Synchronizer-free Digital Link Controller

Johannes Bund, Matthias Függer, Christoph Lenzen, Moti Medina

Synchronizer-free Digital Link Controller

Johannes Bund, Matthias Függer, Christoph Lenzen, Moti Medina

***Abstract*—This work presents a producer-consumer link between two independent clock domains. The link allows for metastability-free, low-latency, high-throughput communication by slight adjustments to the clock frequencies of the producer and consumer domains steered by a controller circuit.**

Any such controller cannot deterministically avoid, detect, nor resolve metastability. Typically, this is addressed by synchronizers, incurring a larger dead time in the control loop. We follow the approach of Friedrichs et al. (TC 2018) who proposed metastability-containing circuits. The result is a simple control circuit that may become metastable, yet deterministically avoids buffer underrun or overflow. More specifically, the controller output may become metastable, but this may only affect oscillator speeds within specific bounds. In contrast, communication is guaranteed to remain metastability-free.

We formally prove correctness of the producer-consumer link and a possible implementation that has only small overhead. With SPICE simulations of the proposed implementation we further substantiate our claims. The simulation uses 65 nm process running at roughly 2 GHz.

***Index Terms*—producer-consumer link, digital controllers, continuous processes, metastability-free, metastability-containing mixed signal control loop**

I. Introduction

Links that enable communication between different clock domains are an important ingredient in every Globally Synchronous Locally Asynchronous (GALS) system [1]. This communication is performed in a "producer-consumer" manner: in one clock domain the producer pushes messages to the link, while in the other clock domain the consumer pulls messages from the other side. Inherently, link implementations are susceptible to failures induced by metastable upsets; even if such errors can be handled, they negatively impact the performance of the link.

Previous digital controller designs resort to different methods to deal with metastability: clock-masking [2], clock-pausing [3], [4], or adding synchronizers (while sacrificing latency) to maintain a realistic (yet finite) mean time between failures (MTBF) of the link [2], [5]–[8]. Downsides of these approaches are that synchronized fill level flags are inherently "stale" by the time they affect the system. This requires almost-full flags [2], long handshake latencies that increase the dead time and affect the latency and throughput, additional slack in a controller cycle accounting for metastability resolution time in controller's flip-flops or mutual exclusion (MUTEX) elements [7], [8].

TABLE I: 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, [6], and [10].

		this work	[10]	[6]
Performance	Latency [ns]	1	375	1.3
	Th.put [$\frac{\text{pkt}}{\text{ns}}$]	2	$\frac{1}{41}$	$\frac{1}{1.3}$
	MTBF	∞	∞	Finite
Overhead	N	2	9	2
	# Gates	8	> 100	> 100
	# FFs	4	> 50	> 100
	Osc. Type	tune	distr.	quartz

At the heart of the problems faced in these controllers lies the impossibility to solve discrete decision problems, e.g., writing to a cell at a certain clock tick or skipping a clock cycle, under continuous inputs (i.e., arbitrary phase shifts between producer and consumer clocks) within bounded time [9]. One way out of this impossibility is to resort to *end-to-end analog designs*, e.g., by letting an analog controller apply continuous phase shifts by (slightly) tuning the producer and/or consumer oscillator. This comes at the burden of a fully-fledged analog design.

An interesting alternative was proposed in [11], where the authors advocate the use of asynchronous controllers, sensing and controlling analog processes. With this approach, analog components are required at the controller interfaces only, and the controller itself is implemented by a digital asynchronous circuit. For certain classes of controllers, this approach allows to completely circumvent metastable upsets within the controller circuit, essentially by allowing for the occurrence of (digital) controller outputs within a continuous time range, rather than at discrete clock ticks only.

Contribution: We propose a fundamentally different approach, exemplifying it at the hand of highly efficient link controllers: like [11], we replace large parts of a (conceptually) analog controller by standard digital circuitry. However, we do *not* resort to asynchronous circuits. Instead, we allow unstable/metastable signal values within our circuit and treat them as a third "logical" value. Clearly, care must be taken that such values do not "infect" the whole controller logic, leading to unconstrained control outputs. For this purpose, we follow [12], using the same worst-case propagation model and analysis to provably contain metastability.

Specifically, we propose a digital controller that drives tunable ring oscillators as presented in [13] at the sender and receiver side and prove its correctness. The controller is small

Johannes Bund and Christoph Lenzen are with the Max Planck Institute for Informatics, Saarland Informatics Campus, Germany. Their work on this project has been supported by funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 716562). Email: {jbund,clenzen}@mpi-inf.mpg.de

Johannes Bund is with the Saarbrücken Graduate School of Computer Science, Germany.

Matthias Függer is affiliated with the CNRS & LSV, ENS Paris-Saclay, Université Paris-Saclay & Inria, France. He receives funding from DigiCosme and DEPEC MODE. Email: mfuegger@lsv.fr

Moti Medina is with the Ben Gurion University of the Negev, Israel. He was partially supported by the Israel Science Foundation grant No. 867/19. Email: medinamo@bgu.ac.il

Manuscript received Mmmm DD, YYYY; revised Mmmm DD, YYYY.

in size, has low control latency and allows for small link buffers. We show that this guarantees high throughput and low latency communication. Most notably, while the controller may become metastable, we ensure that metastability is contained within the controller, and does not lead to metastable upsets, corruption, or drops of communicated data words in the ring buffer between the sender and receiver. We complement our provable system guarantees with simulations (see Sec. IV).

Related Work and Comparison: There is a large body of work on links between clock domains, motivated by their central importance in GALS designs. According to [1], GALS systems can be classified by their clocking schemes: (i) pausible clocked systems, (ii) asynchronous systems with uncorrelated clocks, and (iii) loosely synchronous systems, with (partially) synchronized clocks. We shortly review sender-receiver communication in these three approaches.

(i) *Pausible clocking* overcomes synchronization issues by halting the clock until metastability is resolved [3]; e.g., the design in [4] guarantees no glitches on stopping and starting. Metastability inside the control loop may lead to an arbitrary delay of the final pulse on stopping. This requires that the clock cannot be started again before metastability has been resolved.

