make an exception — the property does not hold for rounds larger than 1.

Based on Corollary 44, we may lift Lemma 43 from covering just the combinational logic to full single-round computation, and obtain:

Corollary 45 ([28])**.** *For a circuit C with m inputs, and for an input $\iota \in \mathbb{B}_M^m$,*

$$\iota' \in \mathrm{Res}_M(\iota) \Rightarrow C_1(\iota') \subseteq C_1(\iota) \quad .$$

3.6 Faithfulness

While one may wish for circuits that reliably either detect or filter metastability within bounded time, reductions to Marino's work [9] prove this impossible in physical circuits. Let us state both problems in our model.

Definition 46 (Synchronous metastability-detector and metastability-filter)**.** *A synchronous metastability-detector is a circuit that implements the function $f \colon \mathbb{B}_M \to \mathcal{P}(\mathbb{B}_M)$ with*

$$f(x) = \begin{cases} \{1\} & \text{if } x = \mathrm{M} \\ \{0\} & \text{otherwise} \end{cases} \quad . \tag{3.5}$$

A synchronous metastability-filter is a circuit that implements the function $f \colon \mathbb{B}_M \to \mathcal{P}(\mathbb{B}_M)$ with

$$f(x) = \begin{cases} \{0,1\} & \text{if } x = \mathrm{M} \\ \{x\} & \text{otherwise} \end{cases} \quad . \tag{3.6}$$

For physical circuits the following holds, by arguments analogous to the ones given in Section 2.5 for the bounded Short Pulse Filtration problem:

Theorem 47. *No physical circuit exists that is a synchronous metastability-detector or a synchronous metastability-filter.*

We thus call a circuit model *faithful with respect to metastability detection and filtration* if within this model, no circuit exists that is a synchronous metastability-detector or a synchronous metastability-filter.

In [28] we have shown that in our synchronous circuit model, for any circuit which has to output different results for at least two different inputs there exists an input that produces metastable outputs.

Theorem 48 ([28])**.** *Let $r \in \mathbb{N}^+$ and C be a circuit with $C_r(\iota) \cap C_r(\iota') = \emptyset$ for some $\iota, \iota' \in \mathbb{B}_M^m$. Then C has an r-round execution in which an output register becomes metastable.*

The proof relies on the construction of so called pivotal sequences. These are finite sequences of stable Boolean inputs of which the first element is ι, the last element is ι', and two successive elements differ in at most one input bit. Within this sequence a switch in corresponding outputs must occur between two successive inputs differing in only a single bit. We then construct an input from these two Boolean inputs that leads to metastability for its corresponding output.

By instantiating Theorem 48 for two functions, one readily shows that the previous two problems are not solvable in our circuit model:

Corollary 49 ([28])**.** *There exists no circuit in the synchronous circuit model that is a synchronous metastability-detector or a synchronous metastability-filter.*

We thus finally obtain:

Theorem 50. *The synchronous circuit model is faithful with respect to metastability detection and filtration.*

3.7 Computability

In [28] we showed that simple and masking registers do not only differ in the functions they compute, but that there exists a non-trivial computational hierarchy in presence of masking registers. The writeup discusses theses results in the following. We start with defining several classes of functions related to computability within our model.

Definition 51 (Computable function classes, from [28])**.** *We define the following classes of functions, with $r \in \mathbb{N}^+$:*

- Fun_S^r *is the class of functions implementable with r rounds of circuits comprising only simple registers.*
- Fun_M^r *is the class of functions implementable with r rounds of circuits that may additionally use masking registers.*

3.7.1 The Computational Hierarchy

As a first result, one readily obtains that the first classes match.

Corollary 52 ([28])**.** $\mathrm{Fun}_S^1 = \mathrm{Fun}_M^1$.

Further, masking registers only play a role if used as input registers. Using them as local or output registers does not provide different behavior. Note, however that this statement is within our worst-case synchronous circuit model. There may very well be, and in fact is, different behavior in a probabilistic analysis. Treatment of these effects, however, is outside the scope of this writeup.

Corollary 53 ([28])**.** *Simple and masking registers are interchangeable when used as non-input registers.*

Finally, we obtain the existence of a hierarchy among the classes. Adding computational rounds never decreases the class of computable function. The proof of this result uses simulation of shorter executions and delaying outputs. Furthermore, allowing masking registers does not decrease the computational power. In terms of the function classes, we thus have:

Corollary 54 ([28])**.** *For all $r \in \mathbb{N}^+$, we have*

- $\mathrm{Fun}_S^r \subseteq \mathrm{Fun}_S^{r+1}$,
- $\mathrm{Fun}_M^r \subseteq \mathrm{Fun}_M^{r+1}$, *and*
- $\mathrm{Fun}_S^r \subseteq \mathrm{Fun}_M^r$.

Note that the corollary leaves it open whether the complete hierarchy collapses to a single class of computable functions. We will soon see that this is not the case.

3.7.2 Simple Registers

The technique of circuit unrolling is folklore in circuit design for circuits with binary-valued gates: given a multi-round computation over $r > 1$ rounds, the combinational circuit may be unrolled $r - 1$ times instead of storing intermediate results in local registers. Indeed, circuits with simple registers can be unrolled also in presence of metastability:

Theorem 55. *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.*

While from a computational perspective, multi-round circuits seem superfluous, they clearly play a role when circuit size does.

Theorem 55 finally yields the collapse of the hierarchy for simple registers:

Corollary 56. *For all $r \in \mathbb{N}^+$, $\mathrm{Fun}_S^r = \mathrm{Fun}_S^1$.*

We may thus omit the index r in Fun_S^r and denote the class by Fun_S.

Characterization. A question of practical relevance is which functions are indeed implementable with only simple registers, i.e., a characterization of Fun_S. For that purpose let us define natural functions and subfunctions:

Definition 57 (Natural functions and subfunctions, from [28])**.** *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*, defined as follows:*

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

$$f(\iota) = f_1(\iota) \times \cdots \times f_n(\iota). \tag{3.7}$$

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

$$\forall \iota \in \mathbb{B}_{\mathrm{M}}^m\colon \quad f(\iota) \in \{\{0\}, \{1\}, \mathbb{B}_{\mathrm{M}}\}^n. \tag{3.8}$$

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

$$\forall \iota \in \mathbb{B}_{\mathrm{M}}^m\colon \quad \iota' \in \mathrm{Res}(\iota) \Rightarrow f(\iota') \subseteq f(\iota). \tag{3.9}$$

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

With these definitions in place, Fun_S is characterized as:

Theorem 58 ([28]). *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.*

The if-direction of the theorem's proof is constructive and uses a strategy also used in designing hazard-free circuits for static hazards [51]: it covers prime-implicants of g_i, with $i \in [n]$. The construction, however, may result in circuits that are exponentially large in m.

Certain functions are particularly well suited as being implemented by circuits: they do not only fall into the class Fun_S, but are also minimal within this class, in a sense that will be made precise in the following. Recalling the metastable extension $f_{\mathrm{M}} : \mathbb{B}_{\mathrm{M}}^m \to \mathbb{B}_{\mathrm{M}}$ of a Boolean function $f : \mathbb{B}^m \to \mathbb{B}$, we define:

Definition 59 (Metastable closure, from [28]). *For a Boolean function $f\colon \mathbb{B}^m \to \mathbb{B}^n$, 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*

$$[f]_{\mathrm{M}}(x)_i = \mathrm{Res}_{\mathrm{M}}\left((f_i)_{\mathrm{M}}(x)\right) = \begin{cases} \{0\} & \text{if } (f_i)_{\mathrm{M}}(x) = 0 \\ \{1\} & \text{if } (f_i)_{\mathrm{M}}(x) = 1 \\ \mathbb{B}_{\mathrm{M}} & \text{if } (f_i)_{\mathrm{M}}(x) = \mathrm{M} \end{cases}. \tag{3.10}$$

For a function $f\colon \mathbb{B}_{\mathrm{M}}^m \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^n)$, we define its metastable closure $[f]_{\mathrm{M}}\colon \mathbb{B}_{\mathrm{M}}^m \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^n)$ by

$$[f]_{\mathrm{M}}(x)_i = \begin{cases} \{0\} & \text{if } \forall x' \in \mathrm{Res}_{\mathrm{M}}(x')_i = \{0\} \\ \{1\} & \text{if } \forall x' \in \mathrm{Res}_{\mathrm{M}}(x')_i = \{1\} \\ \mathbb{B}_{\mathrm{M}} & \text{otherwise} \end{cases}. \tag{3.11}$$

Example 60. *For the Boolean* AND *input-output function $f : \mathbb{B}^2 \to \mathbb{B}$ with $f(a,b) = a \wedge b$, the metastable closure $[f]_{\mathrm{M}}$ is*

$$f(a,b) = \begin{cases} \{a \wedge b\} & \text{if } a \neq \mathrm{M} \wedge b \neq \mathrm{M} \\ \{0\} & \text{if } a = \mathrm{M} \wedge b = 0 \\ \{0\} & \text{if } a = 0 \wedge b = \mathrm{M} \\ \mathbb{B}_{\mathrm{M}} & \text{if } a = \mathrm{M} \wedge b \in \{\mathrm{M}, 1\} \\ \mathbb{B}_{\mathrm{M}} & \text{if } a \in \{\mathrm{M}, 1\} \wedge a = \mathrm{M} \end{cases}. \tag{3.12}$$

By construction, for a function $f\colon \mathbb{B}_{\mathrm{M}}^m \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^n)$, its metastable closure $[f]_{\mathrm{M}}$ is bit-wise, closed, specific, and hence natural. The same holds true for Boolean functions f.

Corollary 61 ([28]). *For all Boolean functions $f\colon \mathbb{B}^m \to \mathbb{B}^n$, it is $[f]_{\mathrm{M}} \in \mathrm{Fun}_S$. Further, for all functions $f\colon \mathbb{B}_{\mathrm{M}}^m \to \mathcal{P}(\mathbb{B}_{\mathrm{M}}^n)$, it is $[f]_{\mathrm{M}} \in \mathrm{Fun}_S$.*

Metastability-containing circuits. Corollary 61 has direct applications for implementing Boolean functions by a circuit in a way that the circuit provably optimally contains metastability. While Theorem 48 shows that non-constant functions must have inputs that lead to metastable outputs, Corollary 61 provides the answer to how much metastability we need to accept in presence of metastable inputs: For every Boolean function $f\colon \mathbb{B}^m \to \mathbb{B}^n$, there is a circuit with only simple registers that implements $[f]_{\mathrm{M}}$. Theorem 58 shows that $[f]_{\mathrm{M}}$ is the minimum extension of f implementable with simple registers: any of its proper subfunctions is not implementable. Thus, the circuit is optimally with respect to containing metastability. Motivated by these considerations, we define:

Definition 62 (Metastability-containing circuit). *We call a circuit* metastability-containing *if it implements the metastable closure of a function.*

Note that this is a property of a circuit. We will later also use metastability-containing as a property of a function. Like their circuit counterparts, these functions will behave optimally with respect to metastable inputs; more about this in Section 3.8.

3.7.3 Masking Registers

If the computational hierarchy collapses for simple registers, does it also for masking registers? To answer this question, let us define the n-bit metastability-containing replication problem for $n \in \mathbb{N}^+$.

Definition 63 (Metastability-containing replication, from [28])**.** *For $n \in \mathbb{N}^+$, we say a circuit solves the n-bit metastability-containing replication problem if it implements $f_{\text{MCR},n} : \mathbb{B}_\text{M} \to \mathbb{B}_\text{M}^n$ with*

$$f_{\text{MCR},n}(\iota) = \begin{cases} \{\iota^n\} & \text{if } \iota \neq \text{M} \\ \bigcup_{i\in[n]} \text{Res}_\text{M}(0^i\text{M}1^{n-i-1}) & \text{otherwise} \end{cases} . \tag{3.13}$$

Essentially, the problems asks a circuit to replicate a single input bit n times. Classically in non-metastable circuits this is trivially solved by driving n outputs directly or indirectly via buffers. The problem, however, becomes interesting and of practical relevance, as we will see, if the input bit is metastable. While in circuits with simple registers it is easily shown that no restrictions can be made on the states of the output registers at all, we obtained:

Lemma 64. *There exists a circuit C with masking registers, such that $n \in \mathbb{N}^+$ rounds of C implement $f_{\text{MCR},n}$, i.e., that solves the n-bit metastability-containing replication problem.*

The circuit is deceptively simple: it copies the input bit, stored in a 0-masking register, to register r in round $r \in \mathbb{N}^+$. Interestingly, this procedure cannot be unrolled — rendering circuits with masking registers fundamentally different from classical circuits. Indeed, we have:

Lemma 65. *There exists no circuit C such that $\ell \in \mathbb{N}^+$ rounds of C implement $f_{\text{MCR},n}$ with $\ell < n$, i.e., that solves the n-bit metastability-containing replication problem in $\ell < n$ rounds.*

Lemmas 64 and 65 finally yield the existence of a non-trivial hierarchy:

Theorem 66 ([28])**.** *For all $r \in \mathbb{N}^+$, we have $\text{Fun}_M^r \subsetneq \text{Fun}_M^{r+1}$.*

A characterization of the classes Fun_M^r is an open problem and subject to ongoing research.

Efficient circuits. Theorem 58 on page 47 presented an exponential size construction to implement functions by metastability-containing circuits if the function is in Fun_S. In particular this allows to construct circuits that implement $[f]_\text{M}$ for a Boolean function f. In [28] we also showed a different construction that inherently uses masking registers to achieve only a linear blowup in gate count:

Theorem 67 ([28])**.** *Let $f\colon \mathbb{B}^m \to \mathbb{B}^n$ be a Boolean function and G a purely combinational circuit with input-output behavior $f^G(\iota) = f(\iota)$ for all $\iota \in \mathbb{B}^m$. Then there is a circuit C such that $2m+1$ rounds of C implement $[f]_\text{M}$. The circuit uses*

- *m mask-0 input registers,*
- *$(2m+1)n$ simple local registers, and*
- *n simple output registers.*

The additive overhead in complexity with respect to G is $O(nm \log m)$ in gate count and $O(\log m)$ in depth.

Figure 3.10 on page 49 sketches the circuit's data path that is at the heart of the construction in Theorem 67. The circuit first solves the $2m+1$-bit metastability-containing replication problem — but instead of computing the identity ι it replicates $f_i(\iota)$, for each output bit i. The majority of the replicas is then computed in each round, with the result written to output register i. Note that the output register may be set to M in round $2m+1$; the proof, however, shows that this is only the case if $(f_\text{M})_i(\iota) = \mathbb{B}_\text{M}$, i.e., this output bit is unrestricted. Further, note that the majority circuit is non-trivial since it must be metastability-containing. If it were not metastability-containing, a single M may overwhelm a majority of binary values. Indeed, circuits that follow this scheme to solve problems are of use as we will discuss in Section 3.9.

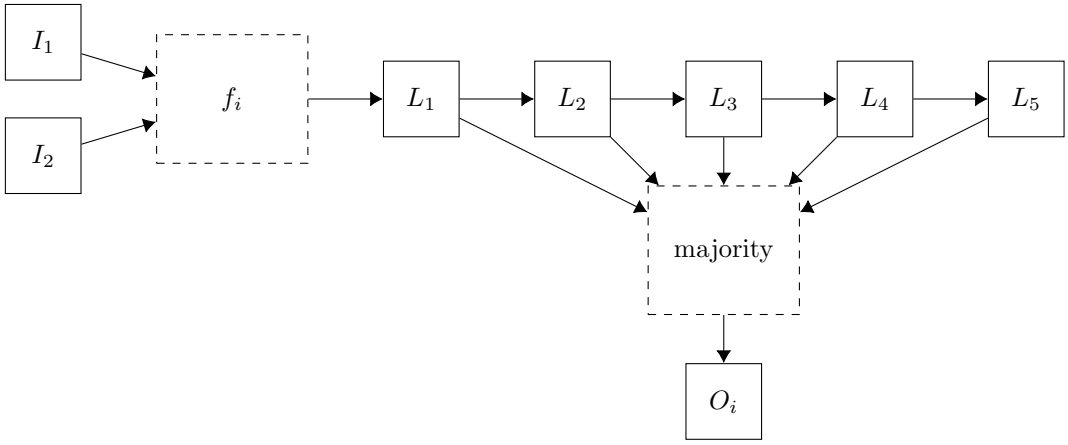


Figure 3.10: Construction for efficiently implementing the metastable-closure $[f]_{\mathrm{M}}$ of a Boolean function f presented in [28]. The depicted circuit is for two input registers and only shows the circuitry for output register i.

3.8 Metastability-containing Functions

Definition 62 on page 47 specified metastability-containing as a property of circuits that, in a sense made clear in the definition, optimally contain metastability. They do not "unnecessarily" propagate metastability to the outputs, unless not otherwise possible; at least for circuits with only simple registers. In that sense, a circuit that is supposed to compute the projection $(x, y) \to x$ and does so by a single buffer is metastability-containing, while another one that does so by a combinational logic reflecting the term $(y \vee (\neg y)) \wedge x$ is not: for $x = 1$ and $y = \mathrm{M}$, it is $(\mathrm{M} \vee (\neg \mathrm{M})) \wedge x = \mathrm{M}$.

In the following we will define metastability-containing as a property of functions on natural numbers with a similar intention. Towards this end, we start with formalizing encodings for natural numbers.

A *code* is an injective function $\gamma\colon \{0, \ldots, n-1\} \to \mathbb{B}^k$ mapping an $x \in \{0, \ldots, n-1\}$ to its *encoded* representation. For y such that $y = \gamma(x)$ for some x, we define the inverse $\gamma^{-1}(y) = x$. For a set X, we follow the convention of writing $\gamma(X)$ for the set $\{\gamma(x) \mid x \in X\}$ and $\gamma^{-1}(X)$ for the set $\{x \mid \gamma(x) \in X\}$. Instead of $\gamma(n)$, with $n \in \mathbb{N}$, we also write $(n)_\gamma$, i.e., the number n encoded by code γ.

Example 68. *For the k-bit thermometer code, also referred to as unary code, $\gamma = \mathrm{un}$, with $\mathrm{un}\colon \{0, \ldots, k\} \to \mathbb{B}^k$, encoded numbers are given by $\mathrm{un}(0) = 0 \ldots 0$, $\mathrm{un}(1) = 0 \ldots 01$, $\mathrm{un}(2) = 0 \ldots 011$, etc. Further, for $k = 5$, we have $\mathrm{un}^{-1}(01111) = 4$ and $\mathrm{un}^{-1}(00101)$ does not exist.*

Example 69. *For the k-bit binary code, $\gamma = \mathrm{bin}$, with $\mathrm{bin}\colon \{0, \ldots, 2^k - 1\} \to \mathbb{B}^k$, encoded numbers are given by $\mathrm{un}(0) = 0 \ldots 0$, $\mathrm{un}(1) = 0 \ldots 01$, $\mathrm{un}(2) = 0 \ldots 010$, etc.*

Example 70. *For the k-bit binary reflected gray code (BRGC), $\gamma = \mathrm{rg}$, with $\mathrm{rg}\colon \{0, \ldots, 2^k - 1\} \to \mathbb{B}^k$ with $\mathrm{un}(0) = 0 \ldots 0$, $\mathrm{un}(1) = 0 \ldots 01$, $\mathrm{un}(2) = 0 \ldots 011$, $\mathrm{un}(3) = 0 \ldots 010$, etc.*

With respect to space efficiency, the unary code is exponentially less efficient than the binary code and BRGC — the latter two codes are in fact bijections, making them optimal.

Formalizing the notion of the amount of metastability in a code word we define:

Definition 71 (Precision, from [28])**.** *Code word $x \in \mathbb{B}^k_\mathrm{M}$ has* precision p with respect to the code γ *if*

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

i.e., if the largest possible difference between resolutions of x is bounded by p. The precision of x with respect to code γ is undefined if some $y \in \mathrm{Res}(x)$ is no code word.

Given a code γ, the Boolean increment function $f_{\mathrm{INC},\gamma}$ is defined as

$$f_{\mathrm{INC},\gamma}(u, (x)_\gamma) = \begin{cases} \{(x)_\delta\} & \text{if } u = 0 \\ \{(x+1)_\delta\} & \text{if } u = 1 \end{cases} \ . \tag{3.14}$$

The increment function $f_{\mathrm{INC},\gamma}$ on not necessarily Boolean values is defined as $[f_{\mathrm{INC},\gamma}]_\mathrm{M}$.

We finally define:

Definition 72 (Metastability-containing function)**.** *Let γ be a code. Let $f_\gamma : \Pi_{i=1}^k \{(0)_\gamma, \ldots, (D_i)_\gamma\} \to (\mathbb{N})_\gamma$ be a function. We say f_γ is* metastability-containing in component $i \in [k]$*, if: For all $x_1, \ldots, x_k \in \Pi_{j=1}^k \{0, \ldots, D_j\}$ with $x_i < D_i$, if*

$$f_\gamma\left((x_1)_\gamma, \ldots, \{(x_i)_\gamma, (x_i+1)_\gamma\}, \ldots (x_k)_\gamma\right)$$

has precision p, then

$$f_\gamma\left((x_1)_\gamma, \ldots, f_{\mathrm{INC},\gamma}(\mathrm{M}, (x_i)_\gamma), \ldots (x_k)_\gamma\right)$$

has precision p.

From considerations analogous to those in the following Example 73, we have that $f_{\mathrm{INC,bin}}$ is not metastability-containing in its first component, while $f_{\mathrm{INC,un}}$ and $f_{\mathrm{INC,rg}}$ are.

Example 73. *For the three increment functions $f_{\text{INC},bin}$, $f_{\text{INC,un}}$, and $f_{\text{INC,rg}}$, with respective code word lengths $k \in \{4, 9, 5\}$, we observe:*

1. $f_{\text{INC},bin}((0)_{\text{un}}, (7)_{bin}) = (7)_{bin} = 0111$ *and* $f_{\text{INC},bin}((1)_{\text{un}}, (7)_{bin}) = (8)_{bin} = 1000$.
   Thus $f_{\text{INC},bin}(\text{M}, (7)_{bin}) = \text{Res}_{\text{M}}(\text{MMMM})$.
   Since $\text{Res}(\text{MMMM}) = \{(0)_{\text{un}}, \ldots, (15)_{\text{un}}\}$ *the function does not contain metastability — it amplifies it in this case.*
2. $f_{\text{INC,un}}((0)_{\text{un}}, (7)_{\text{un}}) = (7)_{\text{un}} = 011111111$ *and* $f_{\text{INC,un}}((1)_{\text{un}}, (7)_{\text{un}}) = (8)_{\text{un}} = 111111111$.
   Thus $f_{\text{INC,un}}(\text{M}, (7)_{\text{un}}) = \text{Res}_{\text{M}}(\text{M1111111})$.
   Since $\text{Res}(\text{M1111111}) = \{(7)_{\text{un}}, (8)_{\text{un}}\}$ *the function contains metastability in this case.*
3. $f_{\text{INC,rg}}((0)_{\text{un}}, (7)_{\text{rg}}) = (7)_{\text{rg}} = 00100$ *and* $f_{\text{INC,rg}}((1)_{\text{un}}, (7)_{\text{rg}}) = (8)_{\text{rg}} = 01100$.
   Thus $f_{\text{INC,rg}}(\text{M}, (7)_{\text{rg}}) = \text{Res}_{\text{M}}(\text{0M100})$.
   Since $\text{Res}(\text{0M100}) = \{(7)_{\text{rg}}, (8)_{\text{rg}}\}$ *the function contains metastability in this case.*

The example suggests that efficiency and metastability-containing are possible; in this example with binary reflected gray codes.

3.9 Applications

Techniques from the previous sections have been applied to obtain metastability-containing solutions for several realistic problems. We will discuss some of these in the following, starting with a metastability-containing TDC proposed we proposed in [29].

3.9.1 Time to Digital Converters

A time to digital converter (TDC) is a circuit that translates time (differences) of events to digital representations. Prominent applications are within logic analyzers, recording high-frequency digital signals, experimental setups in high energy physics, and high precision clock synchronization. While any stop-watch-like device with digital output is a TDC, we focus on high-precision TDCs built in integrated circuits. Motivated by their wide applicability, such TDCs have been extensively studied, see e.g. [59] for a recent overview.

Linear TDCs. The most widely adapted design style for high-precision TDCs is a linear design [60–63]. Figure 3.11 depicts such a design.

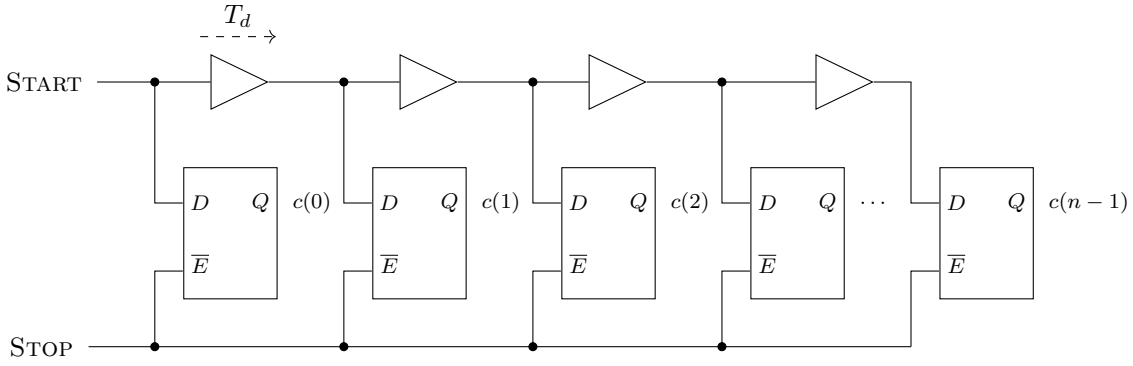


Figure 3.11: Tapped delay-line TDC. Latches are initially enabled and output 0. The delayed starting signal iteratively sets latches to 1 until the stopping signal disables them.

A sequence of $n \in \mathbb{N}^+$ evenly spaced delay elements, each with the same latency T_d, are arranged in a delay line. The delay line is then tapped after each delay element. A tap drives the input of a corresponding bistable storage element, e.g., a latch. With the rising START signal

transition, a logical 1 propagates along the delay line. At arrival of the rising STOP transition, all latches are disabled simultaneously, and their current state is frozen. The latch states $c(0)$ to $c(n-1)$ then form a digital, thermometer encoded, representation of the inter-arrival times of the starting and stopping transitions: flipped-over as $c(n-1), \ldots, c(0)$, the word 000001 represents a short time between starting and stopping, while 011111 represents a long time.

While we assumed (almost) simultaneous disabling of the latches, variants exist where the STOP signal is also delayed by a delay line. We denote the former as *regular* delay lines; the latter are referred to as *Vernier* delay lines.

The achieved time resolution of linear TDCs is roughly the latency T_d of a delay element, e.g., an inverter or a buffer. The latter is preferable to ensure identical (all rising) transitions at the delay element inputs; with the advantage of not having to balance latency for falling and rising transitions. Further, the latch states directly contain the thermometer encoded values without need for further inverting. In recent designs [60, 61, 63], the achieved time resolution is in the order of 10–80 ps, depending on the used process and delay element implementation. For comparison, with more advanced techniques like interpolation between delay elements, a bin size of 5 ps has been demonstrated, using a 130 nm process [61].

Increasing the dynamic range. In contrast to their favorable simplicity and the achieved time resolution, linear TDCs grow linearly in size with the dynamic range: measuring values within $\{0, \ldots, n-1\} \cdot T_d$ requires a linear TDC of length n. To keep circuit size low and to extend the dynamic range, fine-TDC/coarse-TDC approaches have been proposed. Typically the fine-TDC is a delay line, with a resolution in the order of T_d, and the coarse-TDC is based on a binary counter with a time resolution larger than T_d. Several variants of this approach exist and similar strategies have been invented many times independently. To the best of our knowledge, the earliest example of the fine-TDC/coarse-TDC approach is [64].

Metastability in TDCs. As Marino's topological argument [9] shows, any TDC necessarily faces the problem of metastable upsets: the continuous domain of time differences is mapped to the discrete domain of digital readouts.

For a linear TDC as in Figure 3.11, setting and disabling a latch at roughly the same time may occur and lead to a metastable upset of this latch. More precisely: Without loss of generality, assume that the rising transition of START occurs at time 0. Let $T > 0$ be a time duration, the *TDC's stabilization time*. Then, for latch $i \in \{0, \ldots, n-1\}$ with output $c(i)$, we define a countable union of disjoint closed intervals from $\mathbb{R}$, the *critical window* W_i, as follows.

Definition 74 (Critical window W_i, from [29])**.** *Assume that the rising* STOP *transition occurs at time $t \in \mathbb{R}$. If $t \notin W_i$, then the output $c(i)$ of latch i is stable* 0 *or* 1 *by time T. By contrast, if $t \in W_i$, then there is no guarantee that the output $c(i)$ of latch i is stable* 0 *or* 1 *by time T. In a worst-case manner we thus assume that its value is metastable* M*.*

Figure 3.12 shows the alignment of the latches' critical windows if they are non-overlapping for the linear TDC in Figure 3.11. Note that this can be achieved by increasing T_d; clearly though at the cost of increased time resolution. Figure 3.13 depicts a similar scenario, however with overlapping critical windows.

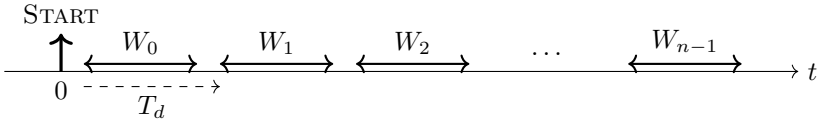


Figure 3.12: Critical windows for the linear TDC in Figure 3.11 with well separated windows.

In terms of guarantees on the readout, the design in Figure 3.12 only returns outputs of the form

$$c(n-1), \ldots, c(0) = 0^k(0 \mid \mathrm{M} \mid 1)1^{n-k-1} \ .$$

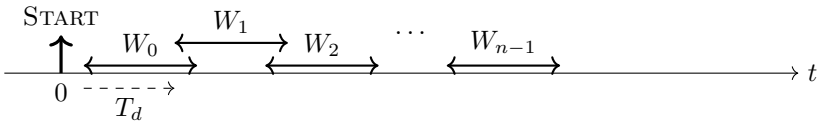


Figure 3.13: Critical windows for the linear TDC in Figure 3.11 with overlapping critical windows.

By contrast multiple metastable bits are possible in the design in Figure 3.13. In a worst-case model, decreasing T_d below the critical window size thus results in larger resolution times.

Give a metastable readout like 00M1111, one has several options to proceed:

- Resolve metastability in the TDC before carrying on with the computation. This is achieved by giving the TDC, and in our case, the latches, sufficiently large time to stabilize. In terms of critical windows, this means to increase T, such that the probability of t lying within a W_i is negligibly small. Clearly thus, this inhibits the reuse of the TDC in the meantime and thus decreases the sampling rate of the TDC.

- Copy the output to a memory and let it resolve there. Implementations, are, e.g., with synchronizer chains [65], or letting it stabilize in a separate memory after copying there. Again, the technique relies on increasing the time T until usage. This technique, however, allows for high sampling rates.

- Applying techniques from metastability-containing computation [66], and directly use the readout within a metastability-containing circuit. This includes stabilization during computation and no necessary stabilization — the latter, if the downstream circuit allows for this. Examples for applications that do not require stabilization [66–68] will be discussed in Section 3.9.

Simple metastability-containing TDC. Interestingly the linear TDC from Figure 3.11 can be directly used in metastability-containing circuits. To see this, note that a readout may contain at most a single M. Given that the values are thermometer encoded, the precision as defined in Definition 71 on page 50 is 1, which is optimal. The resulting time resolution thus is $1 \cdot T_d$.

The only drawback of this design is its large circuit size for larger dynamic ranges. Such designs would thus require a subsequent compact encoding, like BRGC.

Most designs for larger dynamic designs are counter-based. To deal with metastability, they protect the, typically binary, counter from metastable upsets by synchronizer chains. However, the resulting synchronizer delays of several clock cycles pose a deadtime for the TDC during which it cannot be used. An interesting exception for a counter-based design that follows a different approach to protect its counter(s) was presented by Mota *et al.* [69]. The authors use two coarse counters, one driven by the rising and one by the falling clock transition. The counters are aligned so that at most one of them may become metastable for a measurement. To determine which of them is definitely not metastable, the thermometer encoded values must be read, and thus must be guaranteed to not be metastable.