(ii) *Uncorrelated clocks:* Communication between uncorrelated clock frequencies and phases is traditionally done by combining classical two-flop synchronizers with buffers and flow-control circuitry. A downside of these approaches is that the latency and the throughput are determined by the handshake cycle that has to include (at least) two synchronizer cycles at both sides. Clearly, also this approach has a non-zero upset probability and thus finite MTBF. In [5], a mixed-clock first-in first-out pipeline (FIFO) with flow control logic is proposed. Instead of classical handshaking, synchronized full/empty and almost full/empty signals are used. The 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. The approach has finite MTBF. In [2], a ripple FIFO solution with almost full/empty signals is proposed. The approach requires slow sender/receiver speeds compared to data propagation within the ripple FIFO. Moreover, full/empty flags have to be synchronized, which leads to increased latency and finite MTBF. In [7], a locally delayed latching (LDL) approach is proposed: conflicting read/write operations are delayed by an asynchronous controller with a MUTEX element. Controller latency is in the order of 20 gate delays, and the minimum feasible clock cycle is no less than 69 gate delays, accounting for enough time for the MUTEX to stabilize with high probability. Gradual synchronization [8] allows fine-grained interweaving of synchronization and computation, also shifting conflicting ripple FIFO requests by MUTEX elements at each stage. Like synchronizer chains, this approach has finite MTBF that can be increased at the cost of higher latency. Dally and Tell [6] propose a scheme in which the MTBF can be made arbitrarily large without increasing latency. They use synchronizers to continually determine phase offsets between sender and receiver clocks only. A drawback is that the frequency and phase measurement circuits require accurate phase tracking (64bit in their implementation) and can account for slow phase drifts only.

(iii) *Loosely synchronous systems:* in contrast to (i) and (ii), synchronizing clocks allows obtaining worst-case guarantees on latency and throughput together with provable absence of metastable upsets. Our approach also falls into this class. The closest work to our approach presumably is proposed in [10]. By using the distributed DARTS clock generation mechanism [14], 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.

Table I shows a comparison of our controller with the most closely related works, [10] and [6] (cf. Section IV for details).

Our link controller has some similarities to a phase locked loop (PLL) with an all-digital phase detector; see e.g. [15], [16] for all-digital PLL designs. We briefly summarize commonalities and differences in the following.[1]

Classical PPLs lock a slave clock to a typically more stable master clock. In our case we do not distinguish between a slave and a master, but our controller treats both receiver and sender clocks equally; one might think of this as a "peer-to-peer PLL". The reason is that our goal is not to stabilize the absolute frequency of a poor clock by ensuring a bounded phase offset to a more stable master clock, but rather to bound the phase offset between a sender and receiver clock of similar quality. Additionally we provide lower and upper bounds on the frequency of the clocks which are close to the frequency bounds free-running oscillators of the same quality have. For example, this is useful when communicating with the environment.

The initial stage of a classical PLL is a phase frequency detector (PFD), which measures the phase difference between the master and slave clock signals. Designs range from conventional PFDs, which measure negative and positive phase offsets on separate binary output signals by producing pulses whose width is the negative/positive phase offset, to more advanced setups [17]. Phase differences are then either forwarded to charge pumps (analog PLLs) [18] or converted to digital counter offsets (digital PLLs). For the latter, an unstable phase difference poses a risk for increased power consumption and likelihood of metastable upsets; see [16], where a filter on phase difference signals for a low-power digital PLL is proposed.

In our case, there is a (digital) unary-encoded up/down-counter at the heart of the controller, allowing to measure the phase difference between both clocks. Note that since our goal is not to lock to a highly stable oscillator, our design is much simpler: our circuit only determines whether the actual phase offset is larger or smaller than the desired phase offset. It is also worth noting that, while our oscillators are analog components, our circuit relies on the ability to switch between "fast" and "slow" only. This binary decision may become un-

[1]We remark that the exposition does not rely on the information given in this comparison, and it might be easier to follow the comparison based on a more detailed understanding of our approach. Accordingly, readers should feel free to skip to the next section and return later if needed.

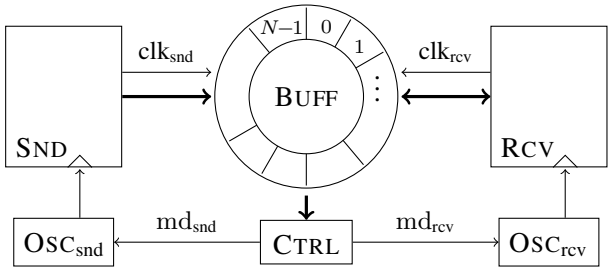


Fig. 1: Link with Digital Controller

or metastable frequently. In stark contrast to a classical digital PLL with a binary counter, this does not pose a problem for our design. We ensure that the potentially metastable output signal of our controller is only used to control the oscillator. The oscillator frequency is required to remain in the range spanned by the frequencies possible under stable operation (slow and fast mode) in presence of a metastable, or in general, unstable signal. This is the case for starved inverter ring oscillators.

The use of local clocks in our design has a further advantage over locking to a centralized clock that is assumed to provide a highly stable frequency reference. In our system the sender and receiver are not impaired by the failure of the respective other's clock. While correct communication between the two nodes inherently requires both oscillators to work correctly, our design guarantees that if one of the oscillators fails, the respective other keeps running within the same frequency bounds. Potential top-level error-detection based on the (non-)communicated data then provides adequate application-specific reaction to such scenarios.

Organization of the paper: We start with presenting the problem of communication in a system of two nodes with controllable oscillators in Section II. We then break the system down into modules, formally specifying their requirements. Section III discusses gate-level implementations of the modules, together with proofs that the implementations satisfy the formal requirements. In Section IV, we present simulations of our implementation at gate-level (VHDL) and transistor level (Spice). The simulation results are consistent with our formally proven results, and allow to obtain detailed performance metrics. We conclude in Section V.