High dynamic range metastability-containing TDCs. A standard technique to increase the dynamic range of linear TDCs is to fold them up into a ring and attach a coarse counter at one tap. Examples of such designs, e.g., are [70] for regular folded delay lines and [71] for Vernier folded delay lines. Figure 3.14 on page 54 shows the design of a regular ring TDC with a generic coarse counter at one tap. The coarse counter has an increment input inc, an (low-active) enable input $\overline{E}$, and a multi-bit output C. While generally binary counters with synchronizers are used, we have proposed three alternative designs in [29] that can be used in metastability-containing circuits. Also note that while inc may be triggered by rising, falling or both transitions in general, we will use designs where both transitions lead to an increment.

The three designs differ only in how the counter in Figure 3.14 is implemented and on the output encoding. We briefly sketch the designs with performance parameters time resolution and dynamic range.

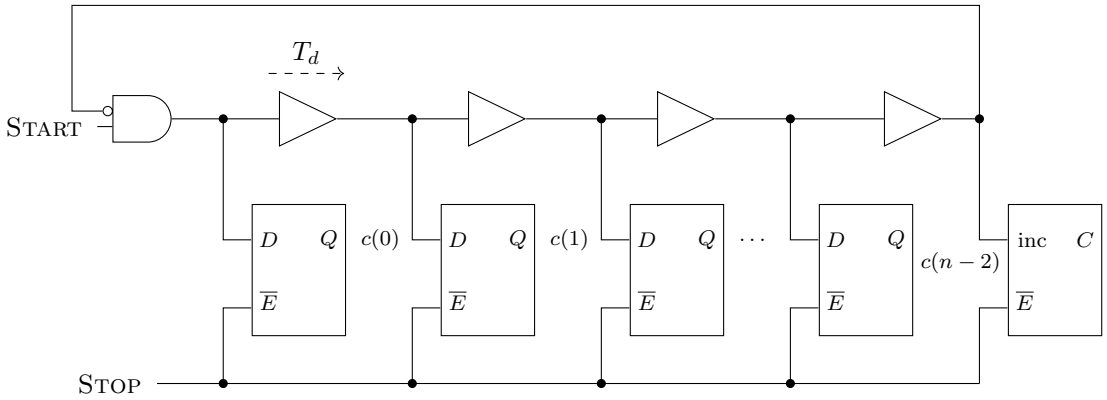


Figure 3.14: Generic ring TDC architecture with coarse counter output C. Latches and counter are initialized to 0. The counter increments on rising and falling transitions on INC if enabled.

1. **TDC-bin.** In this design, the counter comprises of two binary counters and a single bit indicating which counter value to use. This value is guaranteed to be correct and not metastable, while the other may be arbitrarily off.
   This design principle has been first described in [69]. We gave a correctness proof in [29]. It has the following properties:

   + Time resolution: T_d
   + Design: Any binary counter design may be used.
   − Memory: Two counter values need to be stored. If the binary counters have B bits, we store $2B + n$ bits for a measurement, which is slightly suboptimal.
   − Post-processing: Inefficient for currently available metastability-containing operations.

2. **TDC-gray.** In this design, a Gray code counter is used as a coarse counter. It has the following properties:

   + Time resolution: T_d
   − Design: Requires a custom counter and tighter timing constraints.
   + Memory: Optimal encoding. If the counter has B bits, we store $B + \lceil \log n \rceil$ bits for a measurement.
   + Post-processing: suitable for several existing metastability-containing operations

3. **TDC-graybit.** This design again uses a Gray code counter, and an additional single bit. The bit indicates whether an up-count should have taken place, shifting the tight timing constraints from the counter to a single latch.

   + Time resolution: T_d
   + Design: Requires a custom counter, but less tight timing constraints.
   + Memory: Only a single bit memory overhead of encoding. If the counter has B bits, we store $B + \lceil \log n \rceil + 1$ bits for a measurement.
   − Post-processing: Inefficient for existing metastability-containing operations

Before going into greater detail for the three designs, let us state some common properties for the ring TDC.

Similar to the linear TDC, we denote the output of latch $i \in \{0, \ldots, n-2\}$ in Figure 3.14 with $c(i)$. If not stated otherwise, we will use $c(i)$ also for the value of signal $c(i)$ at time $T > 0$ after the STOP signal transition. We then have:

Lemma 75 ([29]). *Assume that latches have zero switching time and thus empty critical windows. Also assume that the coarse counter C is a single-bit, binary latch and denote its output by $c(n-1)$. Denote by* cnt *the sum of the total number of latch output transitions of all stages between the starting and stopping signals, i.e., the total count of the TDC. Then one of the two cases applies:*

- *The ring oscillator made a number of full rounds and all latch outputs are of the form $0\ldots0$ or $1\ldots1$. Then,*

$$\begin{aligned} &\mathrm{cnt} \bmod n = 0 \quad \textit{and} \\ &\forall i, j \in \{0, \ldots, n-1\} \,:\, c(i) = c(j) \ . \end{aligned}$$

- *The ring's state was captured in between two full rounds and the latch outputs are of the form $0\ldots01\ldots1$ or $1\ldots10\ldots0$. Then,*

$$\begin{aligned} &c(\mathrm{cnt} \bmod n) \neq c(\mathrm{cnt} + 1 \bmod n) \quad \textit{and} \\ &\forall i \in \{0, \ldots, n-1\} \setminus \{\mathrm{cnt} \bmod n\} \,:\, c(i) = c(i+1) \ . \end{aligned}$$

This allows one to determine $\mathrm{cnt} \bmod n$ from the latch outputs $c(i)$. A more refined analysis with taking into account the fact that a single bit may become metastable, finally yields:

Corollary 76 ([29]). *Assume that the critical windows of the latches are non-overlapping. Then, after potential metastability resolved,* $\mathrm{cnt} \bmod n$ *can be determined according to Lemma 75 with precision* 1.

TDC-bin. In this design the counter is implementation by two binary counters [69]. Figure 3.15 depicts the circuit.

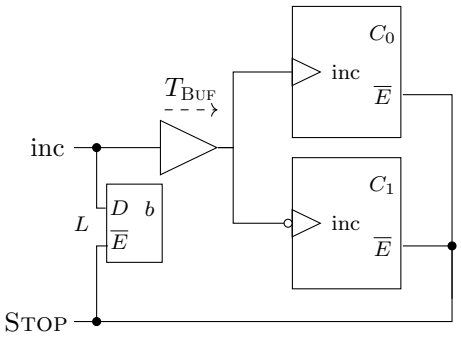


Figure 3.15: TDC-bin implementation of the coarse counter. Binary counters C_0 and C_1 as well as latch L are initialized to 0. The increment inputs of C_0 and C_1 are driven via a buffer with delay T_{BUF}. Counter C_0 increments on rising transitions and counter C_1 on falling transitions.

Both counters and the latch are initialized to 0. Output $b \in \mathbb{B}$ of latch L serves as a select bit for the two counters: counter C_b is used. The circuit, and the following delay constraints, then ensure that C_b is not metastable.

In particular, we require: Denote by W_{C_0} and W_{C_1} the counters' critical windows. Denote by W_L the critical window of latch L. For a window W, denote by W^i, with $i \in \mathbb{N}^+$, the i^{th} interval of W, ordered by time, if it exists. Then we require,

$$W_L^1 < W_{C_0}^1 < W_L^2 < W_{C_1}^1 < W_L^3 < W_{C_0}^2 < \ldots \ .$$

The resulting alignment is shown in Figure 3.16 on page 56. We achieve this alignment by the following design choices: For simplicity assume that $|W_{C_0}| = |W_{C_1}|$. Then

- $W_L^i < W_{C_b}^i$ is ensured by making the delay latency T_{BUF} sufficiently large.
- $W_{C_b}^i < W_L^{t+1}$ is ensured by choosing the ring size n large enough such that counter C_b completes an increment before the transition traverses the ring.

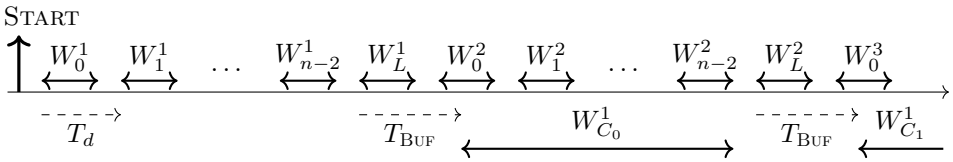


Figure 3.16: Critical windows for TDC-bin.

Given that these constraints hold, we obtain:

Lemma 77 ([29]). *After possible metastability of b has been resolved, it holds that* $\text{cyc} = 2C_b + b$.

The proof relies on observing that a stable b implies a stable C_b, and a metastable b implies that both C_0 and C_1 are stable. It is then shown that all these counts are set correctly to obtain the lemma's statement.

Lemma 77 hints at an interesting interpretation of the counter values and b. Since C_b is a binary counter, the binary representation of $2C_b + b$ is easily obtained by concatenation of C_b and b.

We finally obtain for the readout of TDC-bin:

Theorem 78 ([29]). *After metastability of the latches has been resolved,*

$$\text{cnt} = (2C_b + b)n + x_{1-b} \ ,$$

where x_{1-b} is the number of latches having value $1 - b$. Moreover, it is exactly the leading x_{1-b} latches that have value $1 - b$.

Memory-wise, if the binary counters have B bit, a measurement requires $2B + n$ bit. This is below optimal: The maximum cnt value $(2^{B+1} - 1)n + (n - 1)$ of such a measurement would only require $B + 1 + \lceil \log n \rceil$ bit in an optimal encoding.

TDC-gray. The design TDC-gray, by contrast, uses an optimal $B + \lceil \log n \rceil$ bit per measurement. Its coarse counter is shown in Figure 3.17. At its core there is a single Gray code counter (shown in blue in the figure), and a successive register of the counter's bit width. While in general, the design works with arbitrary Gray codes, we have shown that BRGC counters are particularly favorable—we discuss this later on in the section.

When a rising transition arrives at the inc input, it is first translated into a pulse by the XOR and the buffer. The buffer delay T_{BUF} is chosen sufficiently large such that the rising transition then triggers the latching of the counter output by the register. Only then, and accounting for the register's hold time, the counter is triggered to increment. The resulting critical windows and their alignment are shown in Figure 3.18 on page 57. While the counter increment may take almost a full round-trip time of the TDC cycle, the register's critical window intervals must be roughly of the same size as the critical window interval sizes of the latches along the ring; see Figure 3.18.

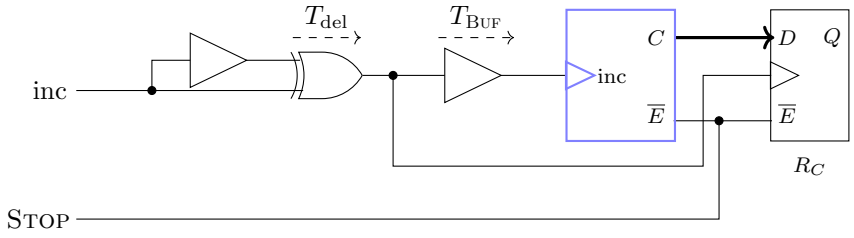


Figure 3.17: TDC-gray design. The counter C (shown in blue) is initialized to 1. Register R_C is initialized to 0. Both are triggered by rising transitions.

While the Gray code from the coarse counter is already optimally compressed, the thermometer encoded latch states require further compression. We proposed to compress them into a Gray code, too.

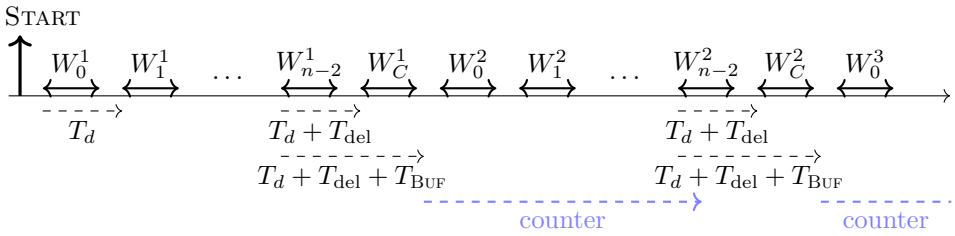


Figure 3.18: Critical windows for TDC-gray. Note the period during which the counter C increments (in blue) and the subsequent critical window intervals W_C^1 and W_C^2 of the register R_C that latches the result.

Lemma 79 ([29])**.** *Consider an arbitrary* $\lceil \log n \rceil$*-bit Gray code. Suppose that we are given an* n*-bit ring latch state of the form* 11..1M00..0 *or* 11..100..0*. If a possible* M *is resolved to either* 0 *or* 1*, the resulting string is a thermometer encoding of either* r *or* $r+1$ *for some* $r \in [n]$*; if there is no* M*, the string is a thermometer encoding of* $r \in [n+1]$*; i.e., it has precision of* 1*.*

There is a purely combinational circuit of at most n XOR *gates and depth at most* $\lceil \log n \rceil$ *that transforms all such inputs into the Gray code, in the sense that in case of metastability in the input, (i) at most one output bit is metastable, and (ii) resolving it to either* 0 *or* 1*, the encoded number becomes either* r *or* $r+1$*; i.e., the circuit preserves precision of* 1*.*

Since the circuit maps inputs with precision 1 to outputs with precision 1, we may say that the function that it implements is metastability-containing. Note that technically, our definition of a metastability-containing function does not apply, since input and output encoding are not identical — however, a generalized version does.

The key observation in the proof of Lemma 79 is that a counter increment, triggered by a latch being set to 1, can be viewed as negating a subset of the output bits (implemented by XOR gates). Clearly, though, while in the first round 1s have to be counted, in the second round 0s have to be counted, *a priori* requiring two such encoding circuits. Luckily, in the case of BRGC the same circuit may be used. For that purpose we refer the reader to Table 3.1: in fact counting leading 1s is equivalent to counting leading 0s and negating all inputs. In BRGC it turns out that this results in a negation of the left-most BRGC bit; see Table 3.1.

Table 3.1: Encoding the ring latch states from thermometer encoded values to BRGC encoded values. The example shows the case for a ring of size $n = 8$ with $n - 1$ latches and the coarse counter.

ring latch states (thermometer encoded)							BRGC encoded			counts	
0	1	2	3	4	5	6	0	1	2	#1s	#0s
0	0	0	0	0	0	0	0	0	0	0	7
1	0	0	0	0	0	0	0	0	1	1	
1	1	0	0	0	0	0	**0**	1	1	2	
1	1	1	0	0	0	0	0	1	0	3	
1	1	1	1	0	0	0	1	1	0	4	
1	1	1	1	1	0	0	1	1	1	5	
1	1	1	1	1	1	0	1	0	1	6	
1	1	1	1	1	1	1	1	0	0	7	0
0	1	1	1	1	1	1	1	0	1		1
0	0	1	1	1	1	1	**1**	1	1		2
0	0	0	1	1	1	1	1	1	0		3
0	0	0	0	1	1	1	0	1	0		4
0	0	0	0	0	1	1	0	1	1		5
0	0	0	0	0	0	1	0	0	1		6

The two Gray code encoded values are optimal in terms of memory efficiency and maintain

precision of 1 in presence of metastable inputs. We thus obtain:

Corollary 80 ([29])**.** *Using a B-bit Gray code coarse counter and the XOR-tree circuit from Lemma 79, a measurement can be stored as a tuple of two Gray codes, using* $B + \lceil \log n \rceil$ *bits, without losing precision.*

In fact, and again making use of the peculiarities of BRGC encoded values, we may easily combine the two Gray code values into a single BRGC encoded values that encodes the TDC value:

Theorem 81 ([29])**.** *If we use a BRGC coarse counter and* n *is a power of* 2*, then* just *the concatenation of (a) the output of the above* Xor*-tree circuit,* without the need to negate *any of its output bits, and (b) the output of the BRGC counter yields a BRGC encoding of the TDC measurement value.*

Figures 3.19, 3.20, and 3.21 demonstrate Theorem 81. They show three cases for even, odd, and metastable coarse counters. Observe that in all cases, the final BRGC encoded values correctly encode the TDC measurement value with the same precision as the inputs (precision 1).

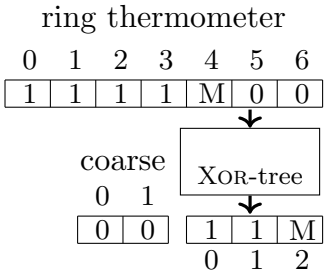


Figure 3.19: Circuit encoding a TDC measurement value ($n - 1$ thermometer encoded ring latch states and B bit coarse BRGC counter state) as a single BRGC encoded value without losing precision in presence of a metastable input bit. The example shows the case $n = 8$ and $B = 2$. The coarse counter stores 0 (even), thus 1s have to be counted in the ring. The final BRGC value 0011M correctly encodes decimal 4 or 5, depending on how M resolves.

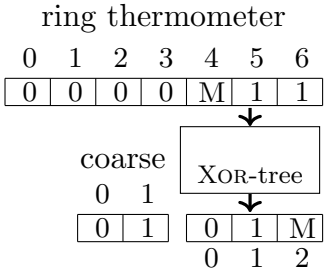


Figure 3.20: The coarse counter stores decimal 1 (odd), thus 0s have to be counted in the ring. The final BRGC value 0101M correctly encodes decimal 12 or 13, depending on how M resolves.

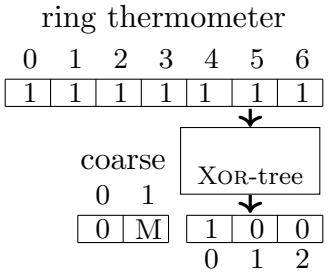


Figure 3.21: The coarse counter is metastable. The final BRGC value 0M100 correctly encodes decimal 7 or 8, depending on how M resolves.

TDC-graybit. The third coarse counter design is shown in Figure 3.22 and its critical windows in Figure 3.23. The design comprises of a Gray counter C and a latch L. The purpose of the latch is to "shrink" the critical window that the Gray counter may have. While the counter may get metastable in one bit during a large window W_C, the state of the latch with the short critical window can be used to determine whether the counter should be even or odd — and correct the counter after stabilization accordingly.

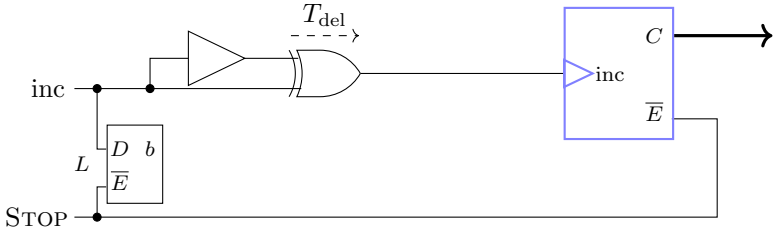


Figure 3.22: TDC-graybit design. The counter C (shown in blue) and the flip-flop L are initialized to 0.

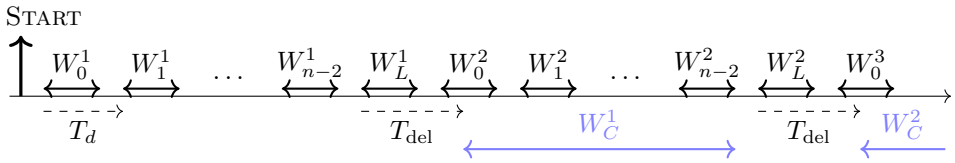


Figure 3.23: Critical windows for TDC-graybit.

3.9.2 Minimum, Maximum, Median, and Sorting

TDCs are widely used modules at the interface to an outer environment. We will next discuss an application of metastability-containing techniques related to computation.

Figure 3.24 on page 60 shows a circuit that computes the minimum of two thermometer encoded values. The circuit is efficiently realized by bit-wise And of the inputs. An analogous circuit with bit-wise Or computes the maximum of the two inputs.

Observer that the minimum and the maximum of thermometer encoded values, with the output again being thermometer encoded, is a metastability-containing function. For example, the minimum of an input with precision 0, say $x = 0001111$, and and input with precision 1, say $y = 0000\text{M}11$, yields $\min(x, y) = 0000\text{M}11$; which has precision 1. Further, the circuit in Figure 3.24 is metastability-containing as has been observed in [28].

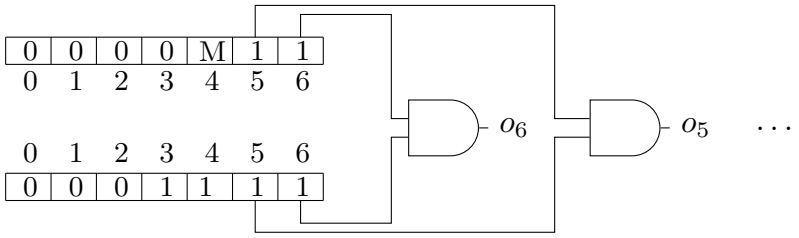


Figure 3.24: Computing the minimum of two thermometer encoded values x and y. The circuit is metastability-containing: an input with precision 1 results in an output with precision 1.

With metastability-containing versions of minimum and maximum, computing the median, mid-value, and sorting in general, are readily computed in a metastability-containing way: minimum and maximum are combined to a 2-sort element, which is then assembled according to techniques from sorting networks [72, 73].

The 2-sort with the minimum circuit shown in Figure 3.24 requires inputs and outputs to be thermometer encoded. However, more efficient encodings like BRGC are preferable for larger values. Indeed, first observe that the minimum and maximum functions are metastability-containing for values that are BRGC encoded as well: precision 1 at the inputs results in precision 1 at the outputs. Second, Corollary 61 on page 47 shows that there also exists a metastability-containing circuit that implements BRGC-encoded maximum and minimum. A priori, such a circuit may be large (exponential in the input size). Efficient implementations have been shown to exist [74–76], however.

3.9.3 Controllable Oscillator

We will next discuss a component that has been used in several of our designs that use metastability-containing circuitry such as the link controller in [32]. In contrast to the previously discussed components, the purpose of this component is not to compute a function, but rather to produce an oscillating signal with a controllable frequency. We will also explain what we mean with metastability-containing in this context; although we do not formalize the term for such components.

A controllable oscillator, in our context, is a component with a temperature encoded input i and a single output port o that will be the oscillating output. In its simplest form, the input is a single bit i. For simplicity, we assume this in the following, although the concept is easily generalized to multiple bits.

For ease of analysis, we will sometimes refer to two other underlying output representations that induce the binary output signal: c and C. We denote by $C(t) \in \mathbb{N}$ the discrete clock count/value at Newtonian time $t \in \mathbb{R}$; that is, the number of active transitions made by output o until time t. Further, let $c(t) \in \mathbb{R}$ be the output's underlying continuous (normalized) phase signal at time t. For simplicity of analysis we assume normalization of the phase by 2π, i.e., the phase of a full clock period is 1 and not 2π. Thus $C(t) = \lfloor c(t) \rfloor$. The instantaneous frequency of c is dc/dt.

Let the *frequency bounds* $s^- \leq s^+ \leq f^- \leq f^+$ be from $\mathbb{R}_0^+$ and the *latency* T_{osc} be from $\mathbb{R}_0^+$. A *controllable oscillator* then fulfills:

(C1) The output signal o always has instantaneous frequencies within $[s^-, f^+]$.

(C2) If $i = 0$ during time $[t - T_{\text{osc}}, t]$ then the instantaneous output frequency of c at time t is within the slow frequency bounds $[s^-, s^+]$. We say the oscillator is in *slow mode* at time t.

(C3) If $i = 1$ during time $[t - T_{\text{osc}}, t]$ then the instantaneous output frequency of c at time t is within the fast frequency bounds $[f^-, f^+]$. We say the oscillator is in *fast mode* at time t.

(C4) If i is neither constant 0 nor constant 1 during time $[t - T_{\text{osc}}, t]$, we say that the oscillator is *unlocked* at time t.

Note that clocks in slow mode are never faster than clocks in fast mode.

While in absence of metastable inputs, in our case this means ruling out $i = \mathrm{M}$, many designs of controllable oscillators have been proposed, care has to be taken for designs that behave according to the above specification in presence of metastability. Most designs, however violate (C1) in presence of a metastable input bit. We thus call the oscillator metastability-containing if it grantees constraint (C1).

In [32], we proposed to use starved-inverter ring oscillators that guarantee such behavior [77]. A starved inverter is an inverter whose propagation delay can be controlled by starving its power supply. This can be done via an input bit that, if set, reduces power supply, and if not set, does not do so or to a less extent. Aligning such inverters along a ring results in a ring oscillator whose period is controllable via the starved inverter propagation delays. The transistor level implementation used in [32] followed the design presented in [78], however without the full control logic overhead typically required to drive the starved inverter cells, since our input values are already temperature encoded and can be directly used to control the inverters. Proper decoupling by intermediate buffers is used in order to minimize unwanted effects on the oscillator frequency by the control logic and by the oscillator output's downstream logic. Not all inverters in the ring need to be starved and a single input bit can control one or several starved inverters. This is useful to set the sensitivity of the oscillator to input changes.

Most importantly, why is this design metastability-containing? Observe that (continuous-valued) propagation delays of the inverters are controlled via the (continuous-valued) reduced V_{dd} with the oscillation frequency extremes being s^- and f^+. Setting the input to M will lead to a V_{dd} between those for logical 0 and 1, and thus will result in an inverter propagation delay between those for inputs 0 and 1. The resulting oscillation frequency thus remains within s^- and f^+. In particular no glitches will be produced by the oscillator.

3.9.4 Byzantine Fault-tolerant Clock Synchronization

In this section, we discuss a more high-level application of metastability-containment techniques that was presented in [28]. The problem of *distributed clock synchronization* is to make a distributed system of $n \in \mathbb{N}^+$ processes, generate clock ticks simultaneously [79, 80]. Interpreting the clock generation times on a global Newtonian timeline as local output values, the problem is closely related to the problem of repeatedly solving approximate consensus [81].

Both problems are well studied in distributed computing [80], with variants differing in assumptions on which nodes can communicate, stability of the underlying network, and process faults.

Faults. Focusing on algorithms implemented in hardware, a first question is for a valid model of faults in such systems.

There is a large body of literature in the circuit community on types of faults and mitigation strategies at different levels of abstractions. For an introduction with a focus on faults in dependable systems, we refer the reader to Kopetz [82]. An overview on trends is presented by Constantinescu [83].

Radiation induced *single event upsets* (SEUs), i.e., bit-flips of storage elements, have received particular attention in circuits intended for critical missions. Radiation hardened synchronous [84] and asynchronous [85] designs have been proposed. Radiation induced *single event transients* (SETs), i.e., glitches within the combinational logic, have gained in importance though [86–88] and may result in downstream bit-flips, delay faults, and glitches. Potential results of such faults are inconsistent reception of signals and even metastable upsets. Simple *crash models* are thus not sufficient for low-level circuits if high coverage is the goal. *Byzantine faults*, by contrast, capture inconsistent message receptions. We have argued in this chapter that Byzantine faults, however, are still not sufficient to cover metastable upsets with non-binary values.

Model. Let us consider the classical Byzantine fault-tolerant message passing model [80], here: The communication system by which the n processes communicate is fully-connected and stable, and up to $f < n/3$ nodes may fail in an arbitrary, i.e., Byzantine, way. Messages latencies between correct nodes are within some $[\tau^-, \tau^+]$. Every node has access to a local (continuous) clock with

rate $[1, 1+\varrho]$, where $\varrho > 1$. For simplicity we assume that all correct nodes start simultaneously at time 0, although we note that this condition can be relaxed.

Denote by $t_i(k)$, with $i \in [n]$ and $k \in \mathbb{N}^+$, the time node i makes its k-th tick. Further let $C_i(t)$ be the number of ticks made by node i until including time t. An algorithm solves distributed clock synchronization, if there exist *accuracy bounds* $\alpha^-, \alpha^+ > 0$ and $d > 0$, as well as a *skew* $\sigma > 0$, such that in all executions:

- For all correct nodes $i \in [n]$, and all $t', t \in \mathbb{R}_0^+$ with $t' > t$,

$$C_i(t') - C_i(t) \in [(t'-t)\alpha^- - d, (t'-t)\alpha^+ + d] \ . \tag{3.15}$$

- For all $k \in \mathbb{N}^+$, and all correct nodes $i, j \in [n]$,

$$|t_i(k) - t_j(k)| \leq \sigma \ . \tag{3.16}$$

The Lynch-Welch distributed clock synchronization algorithm [89] was shown to solve distributed clock synchronization. Without going into details, the algorithm repeatedly lets a node (i) make a tick and broadcast it to all other nodes, (ii) estimate its offset to the other clocks based on the received messages and calculate a correction term from it, and (iii) adapt the time when to produce the next tick based on the correction term. Care has to be taken in that ticks of correct nodes are received timely such that the correction factor can be computed before the next tick has to be made. This results in conditions on (lower and upper bounds of) message delays and the drift ϱ. Central to the algorithm is how the correction factor is computed from the received clock offset estimates: the f largest and smallest values are discarded (setting estimates corresponding to non-received values to ∞), and the *mid-point* of the interval I of the remaining clock offsets is computed as

$$\frac{\min(I) + \max(I)}{2} \ . \tag{3.17}$$

Variants of this algorithm have been implemented in the Time-Triggered Protocol (TTP) [90] and Flexray [91] for clock synchronization in fault-tolerant systems. Both are software–hardware based implementations and achieve a skew in the order of microseconds. To obtain higher operating frequencies and a better skew, pure low-level hardware implementations are inevitable because of jitter. Kinali *et al.* [92] implemented the Lynch-Welch algorithm on a set of FPGA boards All known implementations, however, synchronize potentially metastable inputs before computations.

Inherently, such implementations face the problem of metastable upsets: If the estimated clock offset of node i with respect to node j is digital, there exists an offset between the two clocks that results in a metastable offset measurement. While the above implementations [90–92] solve this problem by making the probability of an upset arbitrarily small by synchronizing i's signal into j's clock domain, a solution with synchronizers increases the control loop's latency, i.e., the algorithm's synchronization period. Latency is important for high-precision synchronization, however, since clocks drift apart with rates up to ϱ during the controller latency. Further, despite increasing synchronizer stages reduces failure rates — at the cost of additional latency — a non-zero probability of failing remains. This is particularly important for the following reasons:

- Clock synchronization algorithms that strive for high precision need to operate at high synchronization frequencies. Metastable upsets become (roughly) proportionally more likely with increased frequencies.

- The Lynch-Welch algorithm requires a fully connected communication network. Metastable upset probabilities thus scale quadratically with the number of nodes.

- The point of the clock synchronization algorithm is to produce clock transitions close in time. Standard formulas that assume uniformly distributed data arrival times within the clock period thus do not apply. Rather the algorithm potentially acts like circuits that deliberately drive a circuit into metastability — see the literature on deep metastability measurements [37, 93, 94].

- As noted before, the standard formula for metastable upset rates assumes uniform arrival times. This also is not a valid model for Byzantine nodes that drive input signals. The signals may not only be deliberately aiming to produce metastable faults by controlled transitions or high frequency oscillations, but may also be unclean signal transitions. In particular, any intermediate voltage may be applied by the Byzantine node.

For the last two point, an upper bound on the probability of faults induced by metastable upsets is impossible without further restrictions made on the Byzantine nodes.

In [28] we showed that metastability-containing circuits, and in particular the TDC and sorting discussed previously, can be used to design the first distributed clock synchronization algorithm that indeed tolerates Byzantine faults in a model that also captures metastability. The algorithm is a metastability-tolerant implementation of the Lynch-Welch algorithm. Figure 3.25 shows its main modules ad how they are interfaced.

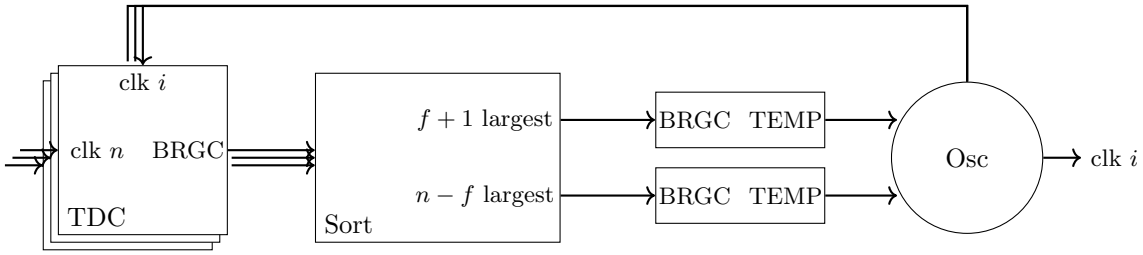


Figure 3.25: Metastability-tolerant implementation of the Lynch-Welch algorithm to tolerate Byzantine faults in a model that accounts for metastability. The circuitry for node $i \in [n]$ is shown. The time-offset of the received clock transitions is measured in a metastability-containing TDC with BRGC encoded outputs. The values are sorted, and the $f+1$ and $n-f$ largest re-encoded to temperature code. The two values are then used to change node i's oscillator frequency.

The phase offset is first measured by TDCs, one for each of the $n-1$ neighboring nodes. Offsets are assumed to be BRGC encoded; see Section 3.9.1 for such designs. Sorting networks, respectively parts of it, are used to select the $(f+1)$-th and $(n-f)$-th largest offset value; see Section 3.9.2 for solutions. The two resulting BRGC outputs are re-encoded to temperature encoded values. While we did not provide circuits for this, the function is readily seen to be metastability-containing, and the existence of metastability-containing versions of such circuits follows from Theorem 58 on page 47. It remains to show the existence of an oscillator whose frequency can be controlled depending on two temperature-encoded values, that potentially contain metastability. A design analogous to the oscillator presented in Section 3.9.3 can be used for this purpose. Note that we do not need to calculate the mid-point in thermometer encoded values, since the controllable oscillator does not depend on correct thermometer representations, but rather on the number of bits set to 1. Assuming we are in the linear control region, we may thus simply concatenate $\min(I)$ and $\max(I)$ and divide the linear slope by 2.

One observes that the proposed circuit implements the Lynch-Welch algorithm and preserves precision from the inputs to the outputs. We thus obtain:

Theorem 82 ([28])**.** *In the circuit model that includes metastable upsets, the circuit in Figure 3.25 solves the distributed clock synchronization problem in a fully connected network of $n > 1$ nodes, in presence of up to $f < n/3$ Byzantine faulty nodes.*

3.9.5 Link Controller

In [32] we used techniques from metastability-tolerant computing in a link controller between a sender and a receiver with their respective clock domains. Communication across clock domains, e.g., occurs in Globally Synchronous Locally Asynchronous (GALS) system [95, 96]. From Marino's work [9], we know that under realistic conditions and uncorrelated clock domains, such a

communication is inherently prone to metastable upsets. Different mitigation strategies in link controllers have been proposed to handle this problem:

- Gating the clock [97] with signals that measure fill levels to prevent overflow and underrun of an asynchronous FIFO with synchronous input and output. This technique requires almost-full flags that have to be protected against metastability; making upset probabilities sufficiently small but with a remaining non-zero probability of metastable upsets.
- Clock-pausing [98, 99], at the cost of unbounded start-up times in presence of metastability.
- Synchronizers [100–103] at the cost of increased latency and non-zero probability of metastable upsets.

  The design in [100] is based on a mixed-clock first-in first-out pipeline (FIFO) with flow control logic. Synchronized full/empty and almost full/empty signals are used as handshaking signals. Its throughput is one data item per clock cycle until the almost full signal is raised; afterwards, the "true" full signal has to be considered, at the cost of increased latency and lower throughput.

  In [102], a locally delayed latching approach is proposed: conflicting read/write operations are delayed by an asynchronous controller with a MUTEX element. The controller's latency is in the order of 20 gate delays. The minimum feasible clock cycle is at least 69 gate delays.

  Gradual synchronization [103] allows fine-grained interweaving of synchronization and computation, also shifting conflicting ripple FIFO requests by MUTEX elements at each stage.

  A notable exception in how synchronizers are used is the work by Dally and Tell [101]. They propose a design where upset probabilities can be made arbitrarily small without increasing latency: synchronizers continually determine sender–receiver phase offsets. The offsets can then be used to skip sender/receiver cycles when upsets could occur. A drawback is that the frequency and phase measurement circuits require accurate phase tracking and can account for slow phase drifts only.
- Synchronizing sender/receiver clocks, i.e., controlling their phase offset. The benefit of this approach is that, once synchronized, communication can be done in a lightweight provably metastability-free way. The approach from [32], presented in this section, also falls into this class. The closest work to our approach presumably is Polzer *et al.* [104]. By using the distributed DARTS clock generation mechanism [12], a buffer size of 9 and latency of 9 clock cycles was achieved for a receiver-sender clock shift of 4 ticks at around 25 MHz in an FPGA. While these numbers clearly can be improved in ASIC designs, DARTS inherently is slower than our approach.

Metastability-containing controller. In [32] we presented an alternative design for a link controller. The design is for a unidirectional link from a synchronous sender node SND to a synchronous receiver node RCV. For simplicity we assume that the communication is unidirectional, although a generalization to bidirectional communication is possible. Sender and receiver have two distinct controllable oscillators as clock sources. The controller makes sure that both oscillators are such that the link's buffer never underruns, overflows, or accesses the same data item at the same time. Figure 3.26 shows the setup. It comprises of three parts: (i) the tunable oscillators OSC_{snd} and OSC_{rcv}, (ii) the (ring) buffer BUFF, and (iii) the digital link controller CTRL.

The link operates as follows: At every active transition of the sender clock clk_{snd}, the sender writes data to a ring buffer of even size $N \in 2\mathbb{N}^+$. On every active transition of the receiver clock clk_{rcv}, the receiver reads data from the ring buffer. Buffer cells are numbered from 0 to $N-1$. We assume that the buffer is addressed by two pointers — the sender and the receiver pointer, both of which are incremented modulo N every respective clock tick.

Controller model. Assume a sender and a receiver controllable oscillator that follows the specification given in Section 3.9.3. Call their single-bit inputs md_s and md_r, respectively. We denote their phase at time $t \in \mathbb{R}_0^+$ with $c_s(t)$ and $c_r(t)$, respectively. Additionally assume that the two oscillators are started roughly at the same time, i.e., within some $\delta \geq 0$ maximum initial phase offset:

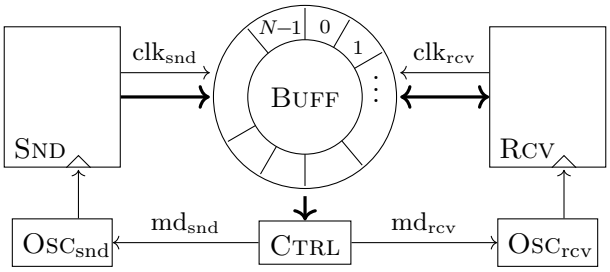


Figure 3.26: Link with digital link controller CTRL, from [32]

(P1) Initially $c_s(0), c_r(0) \in (-\delta, 0]$.

Further, assume the setup shown in Figure 3.26 with N cells. We define cell access and validity of a cell as in [32]:

(A1) The receiver's continuous address pointer is $p_r(t) = c_r(t)$.

(A2) The sender's continuous address pointer is $p_s(t) = c_s(t) + N/2$.

(A3) The receiver's discrete address pointer is $P_r(t) = \lfloor p_r(t) \rfloor \mod N = C_r(t) \mod N$. We say the receiver *(starts to) access cell* $\ell \in \{0, \ldots, N-1\}$ *at time* t if $P_r(t) = \ell$.

(A4) The sender's discrete address pointer is $P_s(t) = \lfloor p_s(t) \rfloor \mod N = C_s(t) + N/2 \mod N$. We say the sender *(starts to) access cell* $\ell \in \{0, \ldots, N-1\}$ *at time* t if $P_s(t) = \ell$.

(A5) To account for non-zero access of cells, we assume that an access has a duration τ_s for the sender and τ_r for the receiver.

(A6) On initialization, cells $\ell \in \{0, \ldots, N/2 - 1\}$ are *valid*, and all other cells are *invalid*.

(A7) If the sender accesses an invalid cell at time t, the cell becomes valid at time $t + \tau_s$ until it is invalidated by the receiver. If the reader accesses a valid cell at time t, it becomes invalid at time $t + \tau_r$ until it becomes valid by the sender.

Linked to each cell ℓ, there is a full/empty flag F_ℓ. Its value at time t is:

(F1) 1 if cell ℓ is valid at time t,

(F2) 0 if cell ℓ is invalid at time t,

(F3) arbitrary within $\mathbb{B}_\mathrm{M}$, otherwise.

The specification of the controller–oscillator interface is as follows. With $T_\mathrm{ctr} > 0$ being an upper bound on the controller latency:

(L1) If the controller output sets md_s to $b \in \mathbb{B}$ during $[t - T_\mathrm{ctr}, t]$, then $\mathrm{md}_s(t) = b$. An analogous statement holds for md_s.

(L2) In all other cases md_s and md_r are arbitrary within $\mathbb{B}_\mathrm{M}$.

Problem. We are now in the position to state the problem of a correct link controller in our sender–receiver setting.

Definition 83 (from [32])**.** *A link is correct if the following holds for all executions:*

(P1) No underrun: the receiver accesses only valid cells.

(P2) No overflow: the sender accesses only invalid cells.

Definition 84 (from [32])**.** *Controller* CTRL *is correct if it computes the signals* md_s *and* MD_r *out of the inputs* $(F_\ell)_{\ell \in \{0,\ldots,N-1\}}$ *so that the link is correct.*

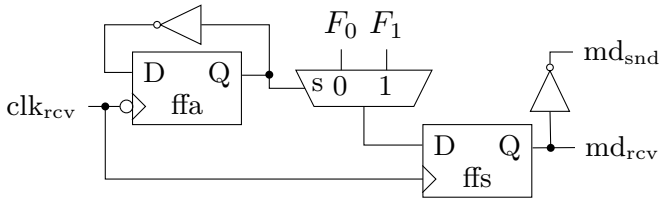


Figure 3.27: Gate-level circuit of the synchronous controller ClockedTh. The controller is for a ring with $N = 2$ cells. Flip-flop ffa stores the address (modulo 2) that is sampled and ffs the sampled full/empty flag. From [32].

Solution. Figure 3.27 depicts the controller ClockedTh, a clocked gate-level circuit that has been proven to be a correct controller in [32].

Besides correctness, we have shown bounds on two performance measures: The link's *latency*, i.e., the time between consecutive accesses of the sender and receiver to the same cell plus the setup/hold time at the receiver (as the data should be stable before it is used). The link's *throughput*, i.e., its rate of delivered packets. We obtained:

Theorem 85 (from [32])**.** *For*

$$\Delta = \lceil (f^+ - s^-)(T_{osc} + 1/s^- + \tau_{\max}) + f^+ \max\{\tau_s, \tau_r\} + \max\{\delta, f^+\tau_s/2\} \rceil \text{ and}$$
$$N \geq 2\Delta$$

the controller ClockedTh is correct and the link has latency N/s^- and throughput $1/s^-$.

The proof is by showing that a non-implementable, ideal, continuous controller is correct and then showing that the controller ClockedTh implements the ideal controller if parameters are chosen properly.

The link controller with the successive oscillators bears some similarities to phase locked loop (PLL) designs [105, 106]: its purpose is to control a frequency with respect to a phase difference. The controller can be viewed as an all-digital phase detector. However, the biggest difference is that our design is a distributed setup with neither the receiver nor the sender dictating the frequency of the other.

Performance and comparison. Table 3.2 shows a comparison of our controller synthesized for the UMC 65 nm library. We compare the link's performance to the most closely related works, Polzer *et al.* [104] and Dally and Tell [101].

Table 3.2: Performance and hardware overhead (buffer size N, gates, flip-flops, oscillator type) of the proposed controller with a tunable 2.0 to 2.3 GHz oscillator, [101], and [104]. From [32].

		ClockedTh	[104]	[101]
Performance	Latency [ns]	1	375	1.3
	Throughput $[\frac{\mathrm{pkt}}{\mathrm{ns}}]$	2	$\frac{1}{41}$	$\frac{1}{1.3}$
	MTBF	∞	∞	Finite
Overhead	N	2	9	2
	# Gates	8	> 100	> 100
	# flip-flops	4	> 50	> 100
	Oscillator type	tune	distributed	quartz

The controller operates within 2–2.3 GHz. VHDL and Spice simulations were in accordance with predictions from theory, and no underrun or overflow was detected in the simulations.

Figure 3.28 shows a subtrace of the VHDL simulations with potentially metastable signals in red color.

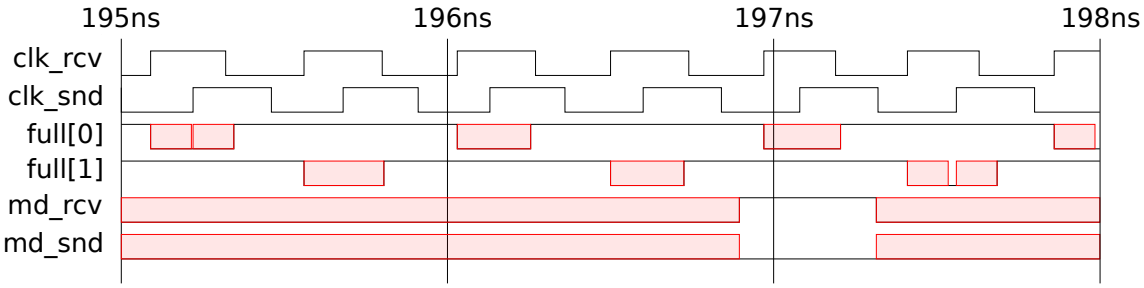


Figure 3.28: Gate-level simulations for link with ClockedTh. From [32].

The VHDL code for the behavioral simulation of a ring buffer of $N = 2$ elements is shown below. Observe the explicit generation of X for the FULL-flags whenever a cell is written and read. The code outputs X in a worst-case manner for τ time, but can be adjusted to generate X for a (uniformly) random time $d \in [0, \tau]$ as indicated in the code.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.math_real.all;

entity Ring is
  port(
    clk_snd: in std_logic;
    clk_rcv: in std_logic;
    full: out std_logic_vector(0 to 1) := "10"
    );
end Ring;


architecture behv of Ring is
  shared variable snd_addr: integer := 1;
  shared variable rcv_addr: integer := 0;
  constant tau: time := 0.125 ns;
  constant N: integer := 2;

begin

  update_full_flags: process(clk_snd,clk_rcv)
    -- variables for uniform
    variable seed1, seed2 : positive;
    variable r : real;
    variable d : time;
  begin
    if (clk_snd'event and clk_snd='1') then
      -- write the full flag
      uniform(seed1,seed2,r);
      d := r * tau;
      full(snd_addr) <= transport 'X' after 0 ps,
                                  '1' after tau; -- (wc) was d

      -- finally increment address pointer
      snd_addr := (snd_addr + 1) mod N;
    end if;
```

```
    if (clk_rcv'event and clk_rcv='1') then
      -- write the full flag
      uniform(seed1,seed2,r);
      d := r * tau;
      full(rcv_addr) <= transport 'X' after 0 ps,
                                  '0' after tau; -- (wc) was d

      -- finally increment address pointer
      rcv_addr := (rcv_addr + 1) mod N;
    end if;
  end process;