II. System Specification and Model

We specify the system requirements and functionality next. The link (see Fig. 1) has three parts: (i) tunable oscillators OSC_{snd} and OSC_{rcv}, (ii) a (ring) buffer BUFF, and (iii) a buffer controller CTRL. The link enables communication between two parties, a sender SND and a receiver RCV, that interact with the link via prescribed interfaces, discussed later on.

The sender writes data to a ring buffer of even size $N > 0$, which is read by the receiver. Cells are numbered from 0 to $N-1$. Read and write access is clocked: following transitions of its clock clk_{snd}, the sender writes to the ring buffer. The register address is specified by the current value of its address pointer, which it subsequently increments (modulo N); likewise, following transitions of its clock clk_{rcv}, the receiver reads from its current address and subsequently increments its pointer.

We remark that our design can easily be altered for bidirectional communication. Each party needs to perform a read/write sequence instead of just a read (RCV) (respectively write (SND)) operation when it is accessing a buffer cell; the only effect is that the respective higher access time needs to be respected in the timing constraints on the system. For ease of presentation, we stick to the asymmetric setting in the following.

A. Local Clocks

Sender and receiver clocks clk_{snd} and clk_{rcv} are derived from clock sources OSC_{snd} and OSC_{rcv}, respectively. We require that these clock sources (or oscillators) are *tunable in frequency* by the *mode signals* md_{snd} and md_{rcv}.

Denote by $C(t) \in \mathbb{Z}$ a discrete clock value at wall-clock time $t \in \mathbb{R}_0^+$. This discrete clock is derived from a continuous clock $c(t) \in \mathbb{R}$ as $C(t) \triangleq \lfloor c(t) \rfloor$, with current frequency $\dot{c}(t)$. Let $C_s(t), C_r(t)$ be the discrete clock values of sender and receiver at wall-clock time t, and $c_s(t), c_r(t)$ their continuous clocks. For properly chosen $T_{\text{osc}} \geq 0$ and $\delta \leq 1$, we require:

(**C1**) We assume that the clocks are started roughly at the same time:[2] $c_s(0), c_r(0) \in (-\delta, 0]$.

(**C2**) If md_{snd} (md_{rcv}) is constantly 0 during $[t - T_{\text{osc}}, t]$, the sender (receiver) is in *slow mode* at time t and $\dot{c}_s(t) \in [s^-, s^+]$ ($\dot{c}_r(t) \in [s^-, s^+]$).

(**C3**) If md_{snd} (md_{rcv}) is constantly 1 during $[t - T_{\text{osc}}, t]$, the sender (receiver) is in *fast mode* at time t and $\dot{c}_s(t) \in [f^-, f^+]$ ($\dot{c}_r(t) \in [f^-, f^+]$).

(**C4**) If md_{snd} (md_{rcv}) is neither constantly 0 nor constantly 1 during $[t - T_{\text{osc}}, t]$, the respective clock is *unlocked* and $\dot{c}_s(t) \in [s^-, f^+]$ ($\dot{c}_r(t) \in [s^-, f^+]$).

(**C5**) Clocks in slow mode are never faster than clocks in fast mode: $s^+ \leq f^-$.

Here, T_{osc} is the response time of the tunable oscillator. Note that our requirements on the oscillator are fairly weak, making it easy to implement (cf. Section IV): Only if the stable control signal is stable for T_{osc} time, the oscillator needs to guarantee the respective rate. At any other time, it is not locked to a fixed frequency mode and may run at any rate between the slowest and fastest possible. This unlocked mode may be entered when the control signal is ambiguous or transitioned recently, i.e., when both parties are almost perfectly synchronized. The last condition is a minimal requirement ensuring that the phase offset between the two clocks cannot increase further when a clock in fast mode is chasing a clock in slow mode.

B. Buffer Access Specification

Next, we specify buffer access in an abstract model with few parameters. We assume that access to a buffer cell starts when the respective clock modulo N (possibly with a fixed offset) equals the buffer index. Note that this is a normalization of the time axis so that one computational cycle takes 1 unit of

[2]For $\delta = 1$, this is a fairly weak constraint. If sender and receiver each access one element of the ring buffer per clock cycle, it means that both oscillators are started within one clock cycle of each other. However, smaller values of δ may reduce the minimum feasible ring size by 2 in some cases.

"local" time as measured by the sender or receiver oscillator, respectively.[3] A computational cycle is defined by the local time between accessing consecutive buffers.

Intuitively, a buffer cell is *valid* (i.e., ready to be read) if it contains stable, logical data and is currently not written. A buffer cell is *invalid* (i.e., ready to be written) if it is not valid and currently not read. Formally:

(**B1**) We define the receiver's (discrete) address pointer as $P_r(t) \triangleq \lfloor p_r(t) \rfloor \bmod N = C_r(t) \bmod N$, where the receiver's (continuous) address pointer is $p_r(t) \triangleq c_r(t)$. That is, the receiver starts to access cell ℓ at each time t when $P_r(t) = p_r(t) \bmod N = \ell$.

(**B2**) We define the sender's address pointer to be $P_s(t) \triangleq \lfloor p_s(t) \rfloor \bmod N$, where $p_s(t) \triangleq c_s(t) + N/2$. That is, the sender pointer has a (nominal) offset of half the ring size relative to the receiver pointer. In the following, we will simply drop the "starts to" and say that the receiver (sender) *accesses cell ℓ at time t* if $p_r(t) \bmod N = \ell$ ($p_s(t) \bmod N = \ell$).

(**B3**) Read and write operations take non-zero time. We account for setup/hold times and latency by parameters τ_s and τ_r, which denote the maximum "durations" of write and read operations. Concretely, if the sender accesses a cell at time t, the receiver must not do so during $[t, t+\tau_\mathrm{s})$, and if the receiver accesses a cell at time t, the sender must not do so during $[t, t+\tau_\mathrm{r})$.