end behv;
```

3.9.6 State Machines

While the principles of metastability-containing functions and circuits can be immediately applied to state machines, and in particular, the their state transition functions and the output functions as is, a more in-depth treatment of state machines is appropriate. The following section is based on [30] where we studied a subclass of state machines where metastability is restricted to a single input bit.

We start with a definition that eases writing specifications in the ternary logic $\mathbb{B}_{\mathrm{M}}$.

Definition 86 (from [30])**.** *For strings $x, y \in \mathbb{B}_{\mathrm{M}}^n$, with $n \geq 1$, let $x * y \in \mathbb{B}_{\mathrm{M}}^n$ such that*

$$\forall i\colon (x * y)_i = \begin{cases} x_i & \text{if } x_i = y_i \\ \mathrm{M} & \text{otherwise} \end{cases} .$$

Then, for $g : \mathbb{B}^n \to \mathbb{B}^m$, the metastable closure $[g] : \mathbb{B}_{\mathrm{M}}^n \to \mathbb{B}_{\mathrm{M}}^m$, can be compactly written as

$$\forall i\colon [g]_i(x) = \mathop{*}_{x' \in \mathrm{Res}(x)} g_i(x') .$$

Recall that the transition function f in a state machine depends on the current state and the input. We refer to this input as *datapath control* as opposed to *datapath data* that does not influence state transitions. Datapath control is assumed to be $c \geq 0$ bits wide and the state to be encode with $b \geq 1$ bits. States from $S = \{0, \ldots, |S|\}$, with $|S| \geq 1$, are encoded via the injective function $\varepsilon : S \to \mathbb{B}^b$. If clear from the context we refer both to $s \in S$ and $\varepsilon(s)$ as a state.

We may now additionally interpret a state s in $\mathbb{B}_{\mathrm{M}}^b$ as superpositions of classical states, i.e., a set of states s_{M} that comprises of all $\varepsilon^{-1}(s')$ with $s' \in \mathrm{Res}(s)$, if it is well-defined for all such s', and a default value $\perp$ otherwise.

An execution of a state machine is defined analogously to the round model in Section 3.3.3. We just deviate in one aspects to adapt the model to a more refined analysis for state machines:

- Executions are allowed to non-deterministically evolve via arbitrary resolutions of the current state. In particular, denoting by $x(r)$ the datapath control part and by $y(r)$ the encoded state at round $r \geq 1$, it is
$$y(r) \in [f](y(r-1), x(r-1)) . \tag{3.18}$$
In case $y(r-1) = \perp$, we allow arbitrary $y(r)$ from $\mathbb{B}_{\mathrm{M}}^b$.

- Further we assume that only the initial input is potentially metastable, and that after an *a priori* defined $k \geq 1$ number of rounds, metastability has resolved. We thus have for $r = k + 1$,
$$y(r) \in \mathrm{Res}\left([f](y(r-1), x(r-1))\right) . \tag{3.19}$$

Example 87. *Assume a system with two datapath control bits* REQ *and A. Figure 3.29 on page 69 depicts an example state machine and an execution over* 3 *rounds. We assume that states are*

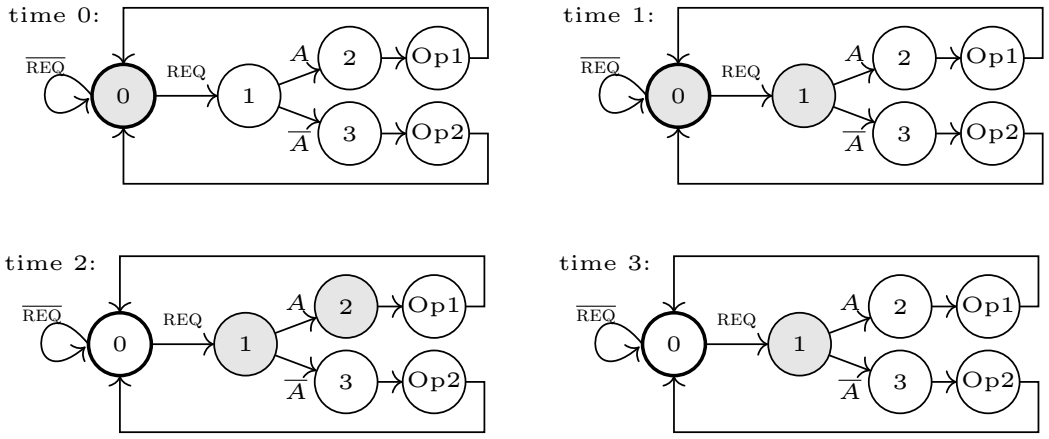


Figure 3.29: Example execution of state transitions. The current superposition state is shown in gray. In the execution the superposition collapses at time 3 into a single state. From [30].

encoded as $\varepsilon(0) = 000$, $\varepsilon(1) = 001$, $\varepsilon(2) = 011$, $\varepsilon(Op1) = 010$, $\varepsilon(3) = 101$, *and* $\varepsilon(Op2) = 100$. *For the input sequence, we assume that* REQ *is of the form* $0^*(\mathrm{M})?1^\omega$, *and* A *is either* 0^ω *or* 1^ω. *This corresponds to a scenario where the circuit has already received the (stable) control data* A, *and is waiting to receive the* REQ *to trigger the computation.*

In the example, we see that the encoding is optimal for the given input traces: the set of superpositions is minimal. Further, observe that at round 3 *the state superposition has collapsed into a single state — this may be the case either by the non-determinism of the execution, or enforced by assuming* $k = 2$.

By contrast assume the encoding as before, but with $\varepsilon(1) = 111$. *One observes that at time* 1, *the state machine is in the superposition state* $\perp$ *allowing arbitrary future behavior.*

The encoding in Example 87 is ad-hoc. So the question arises, whether there exist efficient (in terms of bit-size) encoding, that have the same properties as the presented encoding — an optimal superposition set. In [30] we show that this is the case, and present two efficient encodings for state machines with only a single initially potentially metastable REQ bit that behaves like in Example 87. The solutions are based on appending additional bits to the state encodings that mark the path (e.g., distance as thermometer encoded bits) from the initial state for distances up to k. The question of optimal encodings with $\lceil \log(|S|) \rceil$ bits is an open problem, however.

3.9.7 A Circuit for Voltage Droop Tolerance

In [31] we applied the techniques of building metastability-containing circuits to design a new clock frequency adaptation circuit. The purpose of this circuit is to slow down the clock frequency driving a synchronous circuit upon detection of a voltage droop, i.e., a sudden power supply drop. Slowing down the clock frequency is required if the synchronous circuit operates near its maximum allowed clock frequency and small droops would already lead to violations of the setup-constraint between the critical path and the launching and capturing clock paths.

Alternatively, the classical solution is to equip the clock cycle with large enough guardbands. However this is costly in terms of performance: For example, Bowman *et al.* [107] showed that a 12% droop in a 45 nm microprocessor required to slow down the clock frequency by 16% to accommodate the increased critical path delay.

High-frequency voltage droops. V_{DD} variations exists at different timescales, ranging from slow temperature and aging effects to GHz localized noise. While slow variations can be coped with by classical controllers, and multiple GHz noise is too localized and fast to be coped with by current circuits, it is the MHz domain that has recently been addressed with fast frequency adaptation circuits. Typically effects in this domain are IR drops due to parasitic resistances in the power supply net and $\mathrm{d}I/\mathrm{d}t$ induced noise due to die and package LC [108–110]. Both are

varying in time, e.g., due to activity profiles. Measurements of such V_{DD} variations when injecting sudden switching activity changes showed damped oscillatory behavior [110].

Previously existing solutions. In the following we focus on synchronous and GALS solutions; there is a large body of work on handshaked and (partially) clockless circuits, however, which, additionally to other factors, is motivated by the above problems.

An interesting, and deceptively simple, solution is a ring oscillator whose delay lines are configured to have similar voltage-delay characteristics as critical path delays in the synchronous circuitry they clock [111]. The ring oscillator is then used as an almost instantaneously adapting frequency adaptation module. This solution can be either used globally, or locally do generate clocks for the synchronous islands in a GALS. Fojtik *et al.* [110] proposed such a solution, which additionally uses circuitry to communicate between such islands without the need for deep synchronizers. They propose to combine pausible clocking with ring oscillators. They also advocate the use of local frequency adaptation over centralized, global adaptation since the latter suffers from long insertion delays of the clock tree, which may be prohibitive for fast reaction to droops. Further, local adaptation models reduce droops as they smoothen activity profiles [110].

A large body of solutions is based on a droop detector combined with a frequency adaptation module. Solutions with frequency adaptation modules have the advantage that phase relations between local adaptation modules typically are directly available as digital states. This may be important for fast communication between such synchronous islands. Solutions with slow and fast adjusting frequency adaptation based on PLLs [112] and/or selecting among clock signals after droop detection [113] have been proposed. An example droop detector is proposed in [107]. It is based on mixed gate-interconnect delay line monitoring. The solution in [114] uses the fact that a droop delays not only the critical path, but also the clock launch and capture paths. They thus concentrate on the potentially hazardous reverse effect of decreased frequency after the droop is over, and mask the clock signal until the droop is over.

Direct implementation with phase shifts. Figure 3.30 shows the principle design of a phase shift module (φ) acting as a frequency adaptation module and a droop detector. The module φ receives a, typically, stable clock Clk_{in} and produces the shifted clock signal Clk_{out}. The phase shift between Clk_{out} and Clk_{in} is incremented by a fixed amount if the enable signal is active at an active clock edge of Clk_{in}. The droop detector (DD) drives the enable signal, repeatedly shifting the clock signal and thus slowing down its frequency during a droop.

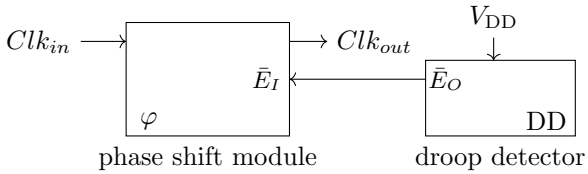


Figure 3.30: System architecture of a direct frequency adaptation module implementation: the DD senses occurrence of a droop and signals the φ module when to shift the phase of the input clock.

The design suffers from potential metastable-upsets, however: the enable signal provided by the droop detector is not aligned to the clock input signal a priori. Thus phase shifts may be triggered exactly when an active input clock edge occurs, leading to glitches in the output clock signal.

Synchronizing the enable signal. Figure 3.31 on page 71 shows a natural solution to this problem: the use of synchronizers for the enable signal. Different shades of red symbolically show the decrease of probability of a metastable enable output from right to left in the figure.

Figure 3.32 depicts the circuit's operation as a timing diagram from the occurrence of a droop, its synchronization in the synchronizer stages to the input clock signal, and the final application of a phase shift at the phase shift module. Again, shades of red are used to symbolically depict the probability of metastable output signals. The drawback of this solution is immediately visible from

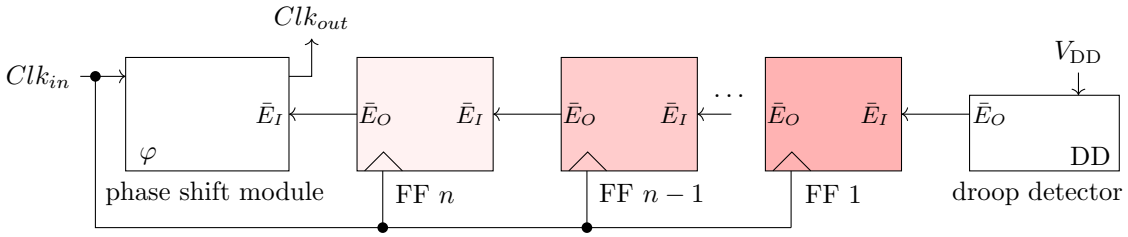


Figure 3.31: System architecture of the frequency adaptation module implementation with synchronizers for the enable signal.

the timing diagram: with increasing probability of a metastability-free enable through deeper synchronizer chains, the reaction time of the circuit increases. This results in missing high-frequency droops.

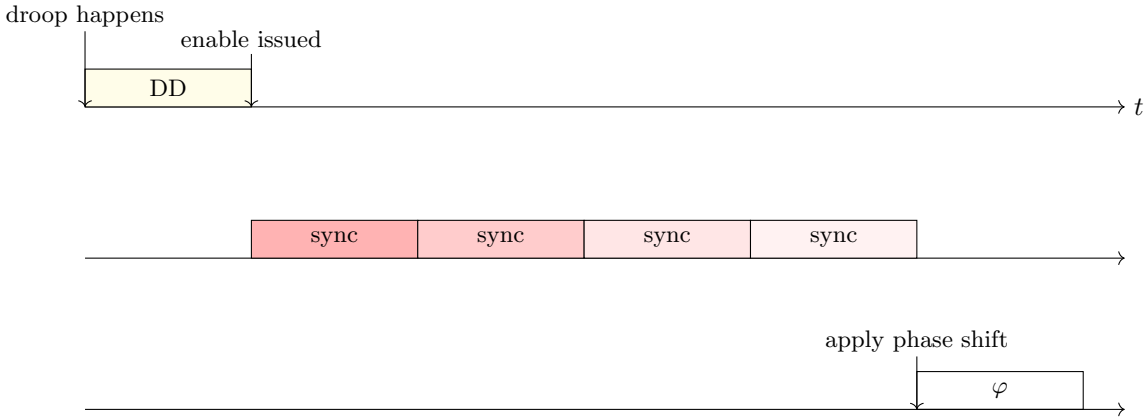


Figure 3.32: Timeline form droop happening until application of the phase shift at the phase shift module φ.

Metastability-containing design. In [31] we proposed and proved correct a different solution that does not have the problem of coupling latency and lower probability of metastable upsets. The design is shown in Figure 3.33 Again, the DD senses the occurrence of a droop and sets its enable output to active during a droop. In fact the enable signal is split into two signals, that we will discuss later and that can be viewed as duplicates for the moment. The enable then travels from the right to the left through a pipeline of delay elements (DE). These apply a phase shift between signals C_I and C_O by switching their internal delay from short to long. From the other direction, left to right, the clock signal is fed into the pipeline of DEs with a respective phase shift from $\tau_{i,k}$ to $\tau_{i,k+1}$ applied at each DE. Each delay element also hands over the enable signal to the left, finally arriving at φ. This makes sure that the phase shifts are also applied to following clock transitions; preventing them from being too close to the initially delayed/shifted clock transition.

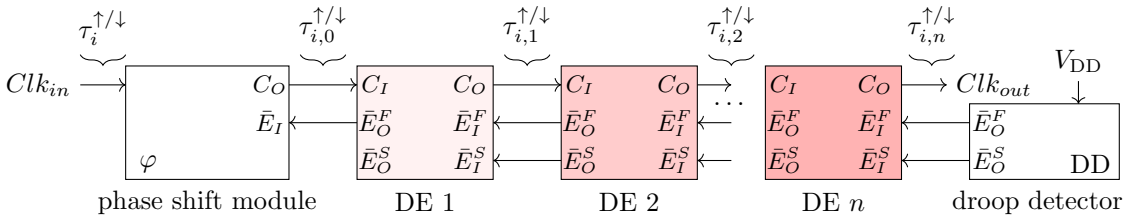


Figure 3.33: System architecture of the proposed frequency adaptation module implementation from [31].

Figure 3.34 shows the timeline from the occurrence of a droop to the application of a phase shift. If the DD issues a clear active enable, the shift is immediately applied at the last delay element. Potentially the DD may be about to change its enable output during a clock transition happening at the last delay element DE n. In this case the enable output signal(s) of DE n may become metastable, and a priori, this may lead to a glitch at the clock output C_O. In [31] we proposed a design based on a metastability-containing implementation of DE that effectively computes with such signals and prevents a glitch form occurring. The latter is possible since the output space of a phase shift is continuous. But, since we cannot prevent enable signals from becoming metastable, we split them into two signals with (monotonicity) guarantees between them when becoming metastable. The interested reader is referred to the article [31] for the circuitry and its analysis.

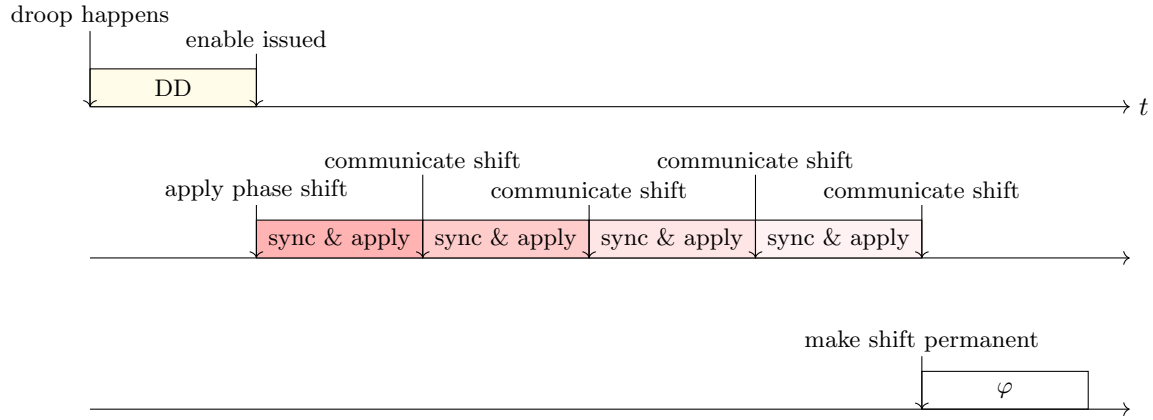


Figure 3.34: Timeline from droop happening until application of the phase shift at the phase shift module φ.

Differently put, the timeline in Figure 3.34 can be interpreted as as replacing the synchronizing element by an element that computes/acts (in this case applies a phase shift) and synchronizes at the same time.

Besides a correctness and performance analysis, Spice simulation of this design are provided in [31], showing that clock frequencies in the GHz domain are indeed feasible.

Further reading. Dolev *et al.* [13] designed and proved correct a distributed clock generation circuit that is probabilistically self-stabilizing, and tolerates less than a third of Byzantine faulty nodes. Two key design constraints were a required low number of interconnect between nodes, and the necessity to deal with metastable upsets. Bund *et al.* [115] proposed a distributed clock generation circuit that is not necessarily fully connected and that provides gradient clock synchronization guarantees. Key design constraints were again a low number of interconnect, small node circuitry, tighter synchronization bounds than for clock trees, and the correct handling of metastable upsets.

Chapter 4

Dynamic Networks

In previous chapters we assumed that the machine's architecture is static. While this is plausible for standard VLSI designs in silicon, accounting for faults and inherent dynamic configurations of components requires to consider dynamic architectures. The concrete requirements of such systems may vary, depending on whether the machine changes behavior, e.g., due to a radiation fault or a component moving in space. At a certain level of abstraction, however, there is no difference between the two machine models: messages between its components are potentially dropped or corrupted. In this chapter, we study computing in the presence of such dynamic machine models.

This chapter discusses results from the following research articles.

- [116] Charron-Bost, Függer, Nowak. *Approximate consensus in highly dynamic networks: The role of averaging algorithms.* In 42nd International Colloquium on Automata, Languages, and Programming (ICALP), pages 528–539, 2015.

  Within dynamic networks that are described by a network model, i.e., a set of communication graphs from which the adversary may freely choose in each round, we characterized solvability of approximate consensus and asymptotic consensus. Surprisingly, there exist simple solutions, namely averaging algorithms, for the solvable cases.

- [117] Charron-Bost, Függer, Nowak. *Fast, robust, quantizable approximate consensus.* In 43rd International Colloquium on Automata, Languages, and Programming (ICALP), volume 55, pages 137:1–137:14, 2016.

  We show that simple averaging algorithms do not only solve asymptotic and approximate consensus in highly dynamic networks, but can be slightly adapted to yield fast and robust solutions, with several potential applications.

- [118] Függer, Nowak, Winkler. *On the Radius of Nonsplit Graphs and Information Dissemination in Dynamic Networks.* Discrete Applied Mathematics, 2020.

  The dynamic radius of a sequence of communication graphs is related to the speed of information dissemination from a single process within the network. We show that the dynamic radius of a sequence of non-split graphs with n nodes is in $O(\log \log n)$.

- [119] Függer, Nowak, Schwarz. *Tight bounds for asymptotic and approximate consensus.* In ACM Symposium on Principles of Distributed Computing (PODC), pages 325–334, New York, NY, USA, 2018. ACM.

  We show that the approximate agreement algorithms from [117] are indeed optimal with respect to time complexity in dynamic networks. Our results also imply a lower bound on time complexity in classical static networks, answering a question by Dolev *et al.* [81] on lower bounds for general approximate agreement algorithms.

- [120] Charron-Bost, Függer, Nowak. *Multidimensional asymptotic consensus in dynamic networks.* CoRR, abs/1611.02496, 2016, and

  [121] Függer, Nowak. *Fast multidimensional asymptotic and approximate consensus.* In 32nd International Symposium on Distributed Computing (DISC), Leibniz International Proceedings in Informatics (LIPIcs), pages 27:1–27:15, 2018.

  We present new algorithms for asymptotic and approximate consensus with multidimensional input values. Our analysis shows that the algorithms work in dynamic networks and improve the time complexity from the previously fastest approximate consensus algorithm in asynchronous message passing systems with Byzantine faults by Mendes *et al.* [122].

4.1 Models of Dynamic Networks

Traditionally, early work in distributed computing [80] mapped changes in a distributed machine to nodes or links within an inherently stable fully connected network. However, any such adversarial model can also be viewed as a model of a dynamic network for this particular setting. Drawing the line between distributed algorithms for dynamic networks and for classical static networks thus becomes a matter of taste. Indeed, some of the results that will be discussed in this section for highly dynamic networks will have direct applications to static networks.

Early work that emphasized on the dynamic aspect of networks is by Santoro and Widmayer [123]. They studied agreement problem variants in a network with message faults and where processes communicate and compute in synchronous rounds. Among other results, they show that the classical exact consensus/agreement problem is not solvable in a network of $n > 1$ nodes, if the adversary is allowed to remove $n - 1$ messages per round. In fact their impossibility proofs only require the adversary to choose up to $n - 1$ omissions from a (different) sender node at each round.

Charron-Bost and Schiper [124] introduced a unified framework for distributed algorithms that communicate within communication-closed rounds and where messages can only be deleted. Benign adversaries can thus be described as sets of allowed communication schedules, i.e., sequences of communication graphs.

Afek and Gafni [125] studied round-wise operating distributed algorithms with the intention to relate solvability of certain problems in such message passing systems to solvability in shared memory systems. The allowed communication schedules are only restricted round-wise: the adversary may schedule a communication graph if a predicate P holds on the graph. Such message adversaries were termed *oblivious* in [126]. A predicate is called message adversary in this context. Like in [123, 124], communication graphs are not necessarily bidirectional. However, unlike this work, Afek and Gafni consider only anonymous message adversaries, i.e., if $P(G)$ for a graph G than $P(G')$ for any G' with renamed nodes.

Raynal and Stainer [127] studied message adversaries in the context of failure-detectors, i.e., oracles that predict properties on communication networks.

Kuhn *et al.* [128] studied distributed problems, like counting nodes, and on top of this, distributed function computation, in dynamic networks. By contrast to the work covered in this chapter, the communication graphs are bidirectional, however.

Casteigts *et al.* [129] systematically studied dynamic network topologies and reductions among certain classes of such networks; formalizing the concept of time varying graphs for distributed systems and providing (simulation) relations between classes of such networks. Like [128], graphs are bidirectional by definition.

In the particular context of solving exact consensus in dynamic networks, Coulouma *et al.* [126] characterized the oblivious massage adversaries in which exact consensus is solvable. Winkler *et al.* [130] characterized so called closed[1] message adversaries in which exact consensus is solvable, and Nowak *et al.* [131] arbitrary message adversaries where this is the case.

A broad overview on computing in dynamic communication networks has been presented by Kuhn and Oshman [132].

[1]The authors call a message adversary, i.e., a set of allowed communication schedules, closed if for each sequence that is not in the set there exists a prefix with all extensions not being in the set.

4.2 Network Models

We start with introducing a model along the lines of [116]. Consider the set $[n]$ of $n \in \mathbb{N}^+$ processes/nodes all of which operate in synchronous rounds, broadcasting a message in each round to all other processes, receiving messages from a subset of processes that have been sent in this round, and performing a computational step. The messages that are successfully received (and not altered) within this round can be described by a graph, with a link from process i to j, if the message has been received, and no link, otherwise. If not stated otherwise, we assume that messages are never altered. Further, we assume the existence of self-loops, i.e., processes always receive their own messages. Call such a directed graph a *communication graph*. For a communication graph G, we denote the incoming neighbors of a node i by $\mathrm{In}_i(G)$ and the outgoing neighbors by $\mathrm{Out}_i(G)$. Likewise, for a set S of nodes, we set $\mathrm{In}_S(G) = \bigcup_{j \in S} \mathrm{In}_j(G)$; and similarly for $\mathrm{Out}_S(G)$.

While several notions of stability in dynamic networks exist, e.g., how many edges can be removed in a communication graph from one round to the next, a lower bound on the existence of an edge before it may be deleted, the arguably most dynamic case is the case where the message adversary can freely choose communication graphs from a pre-defined set. Call such a (non-empty) set of graphs with nodes in $[n]$ a *network model*. A network model thus corresponds to an *oblivious message adversary*.

A *communication sequence* $\mathcal{G}$ is a countably infinite sequence of communication graphs $(G(k))_{k \geq 1}$. We say the communication sequence is from network model $\mathcal{N}$ if all of the communication sequence's graphs are from $\mathcal{N}$.

We are now in the position to define how a distributed system advances in the presence of a communication sequence. Process i's *local state* at the beginning round $k \geq 1$ is denoted by $s_i(k-1)$, and its state at the end of round k is $s_i(k)$. *Local states* are from a potentially infinite set of states. The *global state*, or *configuration*, at (the beginning of) round k is the collection of local states at the beginning of round k, one per process.

A *distributed algorithm* is a state machine that acts on the global state space, with its transition function restricted to locality constraints: it specifies (part of) the initial global state $s(0)$ at the beginning of round 1, as well as a transition function f_i that determines for each process i, the next state $s_i(k)$ from the states $s_j(k-1)$ of all $j \in \mathrm{In}_i(k)$ it has received in round k, i.e., for all rounds $k \in \mathbb{N}^+$,

$$s_i(k) = f_i\left(\{s_j(k-1) \mid j \in \mathrm{In}_i(k)\}\right) \quad . \tag{4.1}$$

While f_i depends only on the set of received states, any ordering may be encoded within the received states, e.g., by appending process identifiers. Further, while the model assumes that processes broadcast their complete local state in every round (potentially including process identifiers, round number, reception history, etc.), realistic implementations may broadcast only the part of the state that determines the algorithm's behavior. An algorithm that does not make use of identifiers, may simply omit broadcasting these.

Requiring that only part of the initial global state $s(0)$ may be specified by the algorithm, the remaining part will serve as the initially presented *input* — analogously to the computational model presented in Section 3.3.3 where inputs are initially loaded into the input register and not changed by the environment during computation.

An *execution* of a distributed algorithm from global initial state $s(0)$ with communication sequence $\mathcal{G} = G_1, G_2, G_3, \ldots$ is defined as the alternating sequence of global states and communication graphs

$$s(0), G_1, s(1), G_2, s(2), G_3, \ldots$$

obtained by successive application of the distributed algorithm's transition function, given the communication sequence $\mathcal{G}$.

4.2.1 Non-split and Rooted Graphs

A communication graph is *non-split* if any two of its nodes have a common incoming neighbor. Figure 4.1 (a) depicts a communication graph that is readily seen to be non-split: nodes 1 and 2

(a) non-split graph (b) rooted graph

Figure 4.1: (a) Non-split and (b) rooted communication graph with set of nodes [3]. Self-loops, that exist at all nodes, are not shown.

have common incoming-neighbors $\{1, 2\}$, nodes 2 and 3 common incoming-neighbors $\{1\}$, and nodes 1 and 3 common incoming-neighbors $\{1\}$ by presence of the self-loop at 1. The graph in Figure 4.1 (b) is not non-split since nodes 2 and 3 have no common incoming-neighbors. While our treatment of non-split graphs is based on the work on approximate agreement in [116], these graphs play an important role for several applications — for example, within the work by Charron-Bost and Schiper [124] studying such graphs as "no split predicates" and Cao *et al.* [133] as "neighbor-shared graphs".

A communication graph is *rooted* if there exists a node i (a root) with a path from i to any of the graph's nodes. The root is not necessarily unique, and the set of roots of a communication graph G has been referred to as the *root component* of G, since the subgraph that consists of all roots is strongly connected [134]. The nodes of the root components, the roots, are denoted by $R(G)$. The graph in Figure 4.1 (a) is rooted with root component $\{1, 2\}$, and the graph in Figure 4.1 (b) with root component $\{2\}$. Like non-split graphs, rooted graphs occur in diverse applications.

4.2.2 Relations between Non-Split and Rooted Graphs

Given a non-split communication graph G, one observes by repeatedly applying the non-split definition that G is also rooted:

Lemma 88. *Every non-split graph is rooted.*

While the existence of a root r is immediate, its *eccentricity* within G, i.e., the maximum of the minimum path length from r to i, for all $i \in [n]$, is not. The *radius* of a directed graph is the minimum eccentricity over all nodes. As such, it has implications within several distributed applications. Consider, e.g., a communication graph G that resembles a network topology of a distributed system that operates in rounds. Then the radius is a tight lower bound on when everyone has heard from a common process.

Example 89. *The radius of the communication graph in Figures 4.1 (a) is* 1 *since all nodes are reachable from node* 1 *over* 1 *edge. The graph in Figure 4.1 (b) as radius* 2 *with all nodes reachable from node* 2 *over* 2 *edges.*

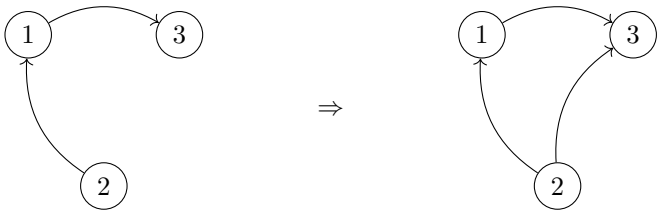
Repeated application of the non-split definition, with incoming neighbors forming a binary tree yield a radius in $O(\log n)$. However, in [118], we have shown that a stronger bound holds for the radius:

Theorem 90 ([118])**.** *The radius of a non-split communication graph with n nodes is in $O(\log \log n)$.*

To compare, Figure 4.1 (a) is non-split and has radius 1. Only a constant lower bound of 3 is known from an example non-split graph, leaving a tight upper bound an open problem.

In the previous considerations, we have assumed a single, fixed communication graph G. However, with adapted definitions, the above two results also hold in presence of dynamic communication graphs.

For that purpose let the product communication graph $G = G_1 \circ G_2$ of two communication graphs G_1 and G_2 with same set of nodes $[n]$, be the graph with set of nodes $[n]$ and an edge



original rooted graph G G^2 is non-split

Figure 4.2: Non-split graph from graph product. Self-loops are not shown.

from i to j in G, if there is a $k \in [n]$ such that (i,k) is an edge in G_1 and (k,j) is an edge in G_2. One observes, that for a communication sequence $G_1, G_2, \ldots$, and a distributed algorithm that relays messages, the processes from which a node j has indirectly received a message at the end of round 2, are exactly those in $\mathrm{In}_j(G_1 \circ G_2)$. The above graph product thus captures collapsing several rounds into a single macro-round.

Following [118], the *dynamic radius* of a sequence of communication graphs $G_1, \ldots, G_\ell, \ldots$ is the smallest $\ell \geq 1$, such that $G_1 \circ \cdots \circ G_\ell$ contains a star graph. Again, the dynamic radius is a tight lower bound on when everyone has heard from a common process in the first ℓ rounds.

One can then show that Theorem 90 also holds in presence of a dynamic network:

Theorem 91 ([118])**.** *The dynamic radius of a sequence of non-split communication graph with n nodes is in $O(\log\log n)$.*

This shows that multiplying $O(\log\log n)$ non-split communication graphs with n nodes, one obtains a graph that contains a star graph. The property is trivially fulfilled for the non-split graph in Figure 4.1 (a), since it contains a star graph.

In fact, a similar transition from rooted to non-split by multiplying sufficiently many graphs exists. In [116] we have shown:

Theorem 92 ([116])**.** *The product of $n-1$ rooted communication graphs with n nodes is non-split.*

Example 93. *Let G be the rooted graph in Figure 4.1 (b) with $n = 3$ nodes. While G is not non-split, $G^{n-1} = G^2$ is shown in Figure 4.2, and easily verified to be non-split.*

Theorem 92 has immediate consequences on the time complexity of distributed algorithms to reach agreement among processes; as we will discuss in Section 4.4. Together with Theorem 91, we also obtain:

Theorem 94 ([118])**.** *The product of $O(n\log\log n)$ rooted communication graphs with n nodes contains a star graph.*

4.3 Averaging Algorithms

A distributed algorithm updates its state based on all received states. In previous work, so called averaging algorithms have received particular attention for solving agreement related problems in dynamic networks; see, e.g., [133, 135–138]

Following [116], we define an averaging algorithm as one with an update rule (4.1) of the form,

$$\begin{aligned} s_i(k) &\in [0,1] \\ s_i(k) &= \sum_{j \in \mathrm{In}_i(k)} w_{ji}(k) s_j(k-1) \ , \end{aligned} \tag{4.2}$$

where weights $w_{ji}(k) \in \mathbb{R}^+$, and $\sum_{j\in\mathrm{In}_i(k)} w_{ji}(k) = 1$. While the above definition restricts the output value s_i of every process i to be from $[0,1]$, we will also speak of averaging algorithms if s_i is from $\mathbb{R}$, and more generally, from $\mathbb{R}^d$ with dimension $d \geq 1$. For the moment, we assume bounded scalar values in $[0,1]$, however.

Averaging algorithms differ in how weights are locally computed from received states. Three natural averaging algorithm classes are:

- The *equal neighbor averaging algorithm* for which

$$w_{ji}(k) = \frac{1}{|\operatorname{In}_i(k)|} \ .$$

- The *mean-value averaging algorithm* with

$$w_{ji}(k) = \frac{1}{|\{s_j(k-1) \mid j \in \operatorname{In}_i(k)\}|} \ .$$

- A *fixed weight averaging algorithm* for the network model $\mathcal{N}$. Let d_i be the maximum in-degree of node i over all graphs in $\mathcal{N}$, and let $\hat{d}_i \geq d_i$. Then,

$$w_{ji}(k) = \begin{cases} \frac{1}{\hat{d}_i} & \text{if } i \neq j \\ 1 - \sum_{j \in \operatorname{In}_i(k),\, j \neq i} w_{ji}(k) & \text{otherwise} \end{cases} \ .$$

- The *midpoint algorithm* has $w_{ij}(k) = \frac{1}{2}$ for $j \in \{m, M\}$, where m, M are such that $s_m(k-1)$ and $s_M(k-1)$ span the interval of the values received by process i in round k. The name is due to the fact that a process sets its new value to the midpoint of the values it receives. The midpoint algorithm lies at the heart of a distributed clock synchronization algorithm proposed by Welch and Lynch [89].

- The *jumping algorithm* analyzed in Charron Bost *et al.* [117] is specified only for $n = 2$ processes. In case a process i receives a value, it sets its own weight to $w_{ii}(k) = \frac{1}{3}$ and the other process' weight to $\frac{2}{3}$.

Parameters and safety of averaging algorithms. In [116], we classified averaging algorithms with respect to two parameters that we have shown to impact time complexity in agreement problems. Towards that goal, we introduce some definitions.

For a set $A \subseteq \mathbb{R}^d$, with dimension $d \geq 1$, its diameter is given by

$$\operatorname{diam}(A) = \sup_{x,y \in A} ||x - y|| \ . \tag{4.3}$$

Note that if $d = 1$ and the set has finite cardinality, then $\operatorname{diam}(A) = \max A - \min A$.

For a process $i \in [n]$ and round $k \in \mathbb{N}^+$, the set of received values is denoted by

$$V_i(k-1) = \{s_j(k-1) \mid j \in \operatorname{In}_i(k)\}$$

and its minimum and maximum value by

$$\begin{aligned} m_i(k-1) &= \min V_i(k-1) \\ M_i(k-1) &= \max V_i(k-1) \ . \end{aligned}$$

The diameter of all output values in round k is denoted by

$$\delta(k-1) = \operatorname{diam}\left(\{s_i(k-1) \mid i \in [n]\}\right) \ .$$

We say an averaging algorithm has *parameter* α, for $\alpha > 0$, in a network model $\mathcal{N}$, if for all nodes $i, j \in [n]$ and rounds $k \in \mathbb{N}^+$ in all its executions in $\mathcal{N}$,

$$w_{ji}(k) \geq \alpha \ .$$

We omit the reference to the network model if the statement holds for all network models. The condition is identical to Assumption 1 in Blondel *et al.* [136]. Note that an averaging algorithm with a parameter $\alpha > 0$ never drops received messages, i.e., does not set weights of received values to 0.

Example 95 (Parameters of some averaging algorithms)**.** *The equal neighbor averaging algorithm is an averaging algorithm with parameter* $1/n$*, the mean-value averaging algorithm with parameter* $1/n$*, and the fixed weight averaging algorithm with parameter* $\min_{i\in[n]}\{1/\hat{d}_i, 1-\sum_{j\in \mathrm{In}_i(k),\, j\neq i} w_{ji}(k)\}$*. The midpoint algorithm does not have a parameter greater than* 0.[2]

Further, an averaging algorithm is *σ-safe*, for $\sigma \in (0, \frac{1}{2}]$, in network model $\mathcal{N}$, if for all nodes $i \in [n]$ and rounds $k \in \mathbb{N}^+$ in all its executions in $\mathcal{N}$,

$$\sigma M_i(k-1) + (1-\sigma)m_i(k-1) \leq s_i(k) \leq (1-\sigma)M_i(k-1) + \sigma m_i(k-1) \ ,$$

i.e., if process i's new value lies within relative distance σ to the interval boundaries $m_i(k-1)$ and $M_i(k-1)$ of the received values. Again, we omit the reference to the network model if the statement holds for all network models.

The following relation holds between averaging algorithms with parameter and safety.

Lemma 96 ([117])**.** *If an averaging algorithm has parameter* $\alpha > 0$ *in network model* $\mathcal{N}$*, then it is* $\alpha - safe$ *in network model* $\mathcal{N}$*.*

Proof. Consider an averaging algorithm that has parameter $\alpha > 0$ in network model $\mathcal{N}$ For $k \in \mathbb{N}^+$, and with $\hat{j} \in [n]$ being a process whose $s_j(k-1)$ is minimal among those received by process i, it is

$$\begin{aligned} s_i(k) &= \sum_{j\in \mathrm{In}_i(k)} w_{ji}(k)s_j(k-1) \\ &= \sum_{j\in \mathrm{In}_i(k)\setminus\{\hat{j}\}} w_{ji}(k)s_j(k-1) + w_{\hat{j}i}m_i(k-1) \\ &\leq \sum_{j\in \mathrm{In}_i(k)\setminus\{\hat{j}\}} w_{ji}(k)M_i(k-1) + w_{\hat{j}i}m_i(k-1) \\ &\leq (1-\alpha)M_i(k-1) + \alpha m_i(k-1) \ . \end{aligned}$$

The lower bound follows by analogous arguments. □

Example 97 (Safety of some averaging algorithms)**.** *From Lemma 96, one immediately obtains: The equal neighbor averaging algorithm, the mean-value averaging algorithm, and the fixed weight averaging algorithm are* α*-safe, where* α *is their respective parameter.*

Further, by definition, the midpoint algorithm is $\frac{1}{2}$*-safe.*