(**B4**) On initialization, cells $0 \le \ell < N/2$ are valid, while cells $N/2 \le \ell < N$ are invalid. If the sender accesses an invalid cell at time t, the cell *becomes valid* at time $t + \tau_\mathrm{s}$. If the reader accesses a valid cell at time t, it *becomes invalid* at time $t + \tau_\mathrm{r}$. This inductively defines for each cell and each time $t \ge 0$ whether it is valid or invalid.

Note that these definitions are crafted in such a way that if the sender accesses only invalid cells and the reader accesses only valid cells, we have mutual exclusion of read and write operations and for each individual cell, reads and writes alternate. This is the intended mode of operation, which we will formalize in Section II-F.

C. Metastability

To minimize dead time of the control loop regulating the clock speeds, we do not make use of synchronizers. Forgoing their use can result in meta-/unstable signals. At any point in time, a signal has a value in $\{0, \mathtt{M}, 1\}$, where $\mathtt{M}$ means that a signal is potentially metastable or in transition. We employ a worst-case analysis, which assumes that $\mathtt{M}$ propagates whenever possible; only explicit logical masking may protect from metastability, no probabilistic statements are used.

In particular, a flip-flop latching when its input is $\mathtt{M}$ will "store" an $\mathtt{M}$ until is latched again with a stable input. Note that an output signal may also be unstable due to a transitioning signal, e.g. after latching a new value different from the previously stored one.

[3]Note that this will typically not be 1 unit of "absolute" time, as oscillator speeds may vary.

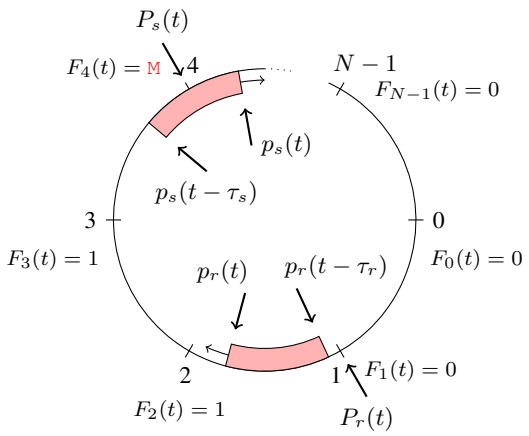


Fig. 2: Ring buffer access at time t. The sender currently accesses cell 4. Hence, its full/empty flag is $\mathtt{M}$. The receiver has just finished accessing cell 1. Thus, its full/empty flag is 0. In executions, we mark (potential) in-/metastability in red, cf. Fig. 8.

D. Link Controller Interface Specification

The mode signals themselves are generated by the controller CTRL. Controller decisions are based on full/empty flags of the ring-buffer cells, which we will describe shortly. We stress that, inherently, the controller acts at the border of two clock domains. Any digital implementation (including ours) is thus susceptible to metastable upsets. Accordingly, the voltage levels of md_snd and md_rcv may become meta-/unstable (between logical 0 and 1, denoted by $\mathtt{M}$), as, in order to minimize delay, we do *not* pipe them through a synchronizer chain before making use of them.

Let T_ctr denote the maximum end-to-end delay of the controller circuit, i.e., between its input (the full/empty flags) and its output (md_snd and md_rcv). The specification of the link controller's interface is as follows:

(**L1**) If for $t \ge T_\mathrm{ctr}$ the controller circuit specification maps the inputs during $[t - T_\mathrm{ctr}, t]$ continuously to 1 for signal md_snd, then $\mathrm{md}_\mathrm{snd}(t) = 1$; analogous statements hold for output 0 as well as signal md_rcv and outputs 0 and 1, respectively.

(**L2**) In all other cases, the output at time t is arbitrary, i.e., any value from $\{0, \mathtt{M}, 1\}$.

E. Full/Empty Flags

With each buffer cell ℓ, we associate a full/empty flag F_ℓ. It is specified as

(**F1**) $F_\ell(t) = 1$ if the cell is valid at time t and it either has not been accessed yet or the most recent access to it was by the sender;

(**F2**) $F_\ell(t) = 0$ if the cell is invalid at time t and it either has not been accessed yet or the most recent access to it was by the receiver;

(**F3**) if neither applies at time t, then $F_\ell(t) \in \{0, \mathtt{M}, 1\}$.

In other words, we allow for the possibility that $F_\ell(t) = \mathtt{M}$ at any point in time during read and write operations.

Algorithm 1 Controller CONTTH($\mathcal{T}$)

At each time $t \geq 0$ **do:**
1: $\mathrm{md}_{\text{rcv}}(t) \leftarrow$ choose arbitrarily in $\{0, \mathtt{M}, 1\}$
2: $\mathrm{md}_{\text{snd}}(t) \leftarrow$ choose arbitrarily in $\{0, \mathtt{M}, 1\}$
3: **if** $c_s(t) - c_r(t) \geq \mathcal{T}$ **then**
4: $\quad \mathrm{md}_{\text{rcv}}(t) \leftarrow 1$ // recall $\mathit{fill}(t) = N/2 + c_s(t) - c_r(t)$
5: $\quad \mathrm{md}_{\text{snd}}(t) \leftarrow 0$
6: **end if**
7: **if** $c_r(t) - c_s(t) \geq \mathcal{T}$ **then**
8: $\quad \mathrm{md}_{\text{rcv}}(t) \leftarrow 0$
9: $\quad \mathrm{md}_{\text{snd}}(t) \leftarrow 1$
10: **end if**

Fig. 2 depicts the state of the above described cell pointers at time t. Observe that all cells between the sender and the receiver are full and thus their full/empty flags equal to to 1, those between the receiver and the sender are empty with full/empty flags equal to 0, and the flags of those currently accessed are $\mathtt{M}$.

F. System Correctness

Expressing the correct order of and separation in time between cell accesses, we can now succinctly state what correct operation of the link architecture means.