4.4 Consensus Problems

A typical coordination task in a distributed system is to achieve (approximate) agreement on (part of) the processes' states. We call these *consensus problems*. The precise definition of when two processes are considered to agree on a state, among other problem parameters, as well as the distributed computing model, lead to either solvable or unsolvable combinations — the former with different degrees of algorithmic performance. In this chapter we will focus on the consensus problems that we studied in [116, 117, 119, 121].

As a distributed computing model we will consider highly dynamic distributed systems specified via network models. Further, we assume that the state space of each process $i \in [n]$ is of the form

$$\begin{aligned} s_i = (x_i, \mathrm{dec}_i, \mathrm{loc}_i) \text{ with} \\ x_i \in [0,1] \, , \\ \mathrm{dec}_i \in \{\perp\} \cup [0,1] \, , \text{ and} \\ \mathrm{loc}_i \in \mathcal{L} \ , \end{aligned}$$

where $\mathcal{L}$ is the potentially infinite local state space of a process. While loc_i serves as an internal, algorithm specific, state, x_i and dec_i serve as inputs and outputs to the distributed algorithm solving a consensus problem.

[2]The statement holds for the algorithm as it is specified. However, one can show that the midpoint algorithm can be rewritten into an algorithm with a positive parameter that produces the same outputs.

- **Inputs.** We assume that the input is one value in $[0, 1]$ per process, initially presented to the corresponding process. In our model $x_i(0)$ is the input for process $i \in [n]$. Further, initially, $\mathrm{dec}_i(0) = \perp$.
- **Outputs.** Process outputs for rounds $k \in \mathbb{N}^+$ are $x_i(k)$ and $\mathrm{dec}_i(k)$. We say process i *has (output) value*, respectively, *outputs* $x_i(t)$ in round t. A process may also set $\mathrm{dec}_i(k)$ to $x_i(k)$ — but only once — after which the variable dec_i does not change value anymore. If so, we say process i *decides value* $x_i(k)$ in round k.

Recall the definition of an averaging algorithm from Section 4.3. By slight abuse of terminology, we will call algorithms with states as defined above and that repeatedly compute weighted averages of x_i instead of s_i also averaging algorithms. Clearly, though an equal neighbor averaging algorithm may vary in this context depending on whether on how it sets dec_i for each process i.

We are now in the position to define the different variants of consensus problems discussed in this chapter.

4.4.1 Exact Consensus

An algorithm *achieves (exact, terminating) consensus with the communication sequence* $\mathcal{G} = (G(k))_{k\geq 1}$ if in each execution of the algorithm from a global initial state, that represents valid inputs to consensus problems, with communication sequence $\mathcal{G}$:

- **Termination.** All processes eventually decide.
- **Agreement.** If two processes i and j decide on values v_i and v_j, then $v_i = v_j$.
- **Validity.** Any value decided by a process is among the initial values.

Further, we say an algorithm *solves exact consensus in network model* $\mathcal{N}$ if for all communication sequences with graphs in $\mathcal{N}$, it achieves consensus with the communication sequence. Exact consensus is *solvable in network model* $\mathcal{N}$ if there exists an algorithm that solves exact consensus in $\mathcal{N}$.

In distributed computing, the exact consensus problem is often simply referred to as the consensus problem; see [80] for an overview. Typically, though, termination, and depending on the fault model, also Agreement and Validity, are required to hold only for correct processes. Note that in our setting of a dynamic network, the notion of a correct process does not exist, and we require the properties to hold for all processes. Nonetheless, relations to solving consensus in classical fault models are possible and are discussed later on. Finally, note that different variants of Validity exist in classical literature [80].

4.4.2 Approximate Consensus

Achieving exact consensus is a natural subproblem when synchronizing multiple copies of databases, or in general, in the context of applications that require replica determinism as is the case for mission critical distributed embedded systems [82]: If the state machines at two different processes are in different states, in general, their machines may diverge arbitrarily during the successive rounds. By contrast, this is overly pessimistic for state spaces that contain a certain structure. One example is the state space $[0, 1]$ where nearby states, in terms of a metric on $[0, 1]$, lead to nearby future states in successive rounds. If so, it often suffices to agree on nearby states instead of exactly the same states. In the following we consider the metric space $[0, 1]$ with the distance $d(x, y) = |x - y|$. We discuss generalizations in Section 4.4.4.

Let $\varepsilon > 0$. We say, an algorithm *achieves* ε*-approximate (terminating) consensus with the communication sequence* $\mathcal{G} = (G(k))_{k\geq 1}$ if in each execution of the algorithm from a global initial state, that represents valid inputs to consensus problems, with communication sequence $\mathcal{G}$:

- **Termination.** All processes eventually decide.
- ε**-Agreement.** If two processes i and j decide on values v_i and v_j, then $|v_i - v_j| \leq \varepsilon$.
- **Validity.** Any value decided by a process is within the interval of initial values.

Solvability is defined analogously to the exact consensus problem.

Note that Termination is as in the consensus problem. Agreement has been weakened to ε-Agreement, and Validity has been weakened accordingly to allow decisions on values initially not taken by processes. The problem is also referred to as approximate agreement in literature [80] and has been thoroughly studied in the context of classical fault models like the fully connected communication architecture with up to $f < n$ crashes. For work on approximate agreement; see, e.g., [81, 139, 140].

4.4.3 Asymptotic Consensus

For some applications the requirement of a single decision value per process may either be unnatural, simply not be required, or even too little for an application. For example, in controller applications one typically requires that the controller output converges to the desired set point with the error approaching 0 with increasing number of rounds. The same is desirable for optimization problems: ideally we would like to have an algorithm that achieves better and better solutions with increasing runtime.

Asymptotic consensus is a problem that captures these requirements in an application independent way. We say an algorithm *achieves asymptotic consensus with the communication sequence* $\mathcal{G} = (G(k))_{k\geq 1}$ if in each execution of the algorithm from a global initial state, that represents valid inputs to consensus problems, with communication sequence $\mathcal{G}$:

- **Convergence.** Each sequence of process outputs $(x_i(k))_{k\geq 1}$ converges.
- **Agreement.** If the output sequences of two processes converge, they converge to the same limit.
- **Validity.** If the output sequence of a process converges, then its limit is in the interval of initial values.

Again, solvability is defined analogously to the exact consensus problem.

Note that the above problem does not allow outputs whose distance to each other approaches 0, but who do not converge, but, e.g., oscillate in synchrony.

Asymptotic consensus has received considerable attention from the control community where it is often referred to as consensus. This includes work on asymptotic consensus in dynamic communication architectures with respect to averaging algorithms [133, 141], in dynamic bidirectional communication networks [135], dynamic strongly connected communication networks [136], and work on non-averaging algorithms in static networks [142], among others. For example, Blondel and Olshevsky [143] study sets of matrices, whose products when multiplied with the vector of initial values, converge to a vector with common entries. These so called consensus sets of matrices directly correspond to network models of directed graphs within which averaging algorithms are executed (that determine the weights of the matrices). While the result can be seen as a specialization of the question of which network models allow to solve asymptotic consensus in case only averaging algorithms are permitted, the result is incomparable to our results from [116] presented within this chapter, however: all our network models are required to have self-loops, but algorithms are not restricted to averaging algorithms.

Before proceeding to higher dimensional generalizations of the above problems and a discussions about correctness and performance of algorithms, let us observe the behavior of two asymptotic consensus algorithms within example network models.

Example 98. *Figure 4.3 on page 82 shows example executions for two different averaging algorithms: the equal neighbor algorithm and the midpoint algorithm. Both are executed within a network model comprising three rooted communication graphs G_0, G_1, and G_2 shown in the figure.*

Observe that the midpoint algorithm seems to perform better within its execution as it converges faster within the first shown rounds. While this may a priori be just bad chance for the equal neighbor algorithm since we chose graphs uniformly at random for both executions, we will show during this chapter that better performance bounds can be shown to hold for the midpoint algorithm.

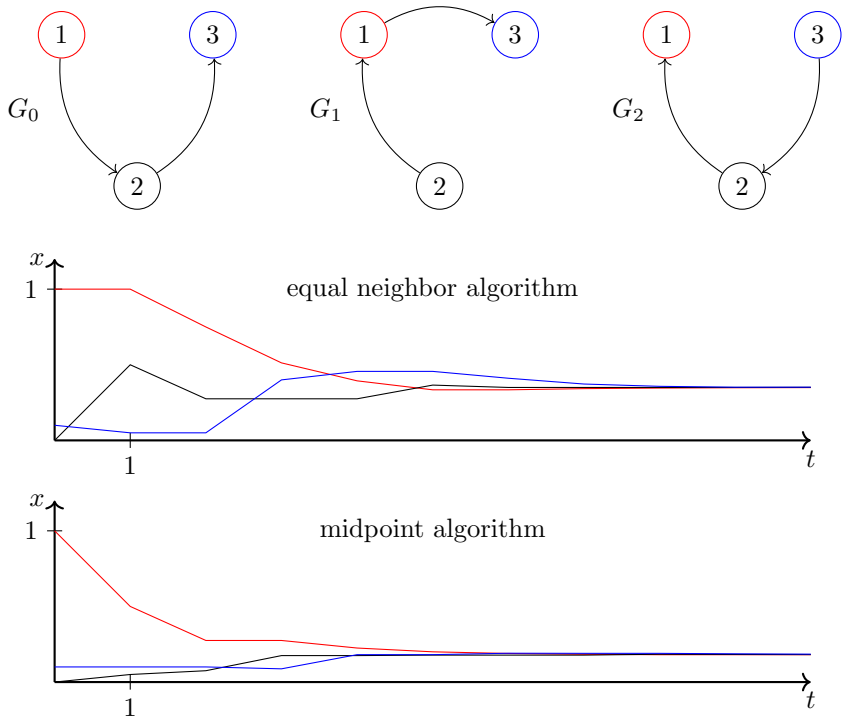


Figure 4.3: Two example executions for two algorithms executed in a system with network model $\mathcal{N} = \{G_0, G_1, G_2\}$ as specified in the figure. There are $N = 3$ nodes, with initial values 1 for node 1 (red, left), 0 for node 2 (black, bottom), and 0.1 for node 3 (blue, right). For each node i its state x_i is depicted over 10 rounds. All graphs are rooted and have been chosen uniformly at random from the network model along the execution.

Example 99. *To address the valid criticism in Example 98 that executions may not be representative, Figure 4.4 on page 83 shows 100 example executions for the same two averaging algorithms as in the example before. Again, communication graphs are chosen uniformly at random from the network model at each round; however the graphs, shown in the figure, differ from Example 98 and have been chosen to be non-split. Further, instead of showing the individual state outputs, we depict the diameter of the set of state outputs at each round. Like in the previous example, the midpoint algorithm seems to perform better. We will later on analyze the performance of both algorithms in non-split network models, and see that this performance ranking coincides with provable bounds.*

4.4.4 Consensus Problems in Higher Dimensions

Until now we assumed a one-dimensional metric state space on $[0, 1]$. Applications like gathering of automated vehicles or drones, control of concentrations to a certain set point, etc., are multidimensional in nature, however. For exact consensus, the generalization is immediate trough reductions in both directions. In the case of asymptotic consensus, for example, for some applications it may be sufficient to converge on each coordinate separately. However, this does not guarantee a natural generalization of the Validity property of asymptotic consensus to stay in the convex hull of the input values. For applications that require this property, a natural generalization of the asymptotic consensus problem is as follows.

Following [121], we define: Let V be some vector space with an inner product $\langle \cdot, \cdot \rangle$ from V^2 to $\mathbb{R}$ and norm $||x|| = \sqrt{\langle x, x \rangle}$. As a typical instantiation we will often use $V = \mathbb{R}^d$, for $d \geq 1$, with inner product $\langle (x_1, \ldots, x_d), (y_1, \ldots, y_d) \rangle = \sum_{i=1}^{d} x_i y_i$.

We say, an algorithm *achieves asymptotic consensus with the communication sequence* $\mathcal{G} = (G(k))_{k \geq 1}$ if in each execution of the algorithm from a global initial state (that represents valid inputs to consensus problems) with communication sequence $\mathcal{G}$, all of the following hold:

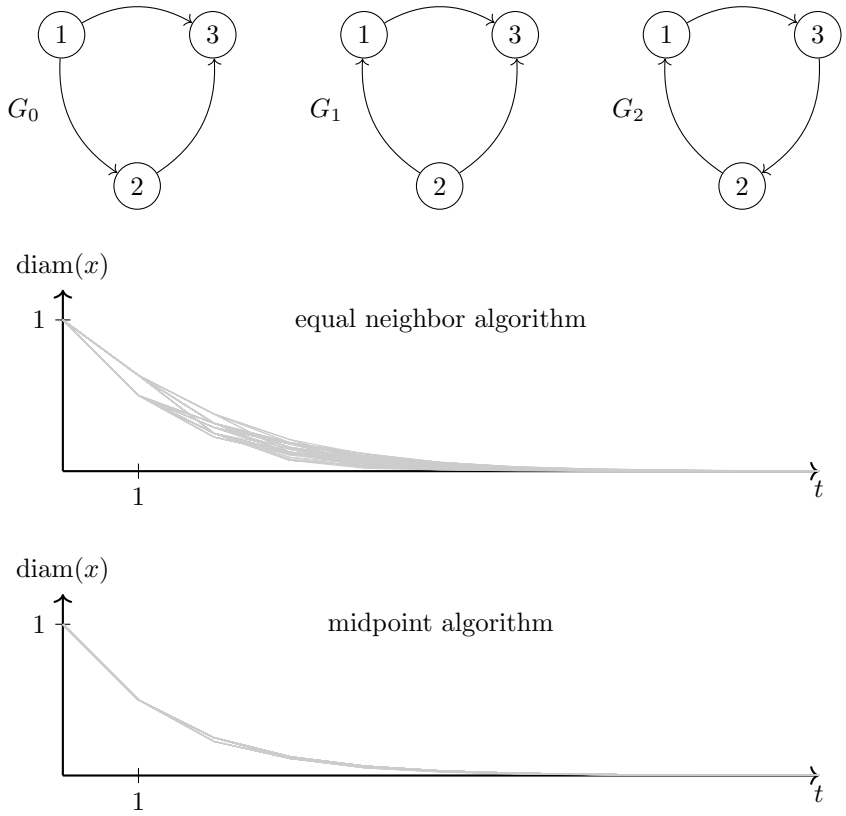


Figure 4.4: Non-split network model comprising of three graphs shown at the top. The diameter of the process output values along 10 rounds is shown for 100 executions of each algorithm. Graphs have been chosen uniformly at random along an execution. The nodes' initial values are 1 for node 1, 0 for node 2, and 0.1 for node 3.

- **Convergence.** Each sequence of process outputs $(x_i(k))_{k\geq 1}$ converges.
- **Agreement.** If the output sequences of two processes converge, they converge to the same limit.
- **Validity.** If the output sequence of a process converges, then its limit is in the convex hull of initial values.

The above definition is analogous to the scalar case, except that we generalized the interval of initial values to the convex hull of initial values in the Validity condition.

Analogously, we say, an algorithm *achieves ε-approximate (terminating) consensus with the communication sequence $\mathcal{G} = (G(k))_{k\geq 1}$* if in each execution of the algorithm from a global initial state (that represents valid inputs to consensus problems) with communication sequence $\mathcal{G}$, all of the following hold:

- **Termination.** All processes eventually decide.
- **ε-Agreement.** If two processes i and j decide on values v_i and v_j, then $||v_i - v_j|| \leq \varepsilon$.
- **Validity.** Any value decided by a process is within the convex hull of initial values.

Here, ε-Agreement has been generalized to the norm on V, and Validity to the convex hull of initial values.

A note on a priori bounded versus unbounded domains. In the one-dimensional case, we assumed that initial values are from a domain that is a priori bounded, in our case $[0, 1]$. Note that

this is central for solving ε-approximate consensus, however, not for solving asymptotic consensus. For the latter, all our negative and positive results also hold for values in unbounded domain $\mathbb{R}$. An analogous statement holds for the higher-dimensional case: we will thus assume initial values to be from $[0,1]^d$ when discussing ε-approximate consensus.

4.5 Solutions to Consensus Problems

We have already mentioned some of the related work on approximate and asymptotic consensus when discussing dynamic networks and consensus problems. Let us briefly sketch the difficulty in solving consensus problems at the example of two publications.

We start with network models that comprise only bidirectional, strongly connected graphs. Kuhn *et al.* [128] showed the following result (rephrased in our model):

Theorem 100 (from [128])**.** *In a network model with only (bidirectional) strongly connected communication graphs, exact consensus is solvable in $n-1$ rounds.*

Proof idea. The algorithm solving exact consensus relays messages during $n-1$ rounds. From the fact that graphs are strongly connected, any node i that has not yet received a value from another node j must be connected via a path from i to j. Along this path there must be two neighbors that differ in the fact whether they have received j's value. Consequently this node must receive j's value. This can happen at most $n-1$ times. □

This promising result is complemented with a result by Santoro and Widmayer [123] who showed that (rephrased in our model) in a network model with graphs that are almost complete and connected but not strongly connected (and not bidirectional), exact consensus is not solvable:

Theorem 101 (from [123])**.** *In a network model that is the set of all communication graphs with $n \geq 2$ nodes, where at most one node has outgoing links to a subset of nodes, but all other nodes have outgoing links to all other nodes, exact consensus is not solvable.*

In particular, such a network model results from an adversary that is allowed to remove up to $n-1$ messages from a fully connected communication graph in each round [123].

Coulouma *et al.* [126] characterized the network models in which exact consensus is solvable; with the above two theorems showing up as special cases.

4.5.1 Approximate Consensus

As previously discussed, for many applications exact consensus in fact is an overkill — raising hopes for less strict conditions on the network model for solvability. Indeed, in [116] we proved the following characterization of network models in which approximate consensus is solvable under quite weak conditions:

Theorem 102 ([116])**.** *Approximate consensus is solvable in network model $\mathcal{N}$, if and only if $\mathcal{N}$ contains only rooted communication graphs.*

We sketch the proof outline that we gave in [117] in the following. For that purpose, say an algorithm is *ϱ-contracting in network model $\mathcal{N}$*, if in all its executions within $\mathcal{N}$, and all rounds $k \in \mathbb{N}^+$,

$$\delta(k) \leq \varrho \cdot \delta(k-1) \ . \tag{4.4}$$

Using the fact that non-split communication graphs have common incoming neighbors for each pair of processes and thus non-empty intersections among the intervals of received values for these processes, we showed that:

Theorem 103 ([117])**.** *In a network model with only non-split communication graphs, a σ-safe averaging algorithm is $(1-\sigma)$-contracting.*

For some of our results, a more conservative contraction bound was used. Observing that, for $0 < \varrho \leq 1$,

$$\varrho \leq e^{-(1-\varrho)} \ ,$$

we have, that an algorithm that is ϱ-contracting in a network model, also is $e^{-(1-\varrho)}$-contracting in that network model.

By repeated application of (4.4), the following bound on the time complexity can be shown for ϱ-contracting averaging algorithms:

Lemma 104 ([117])**.** *For all network models $\mathcal{N}$, and all ϱ-contracting averaging algorithms, with $0 < \varrho < 1$, the algorithm solves approximate consensus in $\mathcal{N}$. It achieves ε-agreement in*

$$\left\lceil \log_{1/\varrho}\left(\frac{1}{\varepsilon}\right)\right\rceil \leq \left\lceil \frac{1}{1-\varrho}\log_e\left(\frac{1}{\varepsilon}\right)\right\rceil$$

rounds.

The inequality follows from the observation before. From the observations in Example 97, we further have:

Lemma 105 ([117])**.** *In any network model with communication graphs with n processes, the equal-neighbor averaging algorithm and the mean-value averaging algorithm are $\frac{1}{n}$-safe. The midpoint algorithm is $\frac{1}{2}$-safe.*

Putting everything together, we thus obtain:

Theorem 106 ([116])**.** *In a non-split network model of n processes, every averaging algorithm that is σ-safe, achieves ε-consensus in*

$$\left\lceil \log_{\frac{1}{1-\sigma}}\left(\frac{1}{\varepsilon}\right)\right\rceil \leq \left\lceil \frac{1}{\sigma}\log_e\left(\frac{1}{\varepsilon}\right)\right\rceil$$

rounds. In particular:

- *The equal neighbor averaging algorithm and the equal value averaging algorithm achieve ε-consensus in $\left\lceil n\log_e\frac{1}{\varepsilon}\right\rceil$ rounds.*
- *The midpoint algorithm achieves ε-consensus in $\left\lceil \log_2\frac{1}{\varepsilon}\right\rceil$ rounds.*

The case $n = 2$. There is an interesting algorithmic solution for $n = 2$ processes [117]: the jumping algorithm. Consider the algorithm for process $i \in [2]$ that sets:

$$s_i(k) = \frac{1}{3}s_i(k-1) + \frac{2}{3}s_j(k-1) \ , \tag{4.5}$$

where $j \neq i$ if i receives a message not only from itself, and $j = i$, otherwise.

While, this algorithm only has parameter $\frac{1}{3}$, and is only $\frac{1}{3}$-safe, one can directly show that it is $\frac{1}{3}$-contracting, i.e.,

$$\delta(k) \leq \frac{1}{3}\cdot\delta(k-1) \ , \tag{4.6}$$

for rounds $k \in \mathbb{N}^+$.

Analysis of macro-rounds. While Theorem 106 is promising for non-split network models, such network models may seem highly limited in their range of application. We will next show that this is not the case, when collapsing rounds into larger blocks of rounds — so called, *macro-rounds*.

First observe, that the output sequence of an averaging algorithm is generated by successive multiplication of the the vector of initial values with adjacency matrices of the communication graphs, whose weights correspond to the weights assigned by the algorithm. Thus,

$$s(k) = W(k) \cdots W(1) \cdot s(0) \ ,$$

where $s(k) \in \mathbb{R}^{n\times 1}$ and $W(k) \in \mathbb{R}^{n\times n}$ with entries $w_{ji}(k)$ as defined in (4.2).

This representation, e.g., has been used in a different proof strategy [116] as discussed in this chapter to establish bounds for averaging algorithms with positive parameters. Here, we stress on another point that has been used in [117] and follow-up work to obtain effective time complexity bounds for averaging algorithms: one may regroup the matrices into blocks of common sizes $K \in \mathbb{N}^+$. For $k \mod K \equiv 0$, we thus obtain

$$\begin{aligned} s(k) &= (W(k) \cdots W(k-K+1)) \cdots (W(K) \dots W(1)) \cdot s(0) \\ &= W'(k) \cdots W'(K) \cdot s(0) \ , \end{aligned} \tag{4.7}$$

with matrices $W'(\ell) \in \mathbb{R}^{n\times n}$ as the respective products of matrices.

Along this lines, following [117], we say a network model $\mathcal{N}$ is *K-non-split*, for $K \in \mathbb{N}^+$, if all products of K communication graphs from $\mathcal{N}$ are non-split. By assessing that the parameters of this hypothetical algorithm shrinks with the power K due to the matrix product, one obtains:

Theorem 107 ([116])**.** *Let $\mathcal{N}$ be a K-non-split network model with n processes. Then, an averaging algorithm with parameter $\varrho > 0$ achieves ε-consensus in*

$$K \left(\frac{1}{\varrho}\right)^K \log_2 \left(\frac{1}{\varepsilon}\right) + K - 1$$

rounds.

We may now apply Theorem 92 on page 77 and obtain that if we choose blocks of length $K = n - 1$ in a rooted network model with n processes, the communication graphs $W'(\ell)$ are non-split. Accordingly:

Theorem 108 ([116])**.** *In a rooted network model with n processes, every averaging algorithm with parameter $\varrho > 0$ achieves ε-consensus in*

$$\left(\frac{1}{\varrho}\right)^n n \log_2 \left(\frac{1}{\varepsilon}\right) + n - 1$$

rounds. For the equal neighbor averaging algorithm, with $\varrho = \frac{1}{n}$, ε-consensus is achieved in $O(n^{n+1} \log_2 \frac{1}{\varepsilon})$ rounds.

Theorem 108 is restricted to averaging algorithms with a positive parameter, and thus a priori cannot be directly applied to the midpoint algorithm that performed well in non-split network models. This is due to the fact that the forming of blocks in (4.7) was performed at matrix-level, resulting in lower bounds on product matrix entries. By contrast, if we multiply these blocks including the initial vector, the fact that an algorithm is σ-safe can be applied to prove bounds instead. Indeed one obtains an analog statement as in Theorem 107 for safety:

Theorem 109 ([117])**.** *Let $\mathcal{N}$ be a K-non-split network model with n processes. Then, a σ-safe algorithm is $(1 - \sigma^K)$-contracting over each block of length K.*

While Theorem 109 may be applied to the midpoint algorithm obtaining a better time complexity bound, we may still improve the bound by considering macro-rounds without averaging.

Amortized algorithms. Revisiting Theorem 92 on page 77 in terms of the macro-round construction yields a new class of algorithms that are not averaging algorithms: Given any averaging algorithm, let the algorithm compute the average of the received values only every $K \in \mathbb{N}$ rounds. In all other rounds, one relays the received values to the other processes. We termed these algorithms *amortized* versions of averaging algorithms [117].

By an analysis similar to the one in the section before, but considering only macro-rounds, one obtains:

Theorem 110 ([117])**.** *In a rooted network model with n processes, every amortized algorithm that is σ-safe, with $\sigma > 0$, achieves ε-consensus in*

$$(n-1)\left\lceil \log_{\frac{1}{1-\sigma}}\left(\frac{1}{\varepsilon}\right)\right\rceil$$

rounds. In particular, for the amortized midpoint algorithm, with $\sigma = \frac{1}{2}$, ε-consensus is achieved in

$$(n-1)\log_2\left(\frac{1}{\varepsilon}\right)$$

rounds.

Impossibility in non-rooted network models. To complete the proof for the characterization in Theorem 102 on 84 one finally shows the following result by a partitioning argument:

Theorem 111 ([116])**.** *No algorithm achieves ε-consensus in a network model that includes a non-rooted communication graph.*

4.5.2 Asymptotic Consensus

While we concentrated on approximate consensus in the previous sections, several of the results presented there can be transfered to statements about asymptotic consensus.

First, observe that asymptotic consensus can be reduced to approximate consensus if precomputable, globally known bounds on the termination time are known. Our solutions fulfill these properties, if the parameter ε and upper bounds on the process number n are globally (and consistently) known. If so, the approximate consensus algorithm may simply be started repeatedly and synchronously at the same round after each termination. The correctness of this approach is seen by viewing each repetition of the approximate consensus algorithm as a macro-round[3] and analyzing contraction of the output values along these macro-rounds.

In fact, however, all presented averaging algorithms, and their amortized variants, can be directly used to solve asymptotic consensus by removing the decision and termination rules. This also removes the requirements of an initial common knowledge of parameters ε and n. Mind though, that amortization may require knowledge of bounds on n if a certain performance is to be guaranteed. We discuss this later on in more detail.

On performance measures. To quantify the speed of an asymptotic averaging algorithm, measures for convergence rates have to replace time complexity measures, though. While the asymptotic consensus problem itself is stated as a limit behavior without any guarantees for finite times, practical solutions will most likely care about the speed of convergence and the guarantees for finite times.

A natural candidate is an upper bound on the *round-by-round output contraction ratio* $\varrho \leq 1$, i.e., for all executions and rounds $k \in \mathbb{N}^+$,

$$\delta(k) \leq \varrho \cdot \delta(k-1) \ .$$

[3]Not to confuse with the macro-rounds from the amortized algorithms.

with δ being the diameter of the process output values in round k.[4] We thus readily obtain round-by-round contraction ratio bounds for all our averaging algorithms, and, at the level of macro-rounds, for their amortized versions. A clear downside of this measure is the trivial bound 1 for algorithms that do not contract every round, such as the amortized algorithms at round-scale.

A natural alternative that works around this problem is the *(asymptotic) output contraction ratio* defined as the supremum over all executions of

$$\limsup_{k\to\infty} \sqrt[k]{\delta(k)} \ . \tag{4.8}$$

Algorithms with a round-by-round output contraction ratio of ϱ will also have an output contraction ratio of at most ϱ. Thus, the midpoint averaging algorithm has output contraction ratio $\frac{1}{2}$ in network models with non-split communication graphs. Its amortized variant has output contraction ratio $\sqrt[n-1]{\frac{1}{2}}$ in network models with rooted communication graphs — making a comparison possible.

However, this measure has two other problems as will be outlined in the following:

(a) A class of algorithms collapses to 0 that has different convergence behavior.

(b) It ranks algorithms superior to others that intuitively should be ranked less from an application point of view.

To show (a), consider the following algorithm: take one of the averaging algorithms $\mathcal{A}$ mentioned before that solves asymptotic consensus, e.g., the midpoint or the equal neighbor averaging algorithm. Assume that we restrict ourselves to inputs from $\{0,1\}$ only. Now construct an algorithm $\mathcal{B}$ from $\mathcal{A}$ that uses $\mathcal{A}$'s analysis to derive an upper bound $\hat{\delta}_{\mathcal{A}}(k)$ on the diameter of output values when executing algorithm $\mathcal{A}$, for every round $k \in \mathbb{N}$. This can be achieved for all averaging algorithms presented in this chapter. Algorithm $\mathcal{B}$ then runs $\mathcal{A}$ and computes $\hat{\delta}_{\mathcal{A}}(k)$ for the current round k. Denote by $s_{i,\mathcal{A}}$ the output of algorithm $\mathcal{A}$ run by process i. Analogously, $s_{i,\mathcal{B}}$ denotes the output of algorithm $\mathcal{B}$ run by process i. In each round k, process i sets

$$s_{i,\mathcal{B}}(k) \leftarrow \begin{cases} 0 & \text{if } s_{i,\mathcal{A}}(k) + \left[-\hat{\delta}_{\mathcal{A}}(k), \hat{\delta}_{\mathcal{A}}(k)\right] \cap [0,0.5] \neq \emptyset \\ 1 & \text{otherwise} \end{cases} .$$

In fact this algorithm solves asymptotic consensus for binary inputs (and outputs). We will not give a full proof here, but sketch the ideas:

- Convergence of a process' output follows from convergence of the output of algorithm $\mathcal{A}$ and the fact that the interval $I_i(k) = s_{i,\mathcal{A}}(k) + \left[-\hat{\delta}_{\mathcal{A}}(k), \hat{\delta}_{\mathcal{A}}(k)\right]$ converges to length 0.
- Validity follows from validity for algorithm $\mathcal{A}$.
- Finally, observe that the process outputs of algorithm $\mathcal{A}$ coverages to the same value $s^*_{\mathcal{A}}$ because of the agreement property of algorithm $\mathcal{A}$. If $s^*_{\mathcal{A}} \leq 0.5$, every process will always output 0 in $\mathcal{B}$ since their intervals necessarily contain $s^*_{\mathcal{A}}$ and thus intersect with $[0,0.5]$. Thus agreement is fulfilled for $\mathcal{B}$ in this case. Otherwise, every process will eventually output 1, since its interval will have non-zero distance from 0.5; again agreement for $\mathcal{B}$ is fulfilled in this case. Thus agreement holds for algorithm $\mathcal{B}$ in all cases.

For any such algorithm $\mathcal{B}$, the measure in (4.8) is 0 — irrespective of the underlying algorithm $\mathcal{A}$; showing point (a).

Shortcoming (b) is seen by comparing an algorithm $\mathcal{B}$ based on $\mathcal{A}$ to the original algorithm $\mathcal{A}$: While, $\mathcal{B}$ has measure 0 and $\mathcal{A}$ can be shown to have a positive measure, $\mathcal{A}$ may have significant advantages as its value may not drastically change in late rounds. Algorithm $\mathcal{B}$ can be shown to jump from 0 to 1 or vice versa arbitrarily late — otherwise, exact consensus would be achievable.

[4] Such ratios are sometimes called rates, including in own previous work. We will use the term ratio in this writeup, since small ratios mean fast contraction, which is counterintuitive with the notation of a rate.

Valency and valency contraction ratio. We thus used a measure that is an adaptation of (4.8): the *(asymptotic) valency contraction ratio* defined as the supremum over all executions of

$$\limsup_{k\to\infty} \sqrt[k]{\Delta(k)} \ , \tag{4.9}$$

where $\Delta(k)$ is a measure on the *reachable limits after round k* that will be defined in the following.

Let $\mathcal{A}$ be an algorithm that solves asymptotic consensus. Recall that a global state of $\mathcal{A}$ is a vector of local process states — we will refer to it as a configuration in the following using $C(k)$ rather than $s(k)$ to denote the *configuration at the beginning of round k*. Configurations occurring in executions with the initial configuration $C(0)$ and a communication sequences in network model $\mathcal{N}$ will be called *reachable from $C(0)$ by $\mathcal{A}$ in $\mathcal{N}$*.

Following the valency definition in Fischer *et al.* [144] for exact consensus algorithms, and a generalization of it algorithms solving asymptotic consensus we introduced in [134], let the *valency of a configuration C of $\mathcal{A}$ in $\mathcal{N}$*, be the set of limits of process outputs in executions by $\mathcal{A}$ in $\mathcal{N}$ that contain configuration C. This set is denoted by $Y^*_{\mathcal{N},\mathcal{A}}(C)$.

We are now in the position to define

$$\Delta_{\mathcal{N},\mathcal{A}}(C) = \operatorname{diam}\left(Y^*_{\mathcal{N},\mathcal{A}}(C)\right) \ . \tag{4.10}$$

For simplicity of notation, if we fix an algorithm $\mathcal{A}$, a network model $\mathcal{N}$, and an execution which induces the sequence of configurations $C_0, C_1, \ldots, C_k, \ldots$, we simply write $\Delta(k)$ for $\Delta_{\mathcal{N},\mathcal{A}}(C_k)$.

Intuitively, $\Delta_{\mathcal{N},\mathcal{A}}(C)$ is a measure on how much the process output is settled in configuration C: the larger Δ, the larger is the amount by which processes may change their outputs later on. As such it has great practical implications. Indeed Δ can be bounded along an execution for the algorithms that we will present in the following. While (4.9) does not allow to bound Δ over time, since it is a measure on the asymptotic performance for the reasons given before, we were able to show a shrinking factor of Δ per fixed number of rounds for the algorithms discussed in [119].

Bounds of valency contraction ratio. As mentioned before, the averaging algorithms and their amortized versions can be used to solve asymptotic consensus after removing the decision and termination rules.

From Theorem 103 and the observation that the valency $Y^*_{\mathcal{N},\mathcal{A}}(C)$ of a configuration C that occurs in the k^{th} round of an execution of an averaging algorithm is always a subset of the convex hull of the process outputs of round k — and thus $\Delta(k) \le \delta(k)$ — we obtain:

Theorem 112. *In a non-split network model of n processes, every averaging algorithm that is σ-safe, with $\sigma > 0$, achieves asymptotic consensus. For round $k \in \mathbb{N}^+$, it is*

$$\Delta(k) \le \delta(k) \le (1-\sigma)^k \cdot \delta(0) \ ,$$

and it has a valency contraction ratio of at most

$$1 - \sigma \ .$$

In particular, for the midpoint algorithm, $\Delta(k) \le \frac{1}{2^k} \cdot \delta(0)$ and it has a valency contraction ratio of at most $\frac{1}{2}$.

Likewise, in a rooted network model, an analogon to Theorem 110 on page 87 can be shown:

Theorem 113. *In a rooted network model with n processes, every amortized averaging algorithm that is σ-safe, with $\sigma > 0$, achieves asymptotic consensus. For round $k \in \mathbb{N}^+$, it is*

$$\Delta(k) \le \delta(k) \le (1-\sigma)^{\left\lfloor \frac{k}{n-1} \right\rfloor} \cdot \delta(0) \ ,$$

and it has a valency contraction ratio of at most

$$\sqrt[n-1]{1-\sigma} \ .$$

In particular, for the midpoint algorithm, $\Delta(k) \leq \left(\frac{1}{2}\right)^{\left\lfloor \frac{k}{n-1} \right\rfloor} \cdot \delta(0)$ *and it has a valency contraction ratio of at most* $\sqrt[n-1]{\frac{1}{2}}$.

4.6 Robustness to Limitations in Implementations

The averaging algorithms and their amortized variants have been analyzed with challenging communication networks in mind, but without considering limitations a practical implementation in software or hardware would have to face. In particular such limitations are:

1. Stored and transmitted values are currently assumed to be from $[0, 1]$. A real-world solution clearly has to deal with finite density of memory and thus quantized values.

2. While the fact that the exact number of processes n or an upper bond is known in advance, this may result in overly pessimistic parametrization, e.g., for building macro-rounds, during most of the rounds. Further, n may unpredictably change in several application.

Quantization. Addressing the first point, we studied Q-quantized versions of (amortized) averaging algorithms, with $Q \in \mathbb{N}^+$ in [117]. For such an algorithm, after a macro-round, we set

$$s_i(k) = [s'_i(k)]_Q \quad , \tag{4.11}$$

where s'_i is computed according to (4.2) and $[\cdot]_Q$ denotes rounding up (or rounding down; but doing this consistently) to the next multiple of $1/Q$.

We showed that quantization does not greatly influence the algorithm's time complexity in solving ε-consensus:

Theorem 114 ([117])**.** *In a network model with only rooted graphs, the amortized midpoint algorithm achieves:*

- $1/Q$*-consensus after*

$$(n-1)\left(\lceil \log_2 (Q-2) + 2 \rceil\right) \tag{4.12}$$

  rounds.

- ε*-consensus, with* $\varepsilon > 2/Q$*, after*

$$(n-1)\left(\left\lceil \log_2\left(\frac{Q-2}{\varepsilon Q - 2}\right) + 1 \right\rceil\right) \tag{4.13}$$

  rounds.

We cannot expect smaller ε than $1/Q$ with the property that a quantized version solves ε-consensus. If so, the same algorithm would also solve exact consensus; a contradiction to unsolvability of exact consensus in arbitrary rooted network models [123].