Definition 1. *A link is* correct *if the following holds in any execution adhering to our model.*

(P1) *No underrun: the receiver accesses only valid cells.*
(P2) *No overflow: the sender accesses only invalid cells.*

Definition 2. *Controller* CTRL *is* correct *if it computes the signals* md_{snd} *and* md_{rcv} *out of the inputs* F_ℓ *so that the link is correct.*

The goal is now to design a (simple) controller that is correct even if the ring size N is small: this minimizes both the size of the buffer and its latency.

III. CONTINUOUS THRESHOLD CONTROLLER

Our control algorithm CONTTH($\mathcal{T}$) is specified in Alg. 1. It is parametrized by $\mathcal{T} \in \mathbb{R}^+$. In the remainder of this section, we explain the intuition behind the approach.

For the purpose of exposition, denote by $\mathit{fill}(t) \triangleq p_s(t) - p_r(t) = N/2 + c_s(t) - c_r(t)$ the fill level of the buffer. Recall that one of our design goals is to have a simple digital controller. The most straightforward choice for such a control algorithm is presumably the threshold controller: If the fill level of the ring buffer is larger than $N/2$, the sender is forced to slow and the receiver to fast mode. If the fill level is less than $N/2$, the sender and receiver are forced into fast and slow mode, respectively.

However, as the various involved circuit components incur non-zero delays, we cannot expect instantaneous (and thus also not exact) information on the fill-level. Also, changing the oscillators' speeds takes non-zero time, so we cannot hope for an immediate response to a small/large fill-level. Alg. 1 takes this into account by introducing *two thresholds*. Fig. 3 shows an execution where the controller CTRL runs the algorithm.

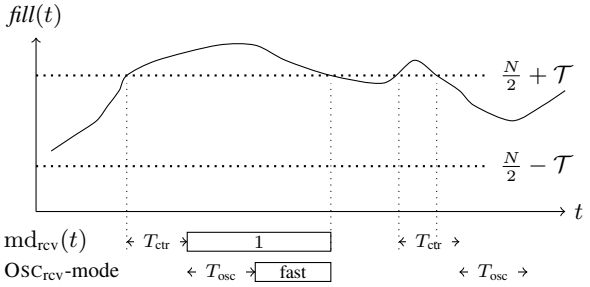


Fig. 3: CONTTH($\mathcal{T}$)'s signals of the receiver. The fill-level increases until it hits $\frac{N}{2} + \mathcal{T}$, which makes the md_{rcv} signal drive 1 after T_{ctr} time. After another T_{osc} time, the receiver and sender clocks are required to run in fast and slow mode, respectively (cf. Section II). Note that the second phase during which the threshold $\frac{N}{2} + \mathcal{T}$ is crossed is too short for CTRL and the oscillators to react with certainty.

A. Correctness of CONTTH($\mathcal{T}$)

Before we show that, for a $\mathcal{T}$ that is chosen sufficiently large, CONTTH($\mathcal{T}$) is *implementable* by a digital circuit in Section III-B, we show that CONTTH($\mathcal{T}$) indeed is *correct* (as per Definition 2) if $\mathcal{T}$ is chosen small enough.

Theorem 3. CONTTH($\mathcal{T}$) *is correct if*

$$\begin{aligned} \delta &\leq \mathcal{T} \\ &\leq N/2 - (f^+ - s^-)(T_{osc} + T_{ctr}) - f^+ \max\{\tau_s, \tau_r\}. \end{aligned} \tag{1}$$

Recall that $p_r(t) = c_r(t)$ and $p_s(t) = c_s(t) + N/2$. Thus, when perfectly synchronized, the sender and receiver concurrently access opposite cells of the buffer. The first subtrahend accounts for the fact that the clocks remain unconstrained for $T_{\text{osc}} + T_{\text{ctr}}$ time even after a threshold is reached: the controller guarantees corresponding output only after T_{ctr} time, which is bound to affect clock speeds at most another T_{osc} time later; during this time period, one clock may "catch up" to the other at rate $f^+ - s^-$. The second subtrahend accounts for the fact that the sender must always access a cell at least τ_r time before the receiver, while the receiver must do so τ_s time before the sender (B3).

Note that these two conditions become fully symmetric when using $\max\{\tau_s, \tau_r\}$ as the minimum required separation between accesses. Translating this wall-clock time difference to the address pointers using the upper bound of f^+ on clock frequencies, we see that the following lemma is the key to showing Theorem 3.

Lemma 4. *If Eq.* (1) *holds, then*

$$\forall t \in \mathbb{R}_0^+ : |c_s(t) - c_r(t)| \leq N/2 - f^+ \max\{\tau_s, \tau_r\}\,.$$

Proof. Assume for contradiction that $c_s(t) - c_r(t) > N/2 - f^+ \max\{\tau_s, \tau_r\} > \mathcal{T}$ for some time t. Let $t_0 \in \mathbb{R}_0^+$ be the minimal time such that $c_s(\tau) - c_r(\tau) \geq \mathcal{T}$ for all $\tau \in [t_0, t]$; as $|c_s(0) - c_r(0)| < \delta \leq \mathcal{T}$ by (C1) and (1) and both c_s and c_r are continuous, such a time t_0 must exist. Observe that $c_s(t_0) - c_r(t_0) = \mathcal{T}$.

By the specification of the controller (L1), we have that $\mathrm{md}_{\text{snd}}(\tau) = 0$ and $\mathrm{md}_{\text{rcv}}(\tau) = 1$ for all $\tau \in [t_0 + T_{\text{ctr}}, t]$.