Underestimating the number of processes. Underestimating the process number n leads to the loss of the guarantee that macro-rounds are non-split. However, we showed that the time complexity gracefully degrades with how much this estimate is off:

Theorem 115 ([117]). *Given an amortized σ-safe algorithm with estimate $\hat{n}$ as the number of processes. In a system with n processes, the algorithm achieves ε-consensus in*

$$O\left(L\sigma^{-L}\log\left(\frac{1}{\varepsilon}\right)\right) \tag{4.14}$$

macro-rounds, where $L = \left\lceil \frac{n-1}{\hat{n}-1} \right\rceil$.

A similar statement holds for a scenario where $\hat{n}$ is a correct estimate of n, but rooted graphs are only guaranteed to occur with some frequency.

4.7 Multidimensional Consensus

Several of the discussed averaging algorithms and their amortized versions directly carry over to solve multidimensional approximate and asymptotic consensus. An example that falls into this class is the equal neighbor algorithm. Figure 4.5 shows an example execution prefix with output states in $\mathbb{R}^3$. Observe that the trace remains in the convex hull (in light red) of the three nodes' initial values that were chosen to be the three base vectors.

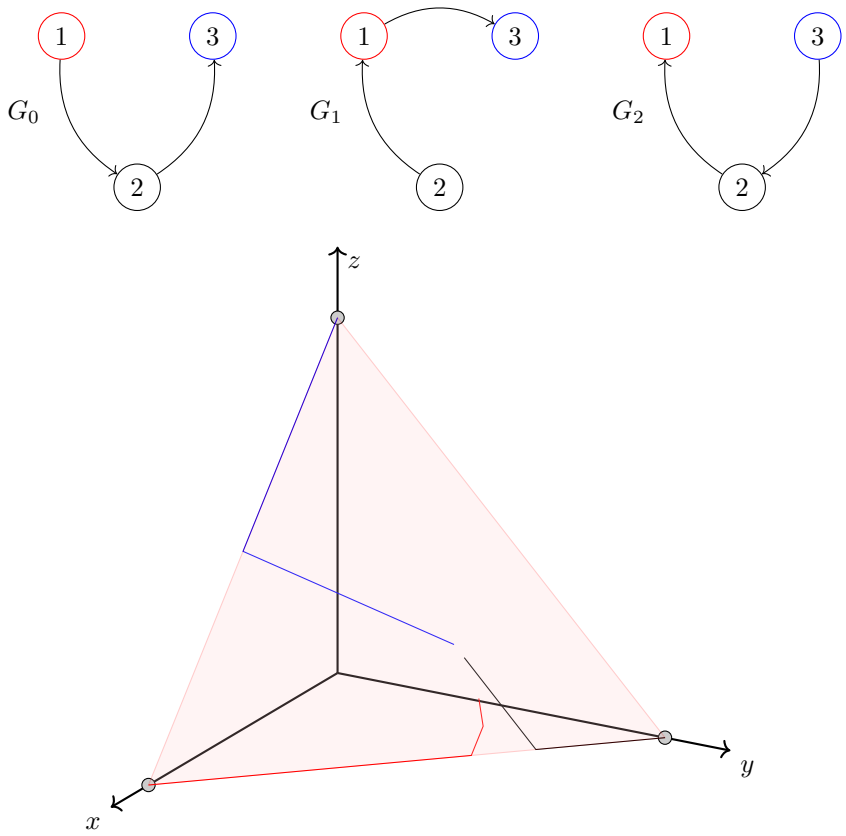


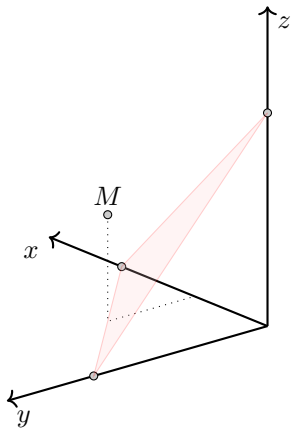
Figure 4.5: A system of $N = 3$ nodes with node states from dimension $d = 3$. The figures shows the outputs of the three nodes during a prefix of an execution of the equal neighborhood algorithm for $T = 5$ rounds. The communication graphs were chosen uniformly at random from the network model shown at the top of the figure.

Unfortunately, the midpoint algorithm that outperforms the equal neighbor algorithm for both problems is not even well-defined for dimensions $d > 1$: the notion of a minimum and a maximum of a finite subset of $\mathbb{R}^d$ is not defined.

4.7.1 Generalizing the Midpoint Algorithm

A first natural generalization of averaging algorithms, and thus also the midpoint algorithm, is to apply the averaging rules coordinate-wise. For dimension $d = 2$, one can show that this is indeed a valid approach for the midpoint algorithm. However, for dimensions $d \geq 3$, the updated value may lie outside of the convex hull of received values. The algorithm thus violates the validity condition of approximate and asymptotic consensus.[5] The following example shows a problematic scenario.

Example 116 (Midpoint algorithm applied component-wise)**.** *Consider the case of a 3-dimensional input and output space and three nodes with initial values* $(1, 0, 0)$*,* $(0, 1, 0)$*, and* $(0, 0, 1)$*. The plane spanned by the three points is* $x + y + z = 1$*. The component-wise midpoint of the three points is* $M = \frac{1}{2} \cdot (1, 1, 1)$*, which is seen not to fulfill the plane equation. See below for a visualization.*



In [120] and [121] we proposed different generalizations of the midpoint algorithm:

- *Centroid.* A natural algorithm is to update one's value to the centroid of the polyhedron that is the hull of all received values.
- *MidExtremes.* Choose two received values $a, b \in \mathbb{R}^d$ that realize the diameter of the received values. The new value is the midpoint on the line ab. Formally, for a node $i \in [n]$ and round $k \in \mathbb{N}^+$,

$$\begin{aligned} a, b &\leftarrow \operatorname{argmax}_{u,v \in \mathrm{In}_i(k)} ||x_u(k-1) - x_v(k-1)|| \\ x_i(k) &\leftarrow \frac{a+b}{2} \ . \end{aligned}$$

- *ApproachExtreme.* Choose a received value $a \in \mathbb{R}^d$ that has maximum distance to one's own value y. The new value is the midpoint on the line ay. Formally, for a node $i \in [n]$ and round $k \in \mathbb{N}^+$,

$$\begin{aligned} a &\leftarrow \operatorname{argmax}_{u \in \mathrm{In}_i(k)} ||x_u(k-1) - x_i(k-1)|| \\ x_i(k) &\leftarrow \frac{a + x_i(k-1)}{2} \ . \end{aligned}$$

  While technically this is not generalizing the midpoint algorithm in the sense that it is equivalent to it in dimension 1, it relies on a similar idea.

Terminating variants of these algorithms for approximate agreement and amortized versions for rooted network models are defined analogously to the midpoint algorithm in dimension 1.

Figure 4.6 shows outputs during an execution prefix of the MidExtremes algorithm in the same setting (with respect to initial values and network model) as Figure 4.5 on page 91 for the

[5] In fact the validity condition is a condition on the decided value or the limit and not on a single step. However, one can show that the output values may also lie outside of the convex hull of initial values after arbitrarily many steps and not just a single step.

equal neighbor algorithm. At the first sight, both seem to perform similarly. We will discuss the performance of the three proposed algorithms in the following though, showing that they have output contraction ratios independent of the number of nodes n in non-split network models; which is not the case for the equal neighbor algorithm.

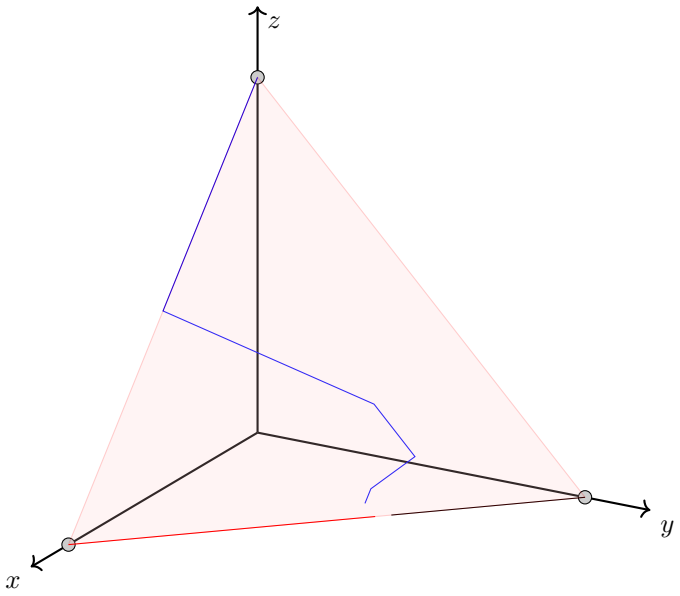


Figure 4.6: A system of $N = 3$ nodes with node states from dimension $d = 3$. The figures shows the outputs of the three nodes during a prefix of an execution of the MidExtremes algorithm for $T = 5$ rounds. The communication graphs were chosen uniformly at random from the network model shown in Figure 4.5.

Coordinate system. Before discussing contraction ratios, let us take a look at coordinate systems. When considering higher-dimensional input and output spaces, coordinate systems relative to which the node's values are determined, play a role.

While a designed cluster running distributed optimization and repeatedly calling an approximate consensus algorithm may very well resort to an a priori determined common coordinate system, a swarm of biological nodes may not. Likewise, moving autonomous vehicles may potentially not easily agree or have access to such a common coordinate system. Rather, it may be more practical to let each node have its own local coordinate systems within which it determines the positions of neighboring nodes.

Another property of algorithms is if process outputs depend on the initially chosen global or local coordinate systems. For example, outputs may change in coordinate-wise algorithms when the coordinate system is rotated. By contrast, other algorithms may guarantee that the outcomes are invariant under several transformations of the coordinate system like rotation and translation.

One observes that the Centroid algorithm, the MidExtremes algorithm, and the ApproachExtreme algorithm require only local coordinate systems and are invariant under rotation and translation.

Performance bounds. By geometric considerations of process outputs that stem from executions in non-split networks, we showed the following results.

In [120] we analyzed the performance of the Centroid algorithm in non-split network models. The result relies on a reshaping technique of the convex hull of received process values. The reshaped convex hull is shown to not change an a priori fixed coordinate of the Centroid.

Theorem 117 ([120]). *For a non-split network model and input values from an arbitrary dimension $d \geq 1$, the Centroid algorithm guarantees a round-by-round output convergence ratio of*

$$c(k) \leq \frac{d}{d+1}$$

for all rounds $k \geq 1$. In the particular case of $d = 1$, the algorithm is equivalent to the midpoint algorithm and achieves $c(k) \leq 1/2$.

By previously discussed techniques, bounds for rooted network models are also obtained for the Centroid and the amortized Centroid algorithm, as well as for their quantized variants.

In [121] we analyzed the two other previously discussed algorithms that rely on extreme values. For the MidExtremes algorithm in non-split networks we obtained:

Theorem 118 ([121]). *For a non-split network model and input values from an arbitrary dimension $d \geq 1$, the MidExtremes algorithm guarantees a round-by-round output convergence ratio of*

$$c(k) \leq \sqrt{7/8}$$

for all rounds $k \geq 1$. In the particular case of $d = 1$, the algorithm is equivalent to the midpoint algorithm with $c(k) \leq 1/2$.

An analogous result was obtained for the ApproachExtremes algorithm. The performance bound we proved is slightly worse than for the MidExtremes algorithm.

Theorem 119 ([121]). *For a non-split network model and input values from an arbitrary dimension $d \geq 1$, the ApproachExtremes algorithm guarantees a round-by-round output convergence ratio of*

$$c(k) \leq \sqrt{31/32}$$

for all rounds $k \geq 1$. In the particular case of $d = 1$, the algorithm achieves $c(k) \leq 3/4$.

There may, however, be plausible reasons to resort to the ApproachExtremes algorithm if one has the choice in designed systems. Besides benefits in computational complexity, it may simply be impossible or hard to estimate distances between all "visible", i.e., received, values, while it may be plausible to measure distances between oneself and remote nodes. This is particularly the case if values are not transmitted but reception simply means seeing another node and estimating its position with respect to oneself. For some naturally occurring biological systems like a flock of birds, this algorithm may also be more plausible to be the case.

Comparison of performance. Table 4.1 on page 95 summarizes the performance of our three algorithms [120, 121] and compares it to two previously existing algorithms for multidimensional approximate/asymptotic consensus: the Mendes–Herlihy (MH) and the Vaidya–Garg (VG) algorithms proposed in [122]. For the purpose of comparison with algorithms that do not necessarily contract outputs every round, we resort to output contraction ratios rather than the round-by-round output contraction ratio. However, for all except MH, the given performance measure is also a valid bound on the round-by-round contraction. The table also includes local computational complexity.

	MidExtremes	ApproachExtreme	Centroid	MH	VG
output contraction ratio	$\sqrt{\frac{7}{8}}$	$\sqrt{\frac{31}{32}}$	$\frac{d}{d+1}$	$\sqrt[d]{\frac{1}{2}}$	$1-\frac{1}{n}$
local `TIME`	$O(n^2 d)$	$O(nd)$	#P-hard	$O(nd)$	$O(nd)$
coordinate-free	yes	yes	yes	no	yes

Table 4.1: Comparison of four algorithms for asymptotic and approximate consensus in non-split network models. From [121].

One observes that the VG algorithm's performance depends on the number of nodes n, while the other algorithm's performance does not. A dependency on the dimension d is observed for MH and Centroid. Notably, the MidExtremes and ApproachExtreme algorithm's performance depends on neither.

4.7.2 Implications on Algorithms for Static Networks

The three presented algorithms are correct for non-split network models, and thus for rooted network models, by previously discussed techniques. Faster, amortized versions for rooted network models are obtained analogously to previously discussed algorithms. Further, the algorithms may as well be used in classical, static systems with failures.

One such classical network is a fully connected network with asynchronous message passing and up to f Byzantine nodes. For example, the VG and MH algorithms have been designed for this setting [122], given that $n > (d+2)f$. For $d = 1$ this gives the requirement of $n > 3f$ shown by Fischer *et al.* [145] to be necessary to solve approximate consensus. While Mendes *et al.* [122] focused on approximate consensus, similar considerations as discussed in the following hold for asymptotic consensus. Processes proceed in asynchronous rounds by waiting for $n - f$ messages from the current round before proceeding to the next round; a common technique in designing algorithms for asynchronous systems. The idea by Mendes *et al.* [122] was to use a safe area calculation in each round. They compute a subspace from received values that lies within the convex hull of values received from correct nodes, and that is shown to pairwise intersect if computed by correct nodes.

Plugging in our algorithms instead of the VG or MH approximate consensus algorithms, but using the same safe area calculation, we are able to achieve ε-consensus within $O\big(\log \frac{\Delta}{\varepsilon}\big)$ rounds, where Δ is the initial diameter of correct node values [121]. By contrast, the MH algorithm, e.g., has a worst-case round complexity of $\Omega\big(d \log \frac{d\Delta}{\varepsilon}\big)$ that depends on the dimension d.

4.8 Lower Bounds on Valency Contraction Ratios and Time Complexity

The previous sections discussed several algorithms solving asymptotic consensus and upper bounds on their output contraction ratios within several network models. A natural question is if these are optimal, i.e., if no algorithm can do better. This would be particularly encouraging since the proposed algorithms are simple and do not require, or require only little, state to be stored besides the current output value. For comparison, in general, an algorithm can be full-information, i.e., compute upon and broadcast all its history in each round. For example, processes may use higher order filters that overshoot and set their outputs outside the convex hull of received values. In [119] we addressed this question of lower bounds for general algorithms with the results summarized in the following.

Related work. Only a small body of literature addressed lower bounds of valency/output contraction ratios in dynamic networks. Cao *et al.* [146] showed lower bounds on so called scrambling constants of stochastic matrices which are weighted versions of the adjacency matrices of the communication graphs. However, such bounds on scrambling constants do not directly imply bounds on valency/output contraction ratios.

By contrast, lower bounds for static classical settings have received attention in distributed computing. For example, Dolev *et al.* [81] posed the question of optimal contraction ratios in fully-connected asynchronous[6] message passing systems with f Byzantine failures. Abraham *et al.* [139] improved on the algorithm in terms of required resilience. Dolev *et al.* [81] proved a lower bound on the round-by-round valency contraction ratio for a class of algorithms within this setting. Fekete [140, 147] proved asymptoticly tight valency contraction ratios along execution prefixes for both the synchronous case (including crashes and Byzantine faults) and the asynchronous case (crashes with $n > 2f$).

Related is also the work on lower bounds of time complexity of approximate consensus. For dynamic networks, Hoest and Shavit [148] proved lower bounds on the time complexity of approximate consensus algorithms in the non-uniform iterated immediate snapshot (NIIS) model; a shared memory communication model which they introduced in this work.

4.8.1 Valency and Output Contraction Ratios

Recall that for a configuration C that appears in an execution of an algorithm solving exact, approximate, or asymptotic consensus, we write $\delta(C)$ for $\operatorname{diam}(x)$, where x is the vector of process outputs. For averaging algorithms we showed [119] that upper bounds on output contraction ratios lead to upper bounds on valency contraction ratios.

Lemma 120. *Let C be a configuration of averaging algorithm $\mathcal{A}$ that solves asymptotic consensus in network model $\mathcal{N}$. Then*

$$\Delta_{\mathcal{N},\mathcal{A}}(C) \leq \delta(C) \ .$$

In particular, for an averaging algorithm that guarantees contraction of output values by a factor $c < 1$ every $k \in \mathbb{N}^+$ rounds, the valency contraction ratio is at most $\sqrt[k]{c}$.

4.8.2 Valency Contraction Ratios in Dynamic Networks

In [119], we established (asymptotically) tight lower bounds for several network models, showing (asymptotic) optimality of some of the previously discussed asymptotic consensus algorithms. Key to showing optimality is Lemma 120 that is used to transfer results about output contraction to upper bounds on valency contraction ratios.

While all lower bounds hold for arbitrary dimensions, tightness was only shown for dimension 1. The question about asymptotically tight valency contraction ratios for higher dimensions thus remains an interesting open problem to date.

Two nodes. We start with the simplest of all network models in which exact consensus is not solvable, but in which asymptotic consensus is solvable. The graphs of this rooted network model for $n = 2$ nodes are shown in Figure 4.7.

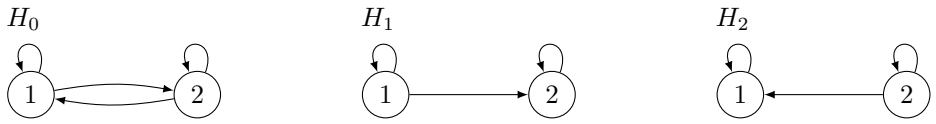


Figure 4.7: The rooted communication graphs H_0, H_1, and H_2 for $n = 2$. Self-loops are shown. From [119].

For this network model, we showed a lower bound of 1/3:

[6]The paper also studied synchronous systems. Since for these exact consensus is solvable if $n > 3f$, the valency contraction ratio is 0. An interesting question, however, is lower bounds for round-by-round valency contraction ratios.

Theorem 121 ([119]). *The valency contraction ratio of any asymptotic consensus algorithm for $n = 2$ nodes in the network model $\{H_1, H_2, H_3\}$ is at least* $1/3$.

The bound is matched by the output contraction ratio of $1/3$ for the jumping algorithm [117] discussed in Section 4.5.1. Applying Lemma 120, we obtain that the valency contraction ratio is tight.

Nodes with no incoming messages. Choose an arbitrary communication graph G with n nodes and let the set nonlistening(G) be the set of n communication graphs obtained from G by removing the incoming links to a node i, except its self-loop, for each node i. In each such graph, node i does not receive messages. Figure 4.8 shows two examples for $n = 3$ nodes.

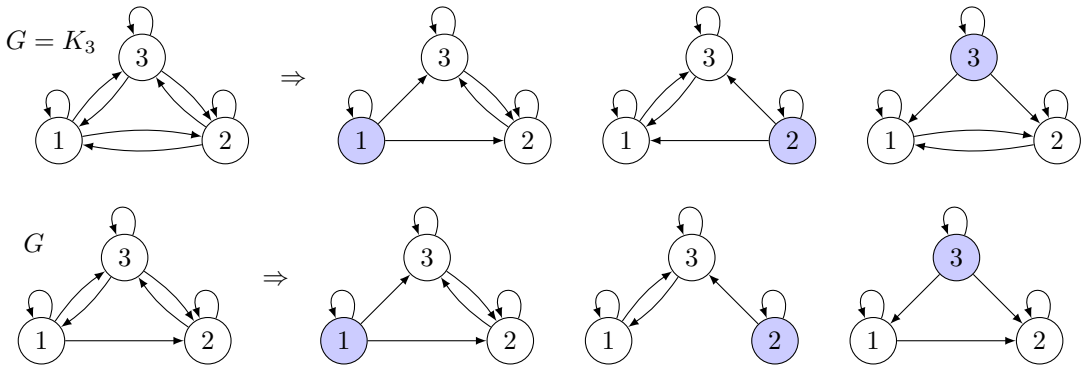


Figure 4.8: Two examples for graphs G and the corresponding set of graphs in nonlistening(G). All shown graphs are rooted; which is not necessarily the case for all G. Self-loops are shown.

We proved the following result for network models that are supersets of nonlistening(G), for arbitrary communication graphs G:

Theorem 122 ([119]). *Let G be a communication graph with $n \geq 3$ nodes The valency contraction ratio of any asymptotic consensus algorithm in a network model that includes all graphs from* nonlistening(G) *is at least* $1/2$.

First observe that the theorem holds for all G, including ones that are not rooted, or whose set nonlistening(G) contains non-rooted communication graphs. While the theorem's bound is also true for these, the valency contraction ratio is 1 in this case, since asymptotic consensus cannot be solved for non-rooted network models [116]; see also Theorem 102 on page 84.

Further, setting $G = K_n$, i.e., to the complete graph with n nodes, Theorem 122 also shows that the network model of all non-split communication graphs has a valency contraction ratio of at least $1/2$, since this network model is a superset of nonlistening(K_n). The lower bound of $1/2$ is met for this network model by the midpoint algorithm; see Theorem 112 on page 89[7] and the application of Lemma 120 on page 96.

Rooted network models. Both previous lower bounds are derived by round-by-round considerations on the valency. While these bounds are met by algorithms for non-split network models, (amortized) algorithms for rooted network models provide only guaranteed contraction per $n - 1$ rounds. In search of matching lower bounds for such algorithms, we thus studied valency contraction along communication sequences instead of single rounds. In [119] we showed that with this strategy one can show that the midpoint algorithm with a contraction of $\sqrt[n-1]{1/2}$ in rooted network

[7] Precisely, the round-by-round output contraction ratio bound is used.

models is asymptotically optimal. In particular, we showed this bound for a network model that comprises the following graphs Ψ_1 to Ψ_3: For $i \in [3]$ we construct communication graph Ψ_i by

- nodes 4 to n forming a path with edges from $j \in \{4, \dots, n-1\}$ to $j+1$,
- nodes $[3] \setminus \{i\}$ having n as their in-neighbor and node 4 as their out-neighbor, and
- node i having node 4 as its out-neighbor.

The graphs for $n = 6$ are shown in Figure 4.9.

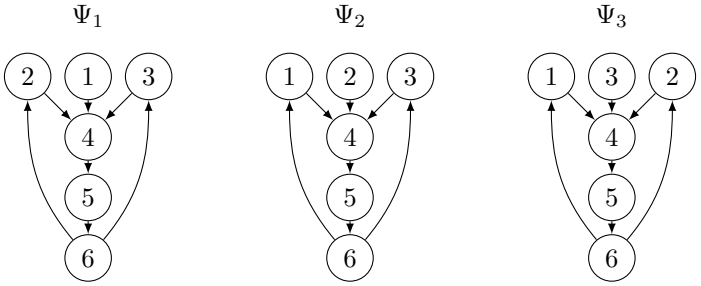


Figure 4.9: Rooted communication graph Ψ_1, Ψ_2, and Ψ_3 for $n = 6$. Self-loops are omitted.

We then proved that:

Theorem 123 ([119]). *The contraction ratio of any asymptotic consensus algorithm in an oblivious message adversary that includes the graphs Ψ_1, Ψ_2, and Ψ_3 is greater or equal to $\sqrt[n-2]{1/2}$.*

Network models with α-diameter. An interesting approach of a characterization of network models in which exact consensus is solvable was presented by Coulouma *et al.* [126]. At its heart is the α-relation, defined as follows.

Definition 124 ([126]). *Let $\mathcal{N}$ be a network model. Let $G, H, K \in \mathcal{N}$.*

$$G \alpha_K H \ :\Leftrightarrow\ \mathrm{In}_{R(K)}(G) = \mathrm{In}_{R(K)}(H) \ .$$

Further, $G \alpha_{\mathcal{N}} H$ is the smallest relation for which

$$G \alpha_K H \ :\Leftarrow\ \exists K \in \mathcal{N} : \mathrm{In}_{R(K)}(G) = \mathrm{In}_{R(K)}(H) \ ,$$

and that is closed transitively.

Based on this definition, we introduced the *α-diameter* of a network model $\mathcal{N}$ in [119]:

Definition 125 ([119]). *Let $\mathcal{N}$ be a network model. We say that $\mathcal{N}$ is* $\alpha_{\mathcal{N}}$-connected *if it has a single $\alpha^*_{\mathcal{N}}$-equivalence class. Otherwise, it is $\alpha_{\mathcal{N}}$-disconnected.*

Let $\mathcal{N}$ be a network model that is $\alpha_{\mathcal{N}}$-connected. The $\alpha_{\mathcal{N}}$-diameter *of $\mathcal{N}$ is the smallest $D \geq 1$ such that for all $G, H \in \mathcal{N}$ there exist communication graphs $H_0, \dots, H_q \in \mathcal{G}$ and $K_1, \dots, K_q \in \mathcal{G}$ with $q \leq D$ such that $G = H_0$, $H = H_q$, and $H_{r-1} \alpha_{K_r} H_r$ for all $r \in [q]$.*

With this definition in place, we were able to prove that even if a network model does not fall into the classes previously discussed, its *α-diameter* provides a lower bound on the valency contraction ratio:

Theorem 126 ([119]). *Let $\mathcal{N}$ be a network model for which exact consensus is not solvable. If $\mathcal{N}$ is $\alpha_{\mathcal{N}}$-connected, then the valency contraction ratio of any asymptotic consensus algorithm in $\mathcal{N}$ is at least $1/(D+1)$ where D is the $\alpha_{\mathcal{N}}$-diameter of $\mathcal{N}$.*

We discuss an application of this bound in the next section.

4.8.3 Valency Contraction Ratios in Static Networks with Failures

A classically considered static network is the fully connected network with processes communication by asynchronous message-passing. Given that there are $n \in \mathbb{N}^+$ processes, we allow for up to $f < n/2$ crashes.

A common technique to design algorithms for such systems is to construct communication-closed rounds [80]: each process broadcasts its current round's message that contains its current state, and waits until it has received at least $n - f$ messages corresponding to this round. Upon doing so, it updates its state based on the received states and proceeds to the next round. Old messages from previous rounds are simply dropped, making the round communication-closed.

Each such round may be viewed as a communication graph with $n - f$ in-neighbors at each node corresponding to a correct process. Establishing a bound on the α-diameter in such systems, and applying Theorem 126 yields that no algorithm operating in asynchronous rounds can achieve a valency contraction ratio less than $\frac{1}{\lceil n/f \rceil + 1}$ [119].

The bound asymptotically matches the valency contraction ratio established by an algorithm by Fekete [147] for system with $f < n/2$ crashes. Further, the bound asymptotically matches the valency contraction ratio for asynchronous algorithms for systems of size $n > 5f$ with up to f Byzantine failures by Dolev *et al.* [81], for which the same lower bound of $\frac{1}{\lceil n/f \rceil + 1}$ can be shown to hold with the discussed technique.

4.8.4 Implications for Approximate Consensus

All presented lower bounds have been derived for asymptotic consensus. We showed in [119] that lower bounds on the time complexity of approximate consensus algorithms can be deduced from these bounds for the same network models. The proof relies on a reduction of asymptotic consensus to approximate consensus.

4.9 Averaging Algorithms in Physical Systems

We finish our discussion of asymptotic consensus algorithms with an application. The simplicity of the averaging algorithms that solve asymptotic consensus may suggest their use in sensor nodes, mobile robots, or nodes in a cluster with potentially faulty links and online reconfiguration. The following example shows that they may be also used to analyze existing physical systems.

Consider a certain chemical substance dispersed within an incompressible medium[8] over a one-dimensional space. Denote with $c(x,t)$, with unit $[1/m^3]$, the concentration of the substance within the medium at location $x \in \mathbb{R}$ and time $t \in \mathbb{R}_0^+$. Further, $J(x,t)$ is the flux at position x and time t. Its unit is $[1/m^3 \cdot m/s] = [1/(m^2 s)]$. The generator function $g(x,t)$ describes the generation (if positive) or consumption rate (if negative) of the substance at position x and time t. Its unit is $[(1/m^3)/s] = [1/(m^3 s)]$. We use the common notation $\nabla = \frac{\partial}{\partial x}$; in our case for dimension one.

The continuity equation relates the above quantities by

$$\frac{\partial}{\partial t} c(x,t) + \nabla \cdot J(x,t) = g(x,t) \ . \tag{4.15}$$

Now let us assume that there is an inflow of the medium at position $x = 1$, and an outflow at position $x = 0$. The inflow, and because of incompressibility of the medium, also the outflow, is

[8] The concentration of the substance within the incompressible medium, i.e., the density of the substance, however, can vary.

assumed to have constant velocity $v \geq 0$ over space. In particular, it holds that $\nabla v = 0$. Further, the concentration of the substance in the inflow is 0 and the substance is neither generated nor consumed at any position x. We thus have,

$$g(x,t) = 0 \tag{4.16}$$

$$J(x,t) = -D\nabla \cdot c(x,t) + v \cdot c(x,t) \ , \tag{4.17}$$

where $D > 0$, with unit $[m^2/s]$, is the diffusion coefficient, assumed to be constant over space and time. In particular, we will use that $\nabla D = 0$. The first term in (4.17) accounts for flux by diffusion, the second for flux by drift, also referred to as convection. Using, $\nabla v = 0$ and $\nabla D = 0$, we get

$$\frac{\partial}{\partial t} c(x,t) - D\nabla^2 \cdot c(x,t) + v \cdot \nabla c(x,t) = g(x,t) \ . \tag{4.18}$$

Let us next discretize (4.18) in space and time. For discretization in space, we uniformly quantize the sample space with discretization granularity $\delta x > 0$. Setting $\hat{c}(i,t) = c(i\delta x, t)$, for $i \in \mathbb{Z}$, we then use the facts that,

$$\begin{aligned} \nabla c(x,t) &= \lim_{\delta x \to 0} \frac{1}{2} \left(\frac{\hat{c}(i+1,t) - \hat{c}(i,t)}{\delta x} + \frac{\hat{c}(i,t) - \hat{c}(i-1,t)}{\delta x} \right) \\ &= \lim_{\delta x \to 0} \frac{1}{2\delta x} \left(\hat{c}(i+1,t) - \hat{c}(i-1,t) \right) \end{aligned} \tag{4.19}$$

$$\begin{aligned} \nabla^2 c(x,t) &= \lim_{\delta x \to 0} \frac{1}{\delta x} \left(\frac{\hat{c}(i+1,t) - \hat{c}(i,t)}{\delta x} - \frac{\hat{c}(i,t) - \hat{c}(i-1,t)}{\delta x} \right) \\ &= \lim_{\delta x \to 0} \frac{1}{(\delta x)^2} \left(\hat{c}(i+1,t) - 2\hat{c}(i,t) + \hat{c}(i-1,t) \right) \ . \end{aligned} \tag{4.20}$$

Clearly alternative discretization Ansätze could have been used. For example instead of the seemingly overcomplicated (4.19), simple forward or backward Euler formulas would have also fulfilled the required limit equations. The chosen discrete approximations have the benefit that they are symmetric in space, which will be crucial when analyzing the system in terms of dynamic communication graphs.

Combining (4.15), (4.17), (4.19), and (4.20), we readily obtain

$$\begin{aligned} \lim_{\delta t \to 0} \frac{\hat{c}(i,t+1) - \hat{c}(i,t)}{\delta t} = \\ \lim_{\delta x \to 0} \frac{1}{(\delta x)^2} \left(\hat{c}(i+1,t) - 2\hat{c}(i,t) + \hat{c}(i-1,t) \right) - \lim_{\delta x \to 0} \frac{v}{2\delta x} \left(\hat{c}(i+1,t) - \hat{c}(i-1,t) \right) \ , \end{aligned}$$

where we used forward Euler discretization in time. From this we get the discrete time and space recurrence,

$$\begin{aligned} \hat{c}(i,t+1) = & \hat{c}(i,t) \left(1 - \frac{2}{\delta t (\delta x)^2} \right) + \\ & \hat{c}(i-1,t) \left(\frac{1}{\delta t (\delta x)^2} + \frac{v}{2\delta t \delta x} \right) + \\ & \hat{c}(i+1,t) \left(\frac{1}{\delta t (\delta x)^2} - \frac{v}{2\delta t \delta x} \right) \ . \end{aligned}$$

Looking at this recurrence, we observe that:

1. The coefficients of $\hat{c}$ for certain parameter ranges of $\delta x > 0$, $\delta t > 0$, and $v \in \mathbb{R}$ are within $[0,1]$. In the following we will assume that the parameters are chosen such that this is the case. If not, one observes that they can be made to do so by decreasing δx and δt — this corresponds to a finer discretization of the differential equation.

2. The coefficients sum up to 1.

3. Viewing $\hat{c}(i, \cdot)$ as the output of node i over time, the static communication graph is a line graph with nodes being bidirectionally connected to neighboring nodes on the line.

The above recurrence can thus be viewed as all nodes i on the line graph executing an averaging algorithm. The results previously discussed on averaging algorithms thus apply and in particular may be used to derive time bounds on when concentrations have sufficiently well equilibrated across the liner space.

This view even carries over to the more interesting case of dynamic networks: assume instead of a constant v, a time dependent $v(t)$. If so, weights within the averaging algorithm will change over time. This may result in some coefficients becoming (almost) 0, in which case we can remove these links from the communication network. The resulting network is not anymore bidirectional, and in fact may change over time as $v(t)$ changes. Similar techniques may be used in higher dimensional spaces, with different and changing flows. Again previously discussed results on averaging algorithms apply.

Further reading. In [149] we applied averaging algorithms to solve clock synchronization in network were the nodes to be synchronized reside within a dynamic network structure. Example of such networks are networks of sensor nodes in harsh environments, vehicles, or drones.

Chapter 5

Outlook

This chapter discusses results from the following research articles.

- [150] Matthias Függer, Manish Kushwaha, Thomas Nowak. *Digital circuit design for biological and silicon computers.* In Vijai Singh, editor, Advances in Synthetic Biology. Springer, Heidelberg, March 2020.

  The work compares digital circuits in computers made from silicon and those in synthetic biology aimed for implementation within microbiological entities like bacteria.

- [151] Da-Jung Cho, Matthias Függer, Corbin Hopper, Manish Kushwaha, Thomas Nowak, Quentin Soubeyran. *Distributed computation with continual population growth.* In Hagit Attiya, editor, 34th International Symposium on Distributed Computing (DISC), volume 179 of Leibniz International Proceedings in Informatics (LIPIcs), pages 7:1-7:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

  Within a chemical reaction network (CRN) inspired by a growing population of two types of bacteria we analyze a simple protocol that we show to solve majority consensus with high probability if the initial gap between the number of the bacteria that make up the majority and the other bacteria is large enough. We also show that this happens in constant expected time and how this protocol can be used to implement Boolean gates.

5.1 Circuits in Microbiological Entities

So far, when we were discussing circuits, we made the silent assumption that they are intended for implementation in silicon; typically within the CMOS (Complementary Metal Oxide Semiconductor) process, which is the dominating design process for digital circuits at date. However, circuits are not restricted to gates in silicon. The by far larger body of existing circuits is discovered within viruses, cells, and among cells. These circuits sense environmental states, adapt internal states accordingly, and trigger externally observable actions. In the past two decades, additionally to naturally evolved circuits, engineered/synthetic circuits have been designed and implemented [152]. While the focus in engineered circuits is on digital circuits with Boolean gates, naturally evolved circuits are typically mixed-signal: signal levels are controlled with feedback loops and adapt their set points over time to sense over large input ranges [153]. The interested reader is referred to [153, 154] for an overview on such microbiological circuits. Teo *et al.* [155] discuss natural and synthetic circuits from an analog circuit perspective. An introduction to synthetic circuits is given in [156, 157].

In [150] we discussed commonalities and differences of such circuits and circuits in silicon. We summarize some of these in the following.

5.1.1 Digital circuits

Like our initial assumption of silicon as the target material, we also focused on digital circuits in this writeup, motivated by their great success in numerous applications.

The concept of a digital circuit as opposed to an analog circuit allows the designer to implement noise-tolerant input-output behavior with manageable complexity for the design. Thereby tolerance to noise is achieved by partitioning the value domain into a clear logical 0, a clear logical 1, and an intermediate domain. For the computation the exact analog input and output values do not play a role. As long as they are (and with noise added, stay) within the respective 0 and 1 domains, the outputs also stay in their respective 0 and 1 domains. A priori, the circuit may behave arbitrarily, if this assumption is violated, e.g., with the inputs being in the intermediate domain. However, typically, further assumptions on the gate's behavior are made, e.g., that a single (fast enough) transition at the inputs of a certain gate produces a single transition at the output. In Chapter 3 we discussed circuit behavior under even weaker assumptions by considering operation in case of metastable, i.e., arbitrary and unstable, analog values.

To guarantee that noisy inputs which stay in their logical domains lead to outputs that stay in their logical domain, amplification (away from the intermediate domain) and saturation (to ensure tightly bounded logical domains) are used; see Figure 5.1.

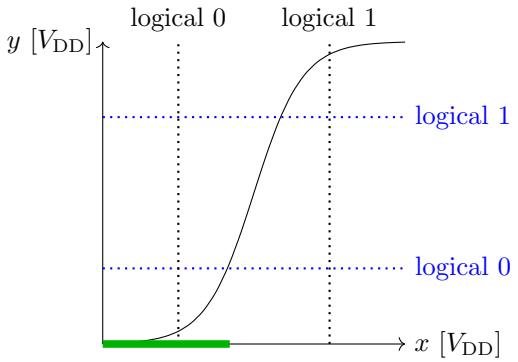


Figure 5.1: Amplification and saturation in digital gates. Here an identity gate is shown with input x and output y. The green input range that is significantly larger than the logical 0 domain is compressed into the logical 0 domain at the output.

Tightly bounded domains via saturation also ensure (i) fast transitions of outputs upon input changes and (ii) simple composition of gates with matching domains.

5.1.2 Specifying Circuits

Widely used hardware description languages are Verilog and VHDL. While similar to software-targeted high-level languages in several aspects, main differences to software-targeted languages are (i) a high degree of inherent parallelism: in hardware each gate operates in parallel to each other gate, and (ii) dedicated means to specify structural composition of modules. We will briefly describe a third language, Communicating hardware processes (CHP) [158] that we believe to be of particular interest within synthetic biology.

Synthetic biology has not yet decided on a common specification languages. While subsets of Verilog have been used [152], designs are typically specified in SBML and stochastic semantics and ODE semantics of these are used for simulations. At the heart of both semantics are chemical reaction networks (CRNs) that we will discuss in this chapter later on.

Communicating hardware processes. The language of communicating sequential processes (CSP) by Hoare [159] greatly shaped the landscape of parallel process semantics until today. Martin [158] adapted this software-centered undertaking to circuits in hardware. In [150] we discussed a simplified/restricted class of CHP as a language to specify circuits in silicon. However,

we will see that this language is also well suited to describe synthetic biological circuits at a higher level of abstraction than CRNs, while still remaining close to CRNs.

Following [158] and the presentation in [150], a (simplified) CHP circuit is a term among:

- A *variable assignment*. Let x be a variable from some alphabet and b be a Boolean expression on 0, 1, and variable names. A variable assignment is of the form