Thus, we have that $\dot{c}_r(\tau) \geq f^- \geq s^+ \geq \dot{c}_s(\tau)$ for all $\tau \in [t_0+T_{\text{ctr}}+T_{\text{osc}}, t]$ by the specification of the clocks ((C2), (C3), (C5), and (C6)). Recall that also $\dot{c}_r(\tau) \geq s^-$ and $\dot{c}_s(\tau) \leq f^+$ at all times τ by (C2) to (C5). If $t - t_0 \geq T_{\text{ctr}} + T_{\text{osc}}$, we can thus bound

$$\begin{aligned} c_s(t) - c_r(t) &= c_s(t_0) - c_r(t_0) + \int_{t_0}^{t} \dot{c}_s(\tau) - \dot{c}_r(\tau)\ d\tau \\ &\leq \mathcal{T} + \int_{t_0}^{t_0+T_{\text{ctr}}+T_{\text{osc}}} f^+ - s^-\ d\tau + \int_{t_0+T_{\text{ctr}}+T_{\text{osc}}}^{t} 0\ d\tau \\ &\leq \mathcal{T} + (f^+ - s^-)(T_{\text{ctr}} + T_{\text{osc}}) \overset{(1)}{\leq} \frac{N}{2} - f^+ \max\{\tau_s, \tau_r\}. \end{aligned}$$

If $t - t_0 < T_{\text{ctr}} + T_{\text{osc}}$, the second part of the integral vanishes and the first part becomes smaller, showing that the same bound holds. Either way, this contradicts our assumption that $c_s(t) - c_r(t)$ exceeds this bound.

Finally, we argue analogously for the case that $c_r(t) - c_s(t) > N/2 - f^+ \max\{\tau_s, \tau_r\}$, where the roles of sender and receiver are exchanged. □

Proof of Theorem 3. By Lemma 4,

$$\begin{aligned} |p_s(t) - p_r(t)| &= |c_s(t) + N/2 - c_r(t)| \\ &\in [f^+ \max\{\tau_s, \tau_r\}, N - f^+ \max\{\tau_s, \tau_r\}]. \end{aligned} \tag{2}$$

In particular, the (continuous) sender and receiver address pointers never have the same value modulo N and thus cannot "pass" each other. Moreover, by our assumptions on the initial clock values (C1), and since $\delta \leq 1$, we have that $c_s(0), c_r(0) \in (-1, 0]$, i.e., $p_r(0) \in (-1, 0]$ and $p_s(0) \in (N/2 - 1, N/2]$ by (B1) and (B2), respectively. Together with (B4), this implies that (i) the first access to each cell that is invalid at time 0 is by the sender, (ii) the first access to each cell that is valid at time 0 is by the receiver, and (iii) each cell is accessed alternatingly by sender and receiver.

It remains to show that the receiver does not access a cell less than τ_s time after a sender access to the same cell. Similarly, we need to show that the sender does not access a cell less than τ_r time after a receiver access. To this end, suppose cell ℓ is accessed by the sender and receiver at times t_s and t_r, respectively. Thus, $\ell = p_r(t_r) + aN = p_s(t_s) + bN$ for some $a, b \in \mathbb{Z}$, i.e.,

$$\begin{aligned} |p_r(t_r) - p_r(t_s)| &= |p_s(t_s) - p_r(t_s) + (b-a)N| \\ &\overset{(2)}{\geq} f^+ \max\{\tau_s, \tau_r\}. \end{aligned}$$

As $\dot{p}_r(t) = \dot{c}_r(t) \leq f^+$ at all times t by (C2) to (C5), we also have $|p_r(t_r) - p_r(t_s)| \leq f^+|t_r - t_s|$ and therefore $|t_r - t_s| \geq \max\{\tau_s, \tau_r\}$. Thus, (P1) and (P2) are satisfied for any access to cell ℓ; since ℓ was arbitrary, this completes the proof. □

B. Clocked Implementation ClockedTh

Next, we provide a simple and efficient controller implementation that works if $\mathcal{T}$ is sufficiently large. Recall that our goal is to detect when $c_s(t) - c_r(t) \geq \mathcal{T}$ or $c_r(t) - c_s(t) \geq \mathcal{T}$. By Lemma 4, assuming a correct implementation satisfying (1), it holds that the address pointers never reach each other. Together with the equality $c_r(t) - c_s(t) = p_r(t) + N/2 - p_s(t)$, it follows that all we need to check is whether one pointer is more or less than $N/2$ cells "ahead" of the other or not. This gives us an indication of whether the buffer is more or less than half full, and the more accurately we can decide, the smaller $\mathcal{T}$ can be for the implementation to be correct.

We use the receiver's clock to sample whether the sender's address pointer is currently by more or less than $N/2$ cells ahead of the receiver's address pointer.[4] This is where the full/empty flags come in handy. Instead of having to communicate and sample $c_s(t)$, the receiver simply samples the flag of cell $\ell + N/2 \bmod N$ when accessing cell $\ell \in [N]$. This occurs at each time t when $\ell = p_r(t) \bmod N = c_r(t) \bmod N$, which means that if the buffer is exactly half full, we had that $p_s(t) \bmod N = \ell + N/2 \bmod N$, i.e., the sender accesses cell $\ell + N/2 \bmod N$ at precisely the same time. This means that it starts setting the full/empty flag of the cell from 0 to 1 at time t, i.e., if the buffer is less than half full, the receiver will successfully sample a stable 0 into flip-flop ffa,[5] see Fig. 4.

In contrast, if the buffer is more than half full, it may be the case that the receiver "reads" an `M`, because the sender is still writing the full/empty flag. Only if it accessed the cell at the latest at time $t - \tau_s$, we can be certain that the result of the read operation is a stable 1. To avoid this asymmetry, we sample cell ℓ at times t when $c_r(t) \bmod N = \ell + f^+\tau_s/2$.

Lemma 5. *Suppose time t and $\ell \in [N]$ are such that $c_r(t) \bmod N = \ell + f^+\tau_s/2$ and Eq. (2) holds. Then*

(i) $c_s(t) - c_r(t) \geq f^+\tau_s/2 \Rightarrow F_\ell(t) = 1$, *and*
(ii) $c_r(t) - c_s(t) \geq f^+\tau_s/2 \Rightarrow F_\ell(t) = 0$.