```
x := b
```

- A *guard*. Let b be a Boolean expression. A guard is of the form

```
[b]
```

- A *sequential composition* of two terms. Let a and b be two terms. A sequential composition of these is of the form

```
a; b
```

- A *parallel composition* of two terms. Let a and b be two terms. A parallel composition of these is of the form

```
a || b
```

- A *repeat execution* of a term a that is of the form

```
*[[ a ]]
```

Further, a circuit comprises of an initial Boolean assignment of all used variables.

An execution of a circuit is a timed sequence of variable assignments, similarly to the circuit execution from Chapter 1, where:

- A variable assignment leads to the specified assignment (after some time).
- A guard leads to this term waiting until the guard becomes true.
- Sequential and parallel composition are sequential and aprallel execution of the respective terms.
- A repeat execution is the repeated execution of this term.

Similarly as discussed in Chapter 2, the timing of such executions is specified via delay models. While Martin considered asynchronous executions for CHP in [158], more constrained delay models may be used instead.

Example 127 (Simple circuit in CHP)**.** *Let $\mathcal{C}$ be the circuit with variables A, B, and C, all initially set to* 0*, and the CHP term*

```
*[[ [A=0 and B=0]; (A:=1 || C:=1) ]]
```

Depending on the delay model, this circuit may have an unbounded number of executions, or a single (deterministic) execution. The circuit waits until $A = B = 0$, which is initially the case and then sets A and C to 1*. It then repeats to execute the term, forever waiting on the guard, since $A = 1$.*

Production rules are CHP terms of the form