Proof. We show (i) first, i.e., assume that $c_s(t) \geq c_r(t) + f^+\tau_s/2$. Then

$$p_s(t - \tau_s) = c_s(t - \tau_s) \geq c_s(t) - f^+\tau_s \geq c_r(t) - f^+\tau_s/2\,.$$

Note that

$$c_r(t) - f^+\tau_s/2 \bmod N = \ell\,,$$

i.e., the sender completed writing cell ℓ (for the most recent time) at time t; here, (2) shows that neither sender nor receiver cannot have accessed the cell again after the operation was complete. In other words, $F_\ell(t) = 1$, as claimed.

Now we show (ii). Thus, we assume that $c_s(t) \leq c_r(t) - f^+\tau_s/2$, while also

$$c_r(t) - f^+\tau_s/2 \bmod N = \ell\,.$$

Hence, the most recent access to cell ℓ was by the reader (again using also (2)), which also completed its access (as $N \geq 2$

[4]It is worth noting that one could use a purely combinational controller to achieve the same result, i.e., there is no need to rely on clocking. Making use of the clock does also not guarantee that stable values are sampled. However, making use of the clock results in a controller with smaller threshold value than a straightforward combinational implementation due to the known alignment of the sampling times with one of the clocks.

[5]For simplicity, we attribute any unstable reading to the transition of the memory flag of the cell via τ_s. However, of course the parameters of the flip-flop we sample into, quality of the clock signal, and the delay from the flag's output to the flip-flop's input through the MUX all have an effect. Based on a timing analysis of the circuit and adding a suitable phase shift to the clock input of ffs by, e.g., using a buffer, the abstract behavior we assume can be realized. τ_s then simply describes the size of the time window during which ffs is vulnerable to metastability induced by a transition of the memory flag of cell ℓ.

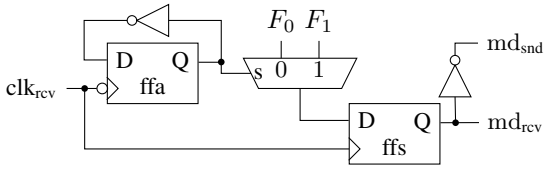


Fig. 4: Controller ClockedTh for ring-size $N = 2$. Flip-flop ffa stores the address (modulo 2) that is sampled and ffs the sampled full/empty flag.

and we assume that operations are completed within a single clock cycle). In other words, $F_\ell(t) = 0$, as claimed. □

Based on this idea, we derive a straightforward implementation of the controller. Put simply, the receiver samples the full/empty flag of the cell opposite to the one it currently reads in the ring. More precisely, $\mathrm{md}_{\mathrm{rcv}}$ is the output of a flip-flop (flip-flop 'ffs' in Fig. 4), into which the receiver samples $F_\ell(t)$ at times t such that $c_r(t) \bmod N = \ell + f^+\tau_s/2$. Signal $\mathrm{md}_{\mathrm{snd}}$ is obtained by negating $\mathrm{md}_{\mathrm{rcv}}$. A circuit implementing this approach for ring size $N = 2$ is shown in Fig. 4. Here, flip-flop ffa is a modulo 2 counter used to track the address to the current cell to sample. It is initialized to the opposite of the receiver address. We need to ensure that the MUX switches to forwarding the respective full flag before flip-flop ffs latches the output of the MUX. We do so by computing the select bit on the negated clock signal. This shifts the computation of the select bit by half a clock cycle and ensures correct timing. Note that here we might get metastable mode signals due to switching full flags. Naturally, it is necessary that the mode signal is computed within a single clock cycle; given the simplicity of the circuit, this is easily achieved.

In the following, denote by $\tau_{\max}$ the maximum propagation time through the circuit shown in Fig. 4 from the full/empty flags at the top to $\mathrm{md}_{\mathrm{snd}}$ (without τ_s, which is already taken into account by Lemma 5). Lemma 5 then characterizes the proposed controller.

Corollary 6. *Assume that the control circuit ClockedTh is used in accordance with Lemma 5 and that (P1) and (P2) hold until time $t > T_{ctr} = 1/s^- + \tau_{\max}$.*

(i) If for all $t' \in [t - T_{ctr}, t]$ we have that

$$c_s(t') - c_r(t') \geq f^+\tau_s/2\,,$$

then $\mathrm{md}_{rcv}(t) = 1$ and $\mathrm{md}_{snd}(t) = 0$.

(ii) If for all $t' \in [t - T_{ctr}, t]$ we have that

$$c_r(t') - c_s(t') \geq f^+\tau_s/2\,,$$

then $\mathrm{md}_{snd}(t) = 1$ and $\mathrm{md}_{rcv}(t) = 0$.

Proof. The outputs $\mathrm{md}_{\mathrm{rcv}}(t)$ and $\mathrm{md}_{\mathrm{snd}}(t)$ at time t are derived from the output of ffs at time t (or one inverter delay earlier). As the receiver clock runs at least at speed s^- (by (C2)–(C4)), flip-flop ffs is latched at least every $1/s^-$ time. Hence, taking into account the propagation time through the MUX and the definition of $\tau_{\max}$, the outputs correspond to the output of one of the flags at some time $t' \in [t - T_{\mathrm{ctr}}, t]$. As the MUX selects the flag output it forwards according to Lemma 5, we can apply the lemma to time t', yielding in Case (i) that a stable 1 is latched and in Case (ii) that a stable 0 is latched. This results in the desired corresponding circuit outputs $\mathrm{md}_{\mathrm{rcv}}(t) = 1$ and $\mathrm{md}_{\mathrm{snd}}(t) = 0$ (Case (i)) or $\mathrm{md}_{\mathrm{snd}}(t) = 1$ and $\mathrm{md}_{\mathrm{rcv}}(t) = 0$ (Case (ii)), respectively. □

We now can derive the correctness of the controller, expressed in Theorem 7, conditional on simple constraints on $\mathcal{T}$.

Theorem 7. *Assume that Eq.* (1) *holds, where $T_{ctr} = 1/s^- + \tau_{\max}$, and $\mathcal{T} \geq f^+\tau_s/2$. Then ClockedTh is an implementation of* ContTh($\mathcal{T}$).