```
*[[ [b]; A:=v ]]
```

with b being a Boolean expression and $v \in \mathbb{B}$. They are abbreviated as $b \to A\uparrow$ in case $v = 1$ and $b \to A\downarrow$ in case $v = 0$. Production rules are of interest as they can be easily mapped to gates and wires (given that they follow certain restrictions). For example the production rules $A \wedge B \to C\downarrow$ and $\neg(A \wedge B) \to C\uparrow$ make up a NAND-gate.

Martin [1, 2] studied the automated translation of CHP circuits into ones that only comprise of production rules, and that behave equivalently.

Chemical reaction networks. A chemical reaction network describes the evolution of species counts over time. Updates to species counts follow (probabilistic) rules specified via reactions with reactants on one side and products on the other side. Each reaction further has a kinetics assigned to it that determines the rate of the reaction. Typically the mass action kinetics is used, where the rate is fully determined by a rate constant.

While CRNs have received some attention in the context of distributed computing, see, e.g. [160, 161], the more restricted population protocols [162, 163] have received more attention in distributed computing. Examples of problems studied are function computation [162] and (self-stabilizing) leader election [164, 165] among others. Population protocols restrict reactions to have exactly two reactants and two products and assume a fairness condition on events motivated by a stochastic interpretation rather than rates. Several variants of population protocols have been proposed [163]; some models allowing for infinitely large species sets (where agents have non-constant state). An interesting variant that partially reintroduces timing into population protocols is by Beauquier *et al.* [166].

Following a standard stochastic formalization based on the collisions between species [167] that we also used in [151], CRNs are defined by:

- A set $\mathcal{S}$ of *species* as well as an initial count $I : \mathcal{S} \to \mathbb{N}$ for each species.
- A set of *reactions* upon the species. Thereby a reaction is triple $(\mathbf{r}, \mathbf{p}, \alpha)$ where $\mathbf{r}, \mathbf{p} \in \mathcal{S} \to \mathbb{N}$ and the reaction's *rate constant* $\alpha \in \mathbb{R}_0^+$. A *reactant* is a species with non-zero count in $\mathbf{r}$, a *product* a species with non-zero count in $\mathbf{p}$. For ease of notation, one writes $\mathbf{r} \xrightarrow{\alpha} \mathbf{p}$ instead of $(\mathbf{r}, \mathbf{p}, \alpha)$. Typically only non-zero entries of $\mathbf{r}$ and $\mathbf{p}$ are written as weighted sums.

Executions are sequences of configurations that are obtained by successively applying applicable reactions to the initial configuration given by the initial count I:

- A *configuration* of a CRN is an element of $\mathcal{S} \to \mathbb{N}$. As a unit we will use counts within a fixed volume $v > 0$. The fixed volume has unit $[l]$, and the counts are unitless numbers. Equivalently, species counts will sometimes be given in terms of mol (within the fixed volume). For a species S, we use $S = \mathcal{S}(S)$ to denote this count, and $[S]$[1] as its molar concentration in $[\text{mol}/l] = [M]$.
- A reaction $\mathbf{r} \xrightarrow{\alpha} \mathbf{p}$ is *applicable* to configuration $\mathbf{c}$ if $\mathbf{r}(S) \leq \mathbf{c}(S)$ for all $S \in \mathcal{S}$.
- The *propensity of reaction* $\mathbf{r} \xrightarrow{\alpha} \mathbf{p}$ in configuration $\mathbf{c}$ is equal to

$$\alpha \prod_{S \in \mathcal{S}} \binom{\mathbf{c}(S)}{\mathbf{r}(S)} , \tag{5.1}$$

  where $\binom{\mathbf{c}(S)}{\mathbf{r}(S)}$ is the binomial coefficient of $\mathbf{c}(S)$ and $\mathbf{r}(S)$. In particular, the binomial coefficient is 1 if $\mathbf{r}(S) = 0$, and 0 if $\mathbf{r}(S) > \mathbf{c}(S)$.

  Following the above specification, the unit $[\alpha]$ depends on the number of reactants. For reactions with a single reactant, it is $[\text{mol}^{-1}s^{-1}]$ for S in mol and $[s^{-1}]$ for unitless S, resulting in the propensity having unit $[s^{-1}]$. For reactions with two reactants, it is $[\text{mol}^{-2}s^{-1}]$ respectively $[s^{-1}]$, again resulting in unit $[s^{-1}]$ for propensity.

  This deviates from notations for ODE semantics where the additive contribution to the rate of change $d[P]/dt$[2] by a reaction $\mathbf{r} \xrightarrow{\alpha'} \mathbf{p}$ for product P is given as

$$\alpha' \mathbf{p}(P) \prod_{S \in \mathcal{S}} \mathbf{c}(S)^{\mathbf{r}(S)} \tag{5.2}$$

  instead of (5.1). An analogous additive negative contribution holds if P appears as a reactant in a reaction. Since (5.2) describes the change of $[P]$ over time, the unit of the term is $[M/s]$. In this case, for reactions with a single reactant, $[\alpha'] = [s^{-1}]$ and for two reactants, $[\alpha'] = [s^{-1}M^{-1}]$. We make use of the stochastic semantics of rates in the following.

[1] Not to be confused with the units of S.
[2] Analogous considerations hold for dP/dt.

- Applying reaction $\mathbf{r} \xrightarrow{\alpha} \mathbf{p}$ to configuration $\mathbf{c}$, results in configuration $\mathbf{c}' = \mathbf{c} - \mathbf{r} + \mathbf{p}$.
- The *stochastic kinetics* of a CRN is a continuous-time Markov chain whose states are the configurations, see, e.g., [168]. For any two states x and y, let the transition rate $Q(x, y)$ be the sum of propensities of reactions in configuration x that result in configuration y if being applied to x.

Thus, in the stochastic model, species counts within an a priori fixed volume change over time, with the time until a reaction takes place being an exponential random variable with rate equal to the reaction's propensity [167]. The following states propensities for some example reactions and configurations:

Example 128 (CRN)**.** *Consider the reaction $A + B \xrightarrow{\alpha} A$ and configuration $\boldsymbol{c}$ with $\boldsymbol{c}(A) = 10$ and $\boldsymbol{c}(B) = 5$. The propensity of this reaction in configuration $\boldsymbol{c}$ is $\alpha \cdot \boldsymbol{c}(A) \cdot \boldsymbol{c}(B)$. The reaction's application results in the configuration with $\boldsymbol{c}(A) = 10$ and $\boldsymbol{c}(B) = 4$.*

The propensity of $A \xrightarrow{\gamma} \emptyset$ is $\gamma \cdot \boldsymbol{c}(A)$. The reaction's application results in the configuration with $\boldsymbol{c}(A) = 9$ and $\boldsymbol{c}(B) = 5$.

The propensity of $2A \xrightarrow{\delta} \emptyset$ is $\delta \cdot \frac{1}{2} \cdot \boldsymbol{c}(A) \cdot (\boldsymbol{c}(A) - 1)$. The reaction's application results in the configuration with $\boldsymbol{c}(A) = 8$ and $\boldsymbol{c}(B) = 5$.

Commonalities between CHP and CRN specifications. The production rule $A \to B \uparrow$ with some delay function and the reaction $A \xrightarrow{\alpha} A + B$ clearly bear some similarities: both trigger a rising voltage of A or increase the species count of A in presence of B. In CMOS, production rules occur in pairs with complementary guards, however, and biological applications of CRNs require conservation of mass which is violated by the above reaction. We will detail on this in the next section.

5.1.3 Implementing Circuits: Decoupling and Driving

A key benefit of circuits implemented in silicon, and in particular in CMOS, is decoupling between gates. Ideally, a gate's output behavior does not change its input voltage, i.e., the output behavior of the gate that drives one of its inputs. High-impedance decoupling with field-effect transistors (FETs) is central to achieve this goal. The charge used to load the gate's output capacitance is not taken from the input charge, but from an ideally undepletable power source connected to the gate's V_{DD} input.

A second key benefit in CMOS that does not necessarily hold for other processes (e.g., NMOS) is the fact that rising and falling output transitions are actively driven with low resistance and not just pulled up/down by a passive resistor.

While also applicable to synthetic biological circuits, many circuits do not ensure both properties. In [150] we discussed circuits in CMOS and biology with respect to decoupling and how they drive outputs. The discussion is summarized in the following.

CMOS gate. Given two production rules with negated Boolean guards, one setting an output to 0 and the other to 1, one can readily implement these rules within a combinational CMOS gate. For a gate with the rules

$$P_{\mathrm{up}} \to Z\uparrow$$
$$P_{\mathrm{down}} \to Z\downarrow$$

one first ensures that P_{up} comprises only of AND/OR combinations of negated variables, and P_{down} of AND/OR combinations of positive variables. If not, inputs are negated, or duplicated and negated, accordingly. A valid example is $P_{\mathrm{up}} = \neg A \wedge \neg B$ and $P_{\mathrm{down}} = A \vee B$, corresponding to a NOR-gate.

Each rule is then implemented by its own transistor stack: the P_{up} by the p-stack of p-type FET transistors, and the P_{down} by the n-stack of n-type FET transistors. Purpose of the respective stacks is to connect the output to V_{DD} or ground when the respective guard becomes true, and

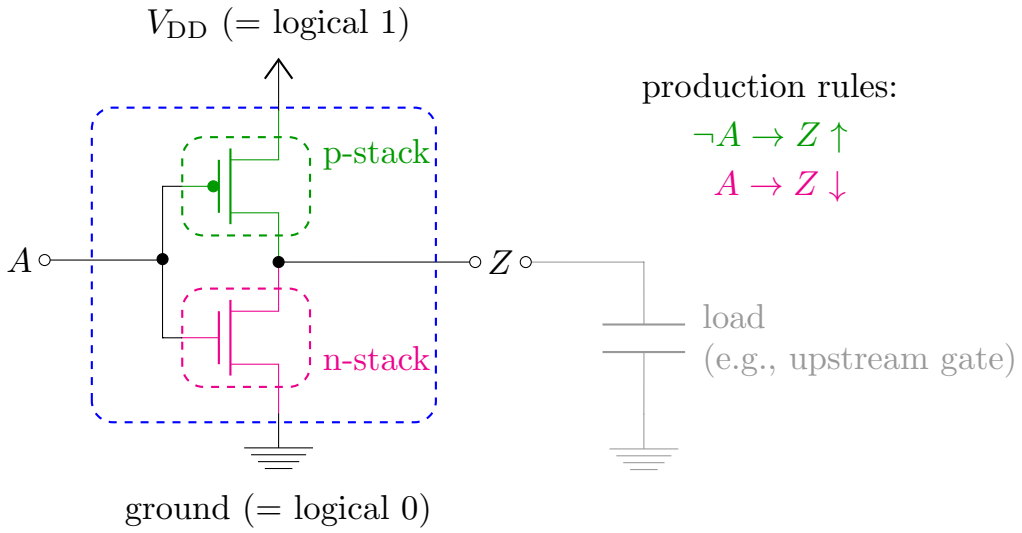


Figure 5.2: CMOS implementation of an INV-gate (left) driving its output, e.g., a gate (right). The output load is represented by a capacitive load. For the inverter with input A and output Z, $P_{\text{up}} = \neg A$ and $P_{\text{down}} = A$. The p-stack (above, in green) is the above p-type FET transistor and connects V_{DD} to Z if $A = 0$. The n-stack (below, in magenta) is the bottom n-type FET transistor and connects ground to Z if $A = 1$. From [150].

charging/discharging the output capacitance via the connected stack, while the other stack is disconnected. Figure 5.2 shows this schematic for a CMOS INV-gate.

Figure 5.3 shows the inverter in operation for $A = 0$ and $A = 1$.

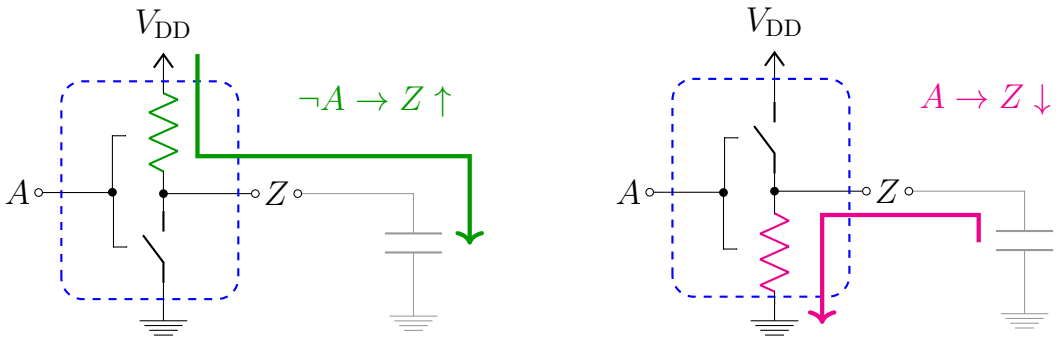


Figure 5.3: CMOS implementation of an INV-gate in operation. Both Boolean input cases along with charging and discharging paths are shown. From [150].

Coming back to the two properties of decoupling and driving, one observes from Figure 5.3:

Driving outputs. The CMOS INV implementation actively drives 0 via the connected n-stack and 1 via the connected p-stack, depending on whether the gate's input is 1 or 0. No high-resistance pull-networks are used to drive either value.

Decoupling inputs and outputs. When driving, the INV-gate uses charge from the power source via its V_{DD} port, rather than using charge from input A. Similarly, the output is discharged to ground. Both thus do not change the input's value.

Microbiological gates. Synthetic gates in living organisms have been built making use of many different pathways within a cell like a bacterial host [157]. Examples range from transcriptional control via transcription factors [152], CRISPRi (such as CRISPR-dCas9) [169–171] to recombinase/flippase-based approaches [172].

We will focus on enzymatic-type reactions that are at the basis of several evolved and synthetic circuits. Thereby a substrate species S is modified to obtain a product species P via an enzyme E. Letting S be the input and P the output, and using the notation from CMOS gates of $A = S$ and $Z = P$, the reaction is of the form

$$A + E \rightleftharpoons AE \to E + O \ .$$

Thus there is no decoupling happening, with the input being directly used to produce the output.

Decoupling is obtained, however, if $E = A$ plays the role of the input and S is the abundant charge available via V_{DD}, in which case the gate becomes

$$S + A \rightleftharpoons AS \to A + O \ .$$

To be precise, and much like gate current in CMOS, the reaction may bind a considerable amount of input A while producing O. Decoupling is maximal if a small such amount already leads to a large production of O; ensured via the reactions' rates.

A related decoupling, combining the above reactions, is by letting AE be the active enzyme that uses a substrate to produce O from S. In this case,

$$A + E \rightleftharpoons AE \overset{S}{\hookrightarrow} AE + O$$

The input/output behavior of the last two reactions can be described by the single production rule $A \to Z \uparrow$ in case $A = 1$. by contrast, if $A = 0$, the output Z is not actively driven to 0, but decays; much like in a pull-down network. We may thus write $Z \to Z \downarrow$, since this process is ongoing even if $A = 1$. The semantics of such a production rule — in particular since two conflicting rules may be active at the same time — may be defined via CRNs.

For a discussion of more involved microbiological gates and their corresponding production rules, the reader is referred to [150].

5.2 Distributed Microbiological Circuits

Circuits that are distributed in space, e.g., by circuit components partitioned among communicating host cells, have recently gained in interest [173]. A main motivation for distribution is the large metabolic load of even small circuits of a few gates. Other motivations are solving problems that are inherently distributed in nature and robustness to failures in a single cell.

In the following we will summarize our work on distributed computation of growing microbiological entities [151], as it is the case for bacteria.

5.2.1 Circuits in Growing Populations as Chemical Reaction Networks

In [151] we defined (distributed) protocols of birth systems as a model for circuits executed by growing populations of microbiological entities like bacteria.

Definition 129 ([151])**.** *A* protocol for a birth system *with input species $\mathcal{I}$ and output species $\mathcal{O}$ is a CRN with the following properties:*

- *Sets $\mathcal{I}$, called the protocol's inputs, and $\mathcal{O}$, the protocol's outputs, are not necessarily disjoint finite sets.*
- *The CRN's set of species $\mathcal{S}$ comprises the species in $\mathcal{I} \cup \mathcal{O}$ and a finite disjoint set of internal species $\mathcal{L}$.*
- *For each internal or output species, the protocol defines the initial species counts.*
- *The protocol specifies a finite set of reactions $\mathcal{R}$ on the species in $\mathcal{S}$.*
- *For each species $X \in \mathcal{S}$, there is a duplication reaction of the form $X \xrightarrow{\gamma} 2X$. All duplication reactions have the same reaction rate constant $\gamma > 0$.*

The probabilistic execution space of a protocol for a birth death system is then defined by the protocol and initial species counts for its inputs. It is the execution space of the corresponding CRN with respective initial counts.

5.2.2 Growing Populations Solving Majority Consensus

Consensus-type problems as discussed in Chapter 4 also play a role within distributed circuits. However, a variant of consensus with a stronger validity condition is relevant in these systems. In [151] we identified *majority consensus* as a key building block to tolerate noise in distributed circuits: microbiological agents may deviate from the correct outcome of a computation, but the majority consensus building block will make sure that the correct ones will finally determine the outcome of the computation; much like the amplification of analog values to clear logical values in CMOS circuits. Formally, we defined:

Definition 130 (A-B protocol, from [151])**.** *The* A-B protocol *is a protocol for a birth system with input species A and B. Given a rate constant $\delta > 0$, its reactions are:*

$$A \xrightarrow{\gamma} 2A \qquad B \xrightarrow{\gamma} 2B \qquad A + B \xrightarrow{\delta} \emptyset$$

The protocol thus has an annihilation rule between A and B as the single rule besides the required birth rules. We then defined consensus and majority consensus by:

Definition 131 (Consensus and majority consensus, from [151])**.** Consensus *is reached if at least one of A or B becomes extinct.* Majority consensus *is reached if:*

1. *In case initially $A(0) \neq B(0)$, consensus is reached and the species that was initially in majority is not extinct.*

2. *In case initially $A(0) = B(0)$, one species is extinct and the other is not.*

Our main result on majority consensus in [151] is stated in the following. Its proof is based on discrete and continuous time couplings to random walks.

Theorem 132 ([151])**.** *Given,*

- *an initial population $n = A(0) + B(0)$, and*
- *an initial gap $\Delta = |A(0) - B(0)|$,*

the A-B protocol reaches consensus in expected time $O(1)$ and in time $O(\log n)$ with high probability. It reaches majority consensus with probability

$$1 - e^{-\Omega(\Delta^2/n)} \ .$$

From this one obtains:

Corollary 133 ([151])**.** *Given,*

- *an initial population $n = A(0) + B(0)$, and*
- *an initial gap $\Delta = |A(0) - B(0)| \in \Omega\left(\sqrt{n \log n}\right)$,*

the A-B protocol reaches consensus in expected time $O(1)$ and in time $O(\log n)$ with high probability. It reaches majority consensus with high probability.

5.2.3 Distributed Gates

Along the idea to use the majority consensus as an amplifier for noisy/unclean signals, leading to their regeneration, we defined dual-rail signals and gates with dual-rail inputs and outputs in [151] as follows. Dual-rail encoded signals are used in clockless circuits to allow the receiver to distinguish old from new data [5, 174].

A *signal* is defined as in Section 1.2 as a potentially infinite sequence of events. However, taking into account non-binary/intermediate values that play an important role in microbiological circuits, we allow events values to be from $\{0, 1, \mathrm{M}\}$, where M denotes the unstable/metastable value.

A *port* is from a finite alphabet $\Sigma = \{X, Y, \dots\}$. Each port X is assigned a signal x; and we use $x(t)$ for the value of x at time t. For each signal X, there are two species X^0 and X^1, making up the two rails of the dual-rail signal; e.g., high numbers of X^0 and low numbers of X^1 at time t encode $x(t) = 0$; and vice versa for $x(t) = 1$. We use $x(t) = \mathrm{M}$ for the case where neither is the case. Formally:

Definition 134 (Dual-rail encoding, from [151])**.** *Let X^0, X^1 be species of a dual-rail encoding of signal X. Let $X(t) = X^0(t) + X^1(t)$. For $n, \Delta \in \mathbb{N}$, we say signal X is* initially (n, Δ)-correct with value $x \in \{0, 1\}$ *if*

$$X(0) \geq n \text{ and}$$
$$X^{\neg x}(0) \leq \frac{n - \Delta}{2} \ .$$

The initial gap of signal X *is* $X^x(0) - X^{\neg x}(0)$.

One observes that the initial gap of an initially (n, Δ)-correct signal is bounded by

$$X^x(0) - X^{\neg x}(0) = X^x(0) + X^{\neg x}(0) - 2X^{\neg x}(0) \geq \Delta \ .$$

Equipped with this encoding of signals, we defined a protocol for a birth system that computes a Boolean function; in this case of a NAND gate. Other Boolean functions are computable along the same lines.

Definition 135 (NAND gate implementation, from [151])**.** *The implementation of a* NAND *gate with input signals A, B and output signal Y is as a protocol for a birth system with*

1. *input species $\mathcal{I} = \{A^0, A^1, B^0, B^1\}$,*
2. *output species $\mathcal{O} = \{Y^0, Y^1\}$ with initial counts 0, and*
3. *no internal species.*

For all $a, b \in \{0, 1\}$ and $y = \neg(a \wedge b)$, the protocol contains a reaction

$$A^a + B^b \xrightarrow{\alpha} A^a + B^b + Y^y \ ,$$

where $\alpha > 0$ is the gate's reaction rate constant. Additionally, all species have duplication reactions with a reaction rate constant $\gamma > 0$.

Based on a reduction to Yule processes, we proved our main result on the gate's behavior which is as follows:

Theorem 136 ([151])**.** *Given, the* NAND *gate protocol with dual-rail encoded input signals A and B and output signal Y, such that:*

- *The input signals are initially (n, Δ)-correct with respective values $a, b \in \{0, 1\}$, where*

$$n \in \mathbb{N}^+ \text{ and}$$
$$\Delta \geq 0.62 \cdot \max\{A(0), B(0)\} \ .$$

Then, with high probability, there exists a time $t = O(1)$ such that

- $Y(t) = n$, *and*
- $Y^y(t) - Y^{\neg y}(t) = \Omega(n)$, *where* $y = \neg(a \wedge b)$.

The gate thus provides a linear gap at its output by producing sufficiently separated dual-rail output species in a sufficient number, given that the dual-rail input species are sufficiently noise-free. Combining the gate with the A-B protocol as an amplification stage, thus allows one to compute Boolean functions and afterwards regenerate the Boolean signal; ready to be fed into the next upstream gate.

5.3 Concluding Remarks

Low-level implementations with design entries at the hardware/circuit-level often rely on state-machine-based approaches. Assumptions made during the design and analysis stages greatly ease this process. However, in several cases simplifying assumptions have to be dropped. Examples are (i) pronounced analog/timed behavior of switching signals, the related problem of non-binary signals, respectively, metastable upsets, and (iii) faulty or mobile components that led to distributed circuits with unstable communication between components. This work discussed approaches and solutions to these cases.

In this last chapter a quick overview on our recent work was presented: the study and design of synthetic biological distributed circuits. At the time of writing, Manish Kushwaha (INRAE), Thomas Nowak (LRI), and myself have formed a group and are working on further pushing this direction. We belief that a combined, interdisciplinary approach in the domains of circuit design, distributed computing, and synthetic biology has great potential to contribute to the road that recent work in the domain of synthetic biology has shown with numerous applications in drug production and intelligent drugs. Based on our previous work and recent unpublished work, we are working on communicating bacterial consortia to design robust distributed circuits. Besides many implementation and circuit design questions that have to be solved along this road, immediate questions from the theory of distributed computing are (i) improved majority consensus algorithms and/or an improved analysis with resource limitations, and (ii) stability of co-cultures that run different circuit parts. The author believes that advancing on some of the techniques presented in this work will be fruitful in solving these questions.

Bibliography

[1] Alain J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1(4):226–234, December 1986. doi: 10.1007/bf01660034. URL https://doi.org/10.1007/bf01660034.

[2] Alain J Martin. Synthesis of asynchronous VLSI circuits. Technical report, Caltech, 1991. URL https://authors.library.caltech.edu/26746/2/postscript.pdf.

[3] Rajit Manohar and Yoram Moses. Asynchronous signalling processes. In *2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. IEEE, may 2019. doi: 10.1109/async.2019.00018. URL https://doi.org/10.1109/async.2019.00018.

[4] Jens Sparsø. *Asynchronous circuit design-a tutorial*. Kluwer Academic Publishers, 2001.

[5] Chris J. Myers. *Asynchronous Circuit Design*. John Wiley & Sons, Inc., July 2001. doi: 10.1002/0471224146. URL https://doi.org/10.1002/0471224146.

[6] Jean-Luc Autran and Daniela Munteanu. *Soft Errors: from particles to circuits*. CRC Press, 2017.

[7] Wojciech Maly. Realistic fault modeling for vlsi testing. In *24th ACM/IEEE Design Automation Conference*, pages 173–180. IEEE, 1987.

[8] Heinrich Theodor Vierhaus, Wolfgang Meyer, and U Glaser. Cmos bridges and resistive transistor faults: Iddq versus delay effects. In *Proceedings of IEEE International Test Conference-(ITC)*, pages 83–91. IEEE, 1993.

[9] Leonard R. Marino. General theory of metastable operation. *IEEE Transactions on Computers*, C-30(2):107–115, February 1981. doi: 10.1109/tc.1981.6312173. URL https://doi.org/10.1109/tc.1981.6312173.

[10] Leonard R. Marino. The effect of asynchronous inputs on sequential network reliability. *IEEE Transactions on Computers*, C-26(11):1082–1090, November 1977. doi: 10.1109/tc.1977.1674754. URL https://doi.org/10.1109/tc.1977.1674754.

[11] Matthias Függer, Ulrich Schmid, Gottfried Fuchs, and Gerald Kempf. Fault-tolerant distributed clock generation in VLSI Systems-on-Chip. In *Sixth European Dependable Computing Conference (EDCC)*, pages 87–96, Coimbra, Portugal, October 2006. doi: 10.1109/EDCC.2006.11. URL https://ieeexplore.ieee.org/document/4020838.

[12] Matthias Függer and Ulrich Schmid. Reconciling fault-tolerant distributed computing and systems-on-chip. *Distributed Computing*, 24(6):323–355, 2012. doi: 10.1007/s00446-011-0151-7. URL https://link.springer.com/article/10.1007/s00446-011-0151-7.

[13] Danny Dolev, Matthias Függer, Ulrich Schmid, and Christoph Lenzen. Fault-tolerant algorithms for tick-generation in asynchronous logic: Robust pulse generation. *J. ACM*, 61 (5):30:1–30:74, September 2014. doi: 10.1145/2560561. URL http://doi.acm.org/10.1145/2560561.

[14] Danny Dolev, Matthias Függer, Markus Posch, Ulrich Schmid, Andreas Steininger, and Christoph Lenzen. Rigorously modeling self-stabilizing fault-tolerant circuits: An ultra-robust clocking scheme for systems-on-chip. *Journal of Computer and System Sciences*, 80(4):860–900, 2014. doi: 10.1016/j.jcss.2014.01.001. URL http://www.sciencedirect.com/science/article/pii/S0022000014000026.

[15] Danny Dolev, Matthias Függer, Christoph Lenzen, Ulrich Schmid, and Andreas Steininger. Fault-tolerant distributed systems in hardware. *Bulletin of EATCS*, 2(116), 2015. URL http://bulletin.eatcs.org/index.php/beatcs/article/view/348.

[16] Matthias Függer, Thomas Nowak, and Ulrich Schmid. Unfaithful glitch propagation in existing binary circuit models. *IEEE Transactions on Computers*, 65(3):964–978, 2016. doi: 10.1109/TC.2015.2435791. URL http://ieeexplore.ieee.org/abstract/document/7110587/.

[17] Matthias Függer, Thomas Nowak, and Ulrich Schmid. Unfaithful glitch propagation in existing binary circuit models. In *2013 IEEE 19th International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 191–199, 2013. URL https://ieeexplore.ieee.org/document/6546194/.

[18] Matthias Függer, Robert Najvirt, Thomas Nowak, and Ulrich Schmid. A faithful binary circuit model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2784–2797, 2020. doi: 10.1109/TCAD.2019.2937748.

[19] Matthias Függer, Robert Najvirt, Thomas Nowak, and Ulrich Schmid. Towards binary circuit models that faithfully capture physical solvability. In *Design, Automation & Test in Europe (DATE)*, pages 1455–1460, 2015. doi: 10.7873/DATE.2015.0799. URL http://ieeexplore.ieee.org/document/7092619/.

[20] Robert Najvirt, Matthias Függer, Thomas Nowak, Ulrich Schmid, Michael Hofbauer, and Kurt Schweiger. Experimental validation of a faithful binary circuit model. In *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI (GLSVLSI)*, pages 355–360. ACM, 2015. doi: 10.1145/2742060.2742081. URL http://doi.acm.org/10.1145/2742060.2742081.

[21] Matthias Függer, Jürgen Maier, Robert Najvirt, Thomas Nowak, and Ulrich Schmid. A faithful binary circuit model with adversarial noise. In *Design, Automation & Test in Europe (DATE)*, pages 1327–1332. IEEE, 2018. doi: 10.23919/DATE.2018.8342219. URL https://ieeexplore.ieee.org/abstract/document/8342219/.

[22] Mayler Martins, Jody Maick Matos, Renato Ribas, André Reis, Guilherme Schlinker, Lucio Rech, and Jens Michelsen. Open cell library in 15nm freepdk technology. 04 2015. doi: 10.1145/2717764.2717783.

[23] Stephen H Unger. Asynchronous sequential switching circuits with unrestricted input changes. *IEEE Transactions on computers*, 100(12):1437–1444, 1971.

[24] MJ Bellido-Diaz, J Juan-Chico, AJ Acosta, M Valencia, and JL Huertas. Logical modelling of delay degradation effect in static cmos gates. *IEE Proceedings-Circuits, Devices and Systems*, 147(2):107–117, 2000.

[25] Daniel Öhlinger, Jürgen Maier, Matthias Függer, and Ulrich Schmid. The involution tool for accurate digital timing and power analysis. In *29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 1–8, July 2019. doi: 10.1109/PATMOS.2019.8862165.

[26] Michael Hofbauer, Kurt Schweiger, Horst Dietrich, Horst Zimmermann, Kay-Obbe Voss, Bruno Merk, Ulrich Schmid, and Andreas Steininger. Pulse shape measurements by on-chip sense amplifiers of single event transients propagating through a 90 nm bulk CMOS inverter chain. *IEEE Transactions on Nuclear Science*, 59(6):2778–2784, dec 2012.

[27] Jürgen Maier, Matthias Függer, Thomas Nowak, and Ulrich Schmid. Transistor-level analysis of dynamic delay models. In *25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 76–85, May 2019. doi: 10.1109/ASYNC.2019.00019.

[28] Stephan Friedrichs, Matthias Függer, and Christoph Lenzen. Metastability-containing circuits. *IEEE Transactions on Computers*, 67(8), 2018. doi: 10.1109/TC.2018.2808185. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8314764.

[29] Matthias Függer, Attila Kinali, Christoph Lenzen, and Thomas Polzer. Metastability-aware memory-efficient time-to-digital converter. In *IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 49–56, 2017. doi: 10.1109/ASYNC.2017.12. URL http://ieeexplore.ieee.org/abstract/document/8097384/.

[30] Ghaith Tarawneh, Matthias Függer, and Christoph Lenzen. Metastability tolerant computing. In *IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 25–32, 2017. doi: 10.1109/ASYNC.2017.9. URL http://ieeexplore.ieee.org/abstract/document/8097381/.

[31] Matthias Függer, Attila Kinali, Christoph Lenzen, and Ben Wiederhake. Fast all-digital clock frequency adaptation circuit for voltage droop tolerance. In *IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 68–77, May 2018. doi: 10.1109/ASYNC.2018.00025. URL https://ieeexplore.ieee.org/document/8589986.

[32] Johannes Bund, Matthias Függer, Christoph Lenzen, and Moti Medina. Synchronizer-free digital link controller. *IEEE Transactions on Circuits and Systems I*, 67(10):3562–3573, 2020. doi: 10.1109/TCSI.2020.2989552. URL https://ieeexplore.ieee.org/document/9085899.

[33] R. Ginosar. Metastability and synchronizers: A tutorial. *IEEE Design & Test of Computers*, 28(5):23–35, September 2011. doi: 10.1109/mdt.2011.113. URL https://doi.org/10.1109/mdt.2011.113.

[34] Thomas Polzer, Florian Huemer, and Andreas Steininger. An experimental study of metastability-induced glitching behavior. *Journal of Circuits, Systems and Computers*, 28 (supp01):1940006, 2019.

[35] Jens U Horstmann, Hans W Eichel, and Robert L Coates. Metastability behavior of cmos asic flip-flops in theory and test. *IEEE Journal of solid-state circuits*, 24(1):146–157, 1989.

[36] David John Kinniment. *Synchronization and Arbitration in Digital Systems*. Wiley, 2008.

[37] David Kinniment, Keith Heron, and Gordon Russell. Measuring deep metastability. In *12th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'06)*, pages 10–pp. IEEE, 2006.

[38] Harry JM 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.

[39] Salomon Beer, Ran Ginosar, Jerome Cox, Tom Chaney, and David M Zar. Metastability challenges for 65nm and beyond; simulation and measurements. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1297–1302. IEEE, 2013.

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

[41] Salomon Beer and Ran Ginosar. Eleven ways to boost your synchronizer. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(6):1040–1049, 2014.

[42] Suwen Yang, Ian W Jones, and Mark R Greenstreet. Synchronizer performance in deep submicron technology. In *2011 17th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 33–42. IEEE, 2011.

[43] Jerome Cox, George Engel, David Zar, and Ian W Jones. Synchronizers and data flip-flops are different. In *2015 21st IEEE International Symposium on Asynchronous Circuits and Systems*, pages 19–20. IEEE, 2015.

[44] Andreas Steininger, Jürgen Maier, and Robert Najvirt. The metastable behavior of a schmitt-trigger. In *2016 22nd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 57–64. IEEE, 2016.

[45] Gottfried Fuchs, Matthias Függer, and Andreas Steininger. On the threat of metastability in an asynchronous fault-tolerant clock generation scheme. *15th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 127–136, May 2009. doi: 10.1109/ASYNC.2009.15. URL https://ieeexplore.ieee.org/document/5010343/.

[46] Thomas Polzer, Andreas Steininger, and Jakob Lechner. Muller c-element metastability containment. In *International Workshop on Power and Timing Modeling, Optimization and Simulation*, pages 103–112. Springer, 2012.

[47] Ian W Jones, Suwen Yang, and Mark Greenstreet. Synchronizer behavior and analysis. In *2009 15th IEEE Symposium on Asynchronous Circuits and Systems*, pages 117–126. IEEE, 2009.

[48] Alex Yakovlev, Michael Kishinevsky, Alex Kondratyev, and Luciano Lavagno. Or causality: modelling and hardware implementation. In *International Conference on Application and Theory of Petri Nets*, pages 568–587. Springer, 1994.

[49] Ghaith Tarawneh and Alex Yakovlev. An rtl method for hiding clock domain crossing latency. In *2012 19th IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012)*, pages 540–543. IEEE, 2012.

[50] Ghaith Tarawneh, Alex Yakovlev, and Terrence Mak. Eliminating synchronization latency using sequenced latching. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(2):408–419, 2013.

[51] David A. Huffman. The design and use of hazard-free switching networks. *Journal of the ACM*, 4(1):47–62, 1957. ISSN 0004-5411.

[52] Steven H. Unger. Hazards and delays in asynchronous sequential switching circuits. *IRE Transactions on Circuit Theory*, 6(1):12–25, 1959. ISSN 0096-2007. doi: 10.1109/TCT.1959.1086527.

[53] Edward B. Eichelberger. Hazard detection in combinational and sequential switching circuits. *IBM Journal of Research and Development*, 9(2):90–99, 1965. ISSN 0018-8646. doi: 10.1147/rd.92.0090.

[54] Janusz A. Brzozowski and Michael Yoeli. On a ternary model of gate networks. *IEEE Transactions on Computers*, C-28(3):178–184, 1979. ISSN 0018-9340. doi: 10.1109/TC.1979.1675317.

[55] Janusz A. Brzozowski, Zoltán Ésik, and Y. Iland. Algebras for hazard detection. In *ISMVL*, page 3, 2001.

[56] Michael Mendler, Thomas R. Shiple, and Gérard Berry. Constructive boolean circuits and the exactness of timed ternary simulation. *Formal Methods in System Design*, 40(3):283–329, 2012. doi: 10.1007/s10703-012-0144-6. URL http://dx.doi.org/10.1007/s10703-012-0144-6.

[57] Dylan Hand, Matheus Trevisan Moreira, Hsin-Ho Huang, Danlei Chen, Frederico Butzke, Zhichao Li, Matheus Gibiluka, Melvin Breuer, Ney Laert Vilar Calazans, and Peter A Beerel. Blade–a timing violation resilient asynchronous template. In *2015 21st IEEE International Symposium on Asynchronous Circuits and Systems*, pages 21–28. IEEE, 2015.

[58] Matthew Fojtik, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Stephen G Tell, Brian Zimmer, Tezaswi Raja, Kevin Zhou, William J Dally, and Brucek Khailany. A fine-grained gals soc with pausible adaptive clocking in 16 nm finfet. In *2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 27–35. IEEE, 2019.

[59] Jozef Kalisz. Review of methods for time interval measurements with picosecond resolution. *Metrologia*, 41(1):17, 2004.

[60] M. A. Daigneault and J. P. David. A Novel 10 ps Resolution TDC Architecture Implemented in a 130nm Process FPGA. In *NEWCAS'10*, pages 281–284, 2010.

[61] L. Perktold and J. Christiansen. A fine time-resolution (<3 ps-rms) time-to-digital converter for highly integrated designs. In *IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pages 1092–1097, May 2013. doi: 10.1109/I2MTC.2013.6555583.

[62] T. Rahkonen and J. Kostamovaara. The use of stabilized CMOS delay lines in the digitization of short time intervals. In *ISCAS'91*, pages 2252–2255, 1991. doi: 10.1109/ISCAS.1991.176828.

[63] R.B. Staszewski, S. Vemulapalli, P. Vallur, J. Wallberg, and P.T. Balsara. 1.3 V 20 ps time-to-digital converter for frequency synthesis in 90-nm CMOS. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 53(3):220–224, 2006. ISSN 1549-7747. doi: 10.1109/TCSII.2005.858754.

[64] Ronald Nutt. Digital Time Intervalometer. *Review of scientific instruments*, 39(9):1342–1345, 1968.

[65] David J. Kinniment. *Synchronization and Arbitration in Digital Systems*. Wiley Publishing, 2008.

[66] Stephan Friedrichs, Matthias Függer, and Christoph Lenzen. Metastability-Containing Circuits. *CoRR*, abs/1606.06570, 2016.

[67] Johannes Bund, Christoph Lenzen, and Moti Medina. Near-Optimal Metastability-Containing Sorting Networks. In *DATE'17*, 2017.

[68] Christoph Lenzen and Moti Medina. Efficient Metastability-Containing Gray Code 2-Sort. In *ASYNC'16*, 2016.

[69] M. Mota, J. Christiansen, S. Debieux, V. Ryjov, P. Moreira, and A. Marchioro. A flexible multi-channel high-resolution time-to-digital converter ASIC. In *Nuclear Science Symposium Conference Record, IEEE*, volume 2, pages 9/155–9/159 vol.2, 2000. doi: 10.1109/NSSMIC.2000.949889.

[70] Stephan Henzler. *Time-to-Digital Converters*. Springer, 2010.

[71] P. Dudek, S. Szczepanski, and J.V. Hatfield. A high-resolution CMOS time-to-digital converter utilizing a Vernier delay line. *Solid-State Circuits, IEEE Journal of*, 35(2):240–247, 2000. ISSN 0018-9200. doi: 10.1109/4.823449.

[72] Miklós Ajtai, János Komlós, and Endre Szemerédi. An 0(n log n) sorting network. In *Proceedings of the fifteenth annual ACM Symposium on Theory of Computing*, pages 1–9, 1983.

[73] Kenneth E Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 307–314, 1968.

[74] Christoph Lenzen and Moti Medina. Efficient metastability-containing gray code 2-sort. In *2016 22nd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 49–56. IEEE, 2016.

[75] Johannes Bund, Christoph Lenzen, and Moti Medina. Near-optimal metastability-containing sorting networks. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 226–231. IEEE, 2017.

[76] Johannes Bund, Christoph Lenzen, and Moti Medina. Optimal metastability-containing sorting networks. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 521–526. IEEE, 2018.

[77] Ali Hajimiri, Sotirios Limotyrakis, and Thomas H Lee. Jitter and phase noise in ring oscillators. *IEEE Journal of Solid-state circuits*, 34(6):790–804, 1999.

[78] Shruti Suman, KG Sharma, and PK Ghosh. Analysis and design of current starved ring vco. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 3222–3227. IEEE, 2016.

[79] Leslie Lamport and P Michael Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM (JACM)*, 32(1):52–78, 1985.

[80] Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996.

[81] Danny Dolev, Nancy A Lynch, Shlomit S Pinter, Eugene W Stark, and William E Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM (JACM)*, 33 (3):499–516, 1986.

[82] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.

[83] Cristian Constantinescu. Trends and challenges in vlsi circuit reliability. *IEEE micro*, 23(4): 14–19, 2003.

[84] Shidhartha Das, Carlos Tokunaga, Sanjay Pant, Wei-Hsiang Ma, Sudherssen Kalaiselvan, Kevin Lai, David M Bull, and David T Blaauw. Razorii: In situ error detection and correction for pvt and ser tolerance. *IEEE Journal of Solid-State Circuits*, 44(1):32–48, 2008.

[85] Felipe A Kuentzer, Moises Herrera, Oliver Schrape, Peter A Beerel, and Milos Krstic. Radiation hardened click controllers for soft error resilient asynchronous architectures. In *2020 26th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 78–85. IEEE, 2020.

[86] Veronique Ferlet-Cavrois, Lloyd W Massengill, and Pascale Gouker. Single event transients in digital cmos—a review. *IEEE Transactions on Nuclear Science*, 60(3):1767–1790, 2013.

[87] Marko Andjelkovic, Yuanqing Li, Zoran Stamenkovic, Milos Krstic, and Rolf Kraemer. Characterization and modeling of set generation effects in cmos standard logic cells. In *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 212–215. IEEE, 2019.

[88] Michael Hofbauer, Kurt Schweiger, Horst Dietrich, Horst Zimmermann, Kay-Obbe Voss, Bruno Merk, Ulrich Schmid, and Andreas Steininger. Pulse shape measurements by on-chip sense amplifiers of single event transients propagating through a 90 nm bulk cmos inverter chain. *IEEE Transactions on Nuclear Science*, 59(6):2778–2784, 2012.

[89] Jennifer Lundelius Welch and Nancy Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and computation*, 77(1):1–36, 1988.

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

[91] R. Belschner, J. Berwanger, F. Bogenberger, C. Ebner, H. Eisele, B. Elend, T.M. Forest, T. Führer, P. Fuhrmann, F. Hartwich, et al. Flexray communication protocol, 2003. URL https://www.google.com/patents/EP1355456A1?cl=en. EP Patent App. EP20,020,008,171.

[92] Attila Kinali, Florian Huemer, and Christoph Lenzen. Fault-tolerant clock synchronization with high precision. In *IEEE Symposium on VLSI (ISVLSI)*, pages 490–495, 2016.

[93] Jun Zhou, David J Kinniment, Charles E Dike, Gordon Russell, and Alexandre V Yakovlev. On-chip measurement of deep metastability in synchronizers. *IEEE Journal of Solid-State Circuits*, 43(2):550–557, 2008.

[94] Thomas Polzer, Florian Huemer, and Andreas Steininger. Refined metastability characterization using a time-to-digital converter. *Microelectronics Reliability*, 80:91–99, 2018.

[95] Paul Teehan, Mark Greenstreet, and Guy Lemieux. A Survey and Taxonomy of GALS Design Styles. *IEEE Design & Test of Computers*, 24(5):418–428, 2007.

[96] Milos Krstic, Eckhard Grass, Frank K Gürkaynak, and Pascal Vivet. Globally asynchronous, locally synchronous circuits: Overview and outlook. *IEEE Design & Test of computers*, 24 (5):430–441, 2007.

[97] William S Coates and Robert J Drost. Congestion and Starvation Detection in Ripple FIFOs. In *ASYNC*, pages 36–45, 2003.

[98] Robert Mullins and Simon Moore. Demystifying Data-Driven and Pausible Clocking Schemes. In *ASYNC*, pages 175–185, 2007.

[99] Robert Najvirt and Andreas Steininger. How to Synchronize a Pausible Clock to a Reference. In *ASYNC*, pages 9–16, 2015.

[100] Tiberiu Chelcea and Steven M Nowick. Robust Interfaces for Mixed-Timing Systems. *IEEE Transactions on VLSI Systems*, 12(8):857–873, 2004.

[101] William J Dally and Stephen G Tell. The Even/Odd Synchronizer: A Fast, All-Digital, Periodic Synchronizer. In *ASYNC*, pages 75–84, 2010.

[102] Rostislav Dobkin, Ran Ginosar, and Christos P Sotiriou. High Rate Data Synchronization in GALS SoCs. *IEEE Transactions on VLSI Systems*, 14(10):1063–1074, 2006.

[103] Sandra Jackson and Rajit Manohar. Gradual Synchronization. In *ASYNC*, pages 29–36, 2016.

[104] Thomas Polzer, Thomas Handl, and Andreas Steininger. A Metastability-Free Multi-synchronous Communication Scheme for SoCs. In *SSS*, pages 578–592, 2009.

[105] Ching-Che Chung and Chen-Yi Lee. An All-Digital Phase-Locked Loop for High-Speed Clock Generation. *IEEE Journal of Solid-State Circuits*, 38(2):347–351, 2003.

[106] Arash Abadian, Mojtaba Lotfizad, Nasser Erfani Majd, Mohammad Bagher Ghaznavi Ghoushchi, and Hossein Mirzaie. A New Low-Power and Low-Complexity All Digital PLL (ADPLL) in 180 nm and 32 nm. In *ICECS*, pages 305–310, 2010.

[107] Keith Bowman, Carlos Tokunaga, James Tschanz, Arijit Raychowdhury, Muhammad Khellah, Bibiche Geuskens, Shih-Lien Lu, Paolo Aseron, Tanay Karnik, and Vivek De. Dynamic variation monitor for measuring the impact of voltage droops on microprocessor clock frequency. In *CICC*, pages 1–4. IEEE, 2010.

[108] Nasser Kurd, Praveen Mosalikanti, Mark Neidengard, Jonathan Douglas, and Rajesh Kumar. Next generation intel core™ micro-architecture (nehalem) clocking. *IEEE SSC*, 44(4):1121–1129, 2009.

[109] Dong Jiao, Jie Gu, and Chris H Kim. Circuit design and modeling techniques for enhancing the clock-data compensation effect under resonant supply noise. *IEEE SSC*, 45(10):2130–2141, 2010.

[110] Matthew Fojtik, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Stephen G Tell, Brian Zimmer, Tezaswi Raja, Kevin Zhou, William J Dally, and Brucek Khailany. A fine-grained gals soc with pausible adaptive clocking in 16 nm finfet. In *2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 27–35. IEEE, 2019.

[111] Jordi Cortadella, Marc Lupon, Alberto Moreno, Antoni Roca, and Sachin S Sapatnekar. Ring oscillator clocks and margins. In *Asynchronous Circuits and Systems (ASYNC), 2016 22nd IEEE International Symposium on*, pages 19–26. IEEE, 2016.

[112] James Tschanz, Nam Sung Kim, Saurabh Dighe, Jason Howard, Gregory Ruhl, Sriram Vangal, Siva Narendra, Yatin Hoskote, Howard Wilson, Carol Lam, et al. Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging. In *ISCC*, pages 292–604. IEEE, 2007.

[113] Tim Fischer, Jayen Desai, Bruce Doyle, Samuel Naffziger, and Ben Patella. A 90-nm variable frequency clock system for a power-managed itanium architecture processor. *IEEE SSC*, 41 (1):218–228, 2006.

[114] Keith A Bowman, Carlos Tokunaga, Tanay Karnik, Vivek K De, and James W Tschanz. A 22 nm all-digital dynamically adaptive clock distribution for supply voltage droop tolerance. *IEEE SSC*, 48(4):907–916, 2013.

[115] Johannes Bund, Matthias Függer, Christoph Lenzen, Moti Medina, and Will Rosenbaum. Pals: Plesiochronous and locally synchronous systems. In *26th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 36–43, Los Alamitos, CA, USA, 2020. IEEE Computer Society. doi: 10.1109/ASYNC49171.2020.00013. URL https://doi.ieeecomputersociety.org/10.1109/ASYNC49171.2020.00013.

[116] Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. Approximate consensus in highly dynamic networks: The role of averaging algorithms. In *42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 528–539, 2015. doi: 10.1007/978-3-662-47666-6_42. URL https://link.springer.com/chapter/10.1007/978-3-662-47666-6_42.

[117] Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. Fast, robust, quantizable approximate consensus. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 55, pages 137:1–137:14, 2016. doi: 10.4230/LIPIcs.ICALP.2016.137. URL http://drops.dagstuhl.de/opus/volltexte/2016/6281/.

[118] Matthias Függer, Thomas Nowak, and Kyrill Winkler. On the radius of nonsplit graphs and information dissemination in dynamic networks. *Discrete Applied Mathematics*, 282:257–264, 2020. doi: 10.1016/j.dam.2020.02.013. URL http://www.sciencedirect.com/science/article/pii/S0166218X20300809.

[119] Matthias Függer, Thomas Nowak, and Manfred Schwarz. Tight bounds for asymptotic and approximate consensus. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 325–334, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5795-1. URL http://doi.acm.org/10.1145/3212734.3212762.

[120] Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. Multidimensional asymptotic consensus in dynamic networks. *CoRR*, abs/1611.02496, 2016. URL http://arxiv.org/abs/1611.02496.

[121] Matthias Függer and Thomas Nowak. Fast multidimensional asymptotic and approximate consensus. In *32nd International Symposium on Distributed Computing (DISC)*, Leibniz International Proceedings in Informatics (LIPIcs), pages 27:1–27:15, 2018. doi: 10.4230/LIPIcs.DISC.2018.27. URL http://drops.dagstuhl.de/opus/volltexte/2018/9816/.

[122] Hammurabi Mendes, Maurice Herlihy, Nitin Vaidya, and Vijay K Garg. Multidimensional agreement in byzantine systems. *Distributed Computing*, 28(6):423–441, 2015.

[123] Nicola Santoro and Peter Widmayer. Time is not a healer. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 304–313. Springer, 1989.

[124] Bernadette Charron-Bost and André Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009.

[125] Yehuda Afek and Eli Gafni. Asynchrony from synchrony. In *International Conference on Distributed Computing and Networking*, pages 225–239. Springer, 2013.

[126] Étienne Coulouma, Emmanuel Godard, and Joseph Peters. A characterization of oblivious message adversaries for which consensus is solvable. *Theoretical Computer Science*, 584: 80–90, 2015.

[127] Michel Raynal and Julien Stainer. Synchrony weakened by message adversaries vs asynchrony restricted by failure detectors. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, pages 166–175, 2013.

[128] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 513–522, 2010.

[129] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[130] Kyrill Winkler, Ulrich Schmid, and Yoram Moses. A characterization of consensus solvability for closed message adversaries. In *23rd International Conference on Principles of Distributed Systems (OPODIS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[131] Thomas Nowak, Ulrich Schmid, and Kyrill Winkler. Topological characterization of consensus under general message adversaries. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 218–227, 2019.

[132] Fabian Kuhn and Rotem Oshman. Dynamic networks: models and algorithms. *ACM SIGACT News*, 42(1):82–96, 2011.

[133] Ming Cao, A Stephen Morse, and Brian DO Anderson. Reaching a consensus in a dynamically changing environment: A graphical approach. *SIAM Journal on Control and Optimization*, 47(2):575–600, 2008.

[134] Manfred Schwarz, Kyrill Winkler, and Ulrich Schmid. Fast consensus under eventually stabilizing message adversaries. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, pages 1–10, 2016.

[135] Luc Moreau. Stability of multiagent systems with time-dependent communication links. *IEEE Transactions on automatic control*, 50(2):169–182, 2005.

[136] Vincent D Blondel, Julien M Hendrickx, Alex Olshevsky, and John N Tsitsiklis. Convergence in multiagent coordination, consensus, and flocking. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 2996–3000. IEEE, 2005.

[137] Bernard Chazelle. The total s-energy of a multiagent system. *SIAM Journal on Control and Optimization*, 49(4):1680–1706, 2011.

[138] Bernard Chazelle. Natural algorithms and influence systems. *Communications of the ACM*, 55(12):101–110, 2012.

[139] Ittai Abraham, Yonatan Amit, and Danny Dolev. Optimal resilience asynchronous approximate agreement. In *International Conference On Principles Of Distributed Systems*, pages 229–239. Springer, 2004.

[140] Alan David Fekete. Asymptotically optimal algorithms for approximate agreement. *Distributed Computing*, 4(1):9–29, 1990.

[141] Ming Cao, A Stephen Morse, and Brian DO Anderson. Reaching a consensus in a dynamically changing environment: convergence rates, measurement delays, and asynchronous events. *SIAM Journal on Control and Optimization*, 47(2):601–623, 2008.

[142] Wenwu Yu, Guanrong Chen, Ming Cao, and Jürgen Kurths. Second-order consensus for multiagent systems with directed topologies and nonlinear dynamics. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 40(3):881–891, 2009.

[143] Vincent D Blondel and Alex Olshevsky. How to decide consensus? a combinatorial necessary and sufficient condition and a proof that consensus is decidable but np-hard. *SIAM Journal on Control and Optimization*, 52(5):2707–2726, 2014.

[144] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[145] Michael J Fischer, Nancy A Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.

[146] Ming Cao, Daniel A. Spielman, and A. Stephen Morse. A lower bound on convergence of a distributed network consensus algorithm. In Hannes Frey, Xu Li, and Stefan Rührup, editors, *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference (CDC-ECC 2005)*, pages 2356–2361, New York, 2005. IEEE.

[147] Alan D. Fekete. Asynchronous approximate agreement. *Information and Computation*, 115 (1):95–124, 1994.

[148] Gunnar Hoest and Nir Shavit. Toward a topological characterization of asynchronous complexity. *SIAM Journal on Computing*, 36(2):457–497, 2006.

[149] Matthias Függer, Thomas Nowak, and Bernadette Charron-Bost. Diffusive clock synchronization in highly dynamic networks. In *2015 49th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–6, March 2015. doi: 10.1109/CISS.2015.7086841. URL http://ieeexplore.ieee.org/document/7086841/.

[150] Matthias Függer, Manish Kushwaha, and Thomas Nowak. Digital circuit design for biological and silicon computers. In Vijai Singh, editor, *Advances in Synthetic Biology*, pages 153–171. Springer, Heidelberg, March 2020. doi: 10.1007/978-981-15-0081-7_9. URL https://link.springer.com/chapter/10.1007%2F978-981-15-0081-7_9.

[151] Da-Jung Cho, Matthias Függer, Corbin Hopper, Manish Kushwaha, Thomas Nowak, and Quentin Soubeyran. Distributed computation with continual population growth. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing (DISC)*, volume 179 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISBN 978-3-95977-168-9. doi: 10.4230/LIPIcs.DISC.2020.7. URL https://drops.dagstuhl.de/opus/volltexte/2020/13085/.

[152] Alec AK Nielsen, Bryan S Der, Jonghyeon Shin, Prashant Vaidyanathan, Vanya Paralanov, Elizabeth A Strychalski, David Ross, Douglas Densmore, and Christopher A Voigt. Genetic circuit design automation. *Science*, 352(6281), 2016.

[153] Uri Alon. *An introduction to systems biology: design principles of biological circuits*. CRC press, 2019.

[154] Bruce Alberts. Molecular biology of the cell. 2018.

[155] J. J. Y. Teo, S. S. Woo, and R. Sarpeshkar. Synthetic biology: A unifying view and review using analog circuits. *IEEE Transactions on Biomedical Circuits and Systems*, 9(4):453–474, 2015. doi: 10.1109/TBCAS.2015.2461446.

[156] Chris J Myers. *Engineering Genetic Circuits*. Chapman and Hall/CRC, April 2016. doi: 10.1201/b16381. URL https://doi.org/10.1201/b16381.

[157] Jennifer AN Brophy and Christopher A Voigt. Principles of genetic circuit design. *Nature methods*, 11(5):508–520, 2014.

[158] Alain J Martin. Programming in vlsi: From communicating processes to delay-insensitive circuits. Technical report, California Inst of Tech Pasadena Dept of Computer Science, 1989.

[159] Charles Antony Richard Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[160] Ho-Lin Chen, David Doty, and David Soloveichik. Deterministic function computation with chemical reaction networks. *Natural computing*, 13(4):517–534, 2014.

[161] Eric E Severson, David Haley, and David Doty. Composable computation in discrete chemical reaction networks. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 14–23, 2019.

[162] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

[163] James Aspnes and Eric Ruppert. An introduction to population protocols. *Middleware for Network Eccentric and Mobile Applications*, pages 97–120, 2009.

[164] David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*, 31(4):257–271, 2018.

[165] Janna Burman, David Doty, Thomas Nowak, Eric E Severson, and Chuan Xu. Efficient self-stabilizing leader election in population protocols. *arXiv preprint arXiv:1907.06068*, 2019.

[166] Joffroy Beauquier, Janna Burman, Julien Clément, and Shay Kutten. On utilizing speed in networks of mobile agents. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 305–314, 2010.

[167] Daniel T Gillespie. Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, 81(25):2340–2361, 1977.

[168] Sheldon M Ross. *Introduction to probability models*. Academic press, 2014.

[169] Miles W Gander, Justin D Vrana, William E Voje, James M Carothers, and Eric Klavins. Digital logic circuits in yeast with crispr-dcas9 nor gates. *Nature Communications*, 8(1): 1–11, 2017.

[170] Javier Santos-Moreno, Eve Tasiudi, Joerg Stelling, and Yolanda Schaerli. Multistable and dynamic crispri-based synthetic circuits. *Nature Communications*, 11(1):1–8, 2020.

[171] Shuyi Zhang and Christopher A Voigt. Engineered dcas9 with reduced toxicity in bacteria: implications for genetic circuit design. *Nucleic acids research*, 46(20):11115–11125, 2018.

[172] Sarah Guiziou, Federico Ulliana, Violaine Moreau, Michel Leclere, and Jerome Bonnet. An automated design framework for multicellular recombinase logic. *ACS synthetic biology*, 7 (5):1406–1412, 2018.

[173] Javier Macía, Francesc Posas, and Ricard V Solé. Distributed computation: the new wave of synthetic biology devices. *Trends in biotechnology*, 30(6):342–349, 2012.

[174] Jens Sparsø and Steve Furber. *Principles asynchronous circuit design*. Springer, 2002.

université
PARIS-SACLAY

Titre : Le calcul à la frontière des abstractions : la puissance des circuits temporisés, non binaires, distribués

Mots clés : modèles de calcul, circuits, calcul distribué, timing, métastabilité, robustesse

Résumé : La conception et l'analyse des dispositifs informatiques directement mis en œuvre dans le matériel sont généralement basées sur les automates finis. Dans ce travail, nous passons en revue certaines des hypothèses faites dans ces conceptions et discutons des techniques pour les cas où les hypothèses ne sont pas valables.

Ce travail se concentre sur trois aspects : (i) le calcul dans des conditions où les effets temporels ne peuvent être négligés, (ii) lorsque les signaux non binaires jouent un rôle central, et (iii) le calcul dans une infrastructure changeante, i.e., les réseaux dynamiques. Bien que la majeure partie de ce travail soit consacrée aux implémentations en silicium, les circuits microbiologiques sont abordés dans les perspectives.

Title : Computing at the border of abstractions: the power of timed, non-binary, distributed circuits

Keywords : models of computation, circuits, distributed computing, timing, metastability, robustness

Abstract : The design and analysis of low-level computing devices directly implemented in hardware is commonly based on finite state machine models.
In this work we review some of the assumptions made in these designs and discuss techniques for the cases where the assumptions fail to hold.

The work concentrates on three such aspects: (i) computing under conditions where low-level timing effects cannot be neglected, (ii) when non-binary signals play a central role, and (iii) computing within a changing infrastructure, i.e., dynamic networks.
While most of this work is devoted to implementations in silicon, microbiological circuits are discussed in the outlook.