Proof. If there is some access to a valid cell by the sender or to an invalid cell by the reader, there must be a minimal such time (because the start of a cell access is a discrete event). Denote by $\bar{t}$ the minimal such time if such an access occurs and set $\bar{t}$ to infinity otherwise.

We claim that the circuit implements ContTh($\mathcal{T}$) at all times $0 \leq t < \bar{t}$; from this we will infer the statement of the theorem. Recall that by (L1) and (L2), the controller implementation needs to output a specific (and stable) signal only if the condition in Line 3 or the one in Line 7 of Algorithm 1 continuously holds during the previous T_{ctr} time. According to Algorithm 1, this is the case at time t if and only if $c_s(t') - c_r(t') \geq \mathcal{T}$ for all $t' \in [t - T_{\mathrm{ctr}}, t]$ or $c_r(t') - c_s(t') \geq \mathcal{T}$ for all $t' \in [t - T_{\mathrm{ctr}}, t]$.

Consider such a time t. Note that $t > T_{\mathrm{ctr}}$, as $|c_s(0) - c_r(0)| < \delta \leq \mathcal{T}$ by (C1) and Eq. (1), i.e., neither condition is satisfied at time 0. We consider the two cases (i) $c_s(t') - c_r(t') \geq \mathcal{T}$ for all $t' \in [t - T_{\mathrm{ctr}}, t]$ and (ii) $c_r(t) - c_s(t) \geq \mathcal{T}$ for all $t' \in [t - T_{\mathrm{ctr}}, t]$.

Case (i): Since $\mathcal{T} \geq f^+\tau_s/2$, we may apply Case (i) of Corollary 6. We conclude that $\mathrm{md}_{\mathrm{rcv}}(t) = 1$ and $\mathrm{md}_{\mathrm{snd}}(t) = 0$.

Case (ii): In this case we may apply Case (ii) of Corollary 6, from which we deduce that $\mathrm{md}_{\mathrm{rcv}}(t) = 0$ and $\mathrm{md}_{\mathrm{snd}}(t) = 1$.

We conclude that the circuit meets the specification at all times $t < \bar{t}$. In particular, we can apply Lemma 4 at times $t < \bar{t}$, showing that $|c_s(t) - c_r(t)| \leq N/2 - f^+ \max\{\tau_s, \tau_r\}$. If $\bar{t} \neq \infty$, continuity of c_s and c_r implies that also $|c_s(\bar{t}) - c_r(\bar{t})| \leq N/2 - f^+ \max\{\tau_s, \tau_r\}$. Reasoning analogously to the proof of Theorem 3, it follows that (P1) and (P2) are not violated at times $t \leq \bar{t}$, contradicting the definition of $\bar{t}$. We conclude that $\bar{t} = \infty$, implying that the circuit from Fig. 4 indeed implements ContTh($\mathcal{T}$). □

Finally, we translate the theorem into a sufficient condition for correctness of the link implementation. To state its performance, we define the *latency* as the maximum 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 *throughput* is the guaranteed minimum rate of delivered packets; note that no packet drops or corruptions occur in our implementations.

Corollary 8. *For $\Delta = \lceil (f^+ - s^-)(T_{osc} + 1/s^- + \tau_{\max}) + f^+ \max\{\tau_s, \tau_r\} + \max\{\delta, f^+\tau_s/2\} \rceil$ and $N \geq 2\Delta$, the given clocked link implementation is correct with latency N/s^- and throughput $1/s^-$.*

Proof. Set $T_{\text{ctr}} = 1/s^- + \tau_{\max}$. We choose $\mathcal{T}$ such that (1) and $\mathcal{T} \geq f^+\tau_s/2$ are both satisfied. This is possible if and only if $N/2 \geq \Delta$, which holds by the prerequisites of the corollary. Then Theorem 7 yields that the circuit from Fig. 4 indeed implements CONTTH($\mathcal{T}$), and Theorem 3 shows that the implemented controller is correct.

The performance bounds follow immediately from correctness and the fact that the guaranteed minimum clock rate is s^-. □

IV. PERFORMANCE EVALUATION

We discuss a UMC 65 nm ASIC design operating at roughly 2 GHz for which we carried out simulations. This demonstrates that the derived performance bounds indeed lead to promising results.

In the section, we also demonstrate simulated executions that show the circuit behaving according to the specification, despite reoccurring metastability of its control signals; see Fig. 8. In fact metastability of the control signals is likely to be observed in an implementation, since by its attempt to synchronize the two oscillators, the controller repeatedly drives the control signals into metastability; much like experimental setups to measure deep metastability of synchronizers [19], [20]. We would like to point out that any such demonstration, however, does not replace the correctness proofs in Section III. Proving that metastability is not a problem would require to verify the absence of metastability (or resulting effects) in all circuit components, except for the places to which our proofs show metastability to be confined.

A. ASIC Implementation

The complete design is shown in Fig. 5. It comprises the digital controller (CTRL), tunable sender and receiver oscillator (OSC_{snd},OSC_{rcv}), and the ring buffer of size $N = 2$.

At a buffer size of 2 the address logic in SND and RCV reduces to a simple modulo 2 counter. Hence, we only have a single register for the sender and the receiver side. The modulo counter operates on the negated clock to ensure a stable output at the time a register in the buffer is accessed. The buffer consists of two buffer cells that store the full/empty-flags. The design of a buffer cell that can be set to 1 by one clock domain and reset to 0 by another clock domain is given in Fig. 6. The design uses a flip-flop for each clock domain that forwards its output to a XOR which computes the output. If the sender flip-flop is enabled it copies the negation of the receiver state. For differing states the XOR will output a 1. If the receiver enables its flip-flop the state of the sender is copied. Hence, the output of the buffer cell is reset to 0.

We can optimize the controller from Fig. 4, as we already compute the write address of the sender. We remove flip-flop ffa and read the address from the SND address logic. The multiplexer in CTRL is connected such that we sample from the buffer cell that is currently not written by the sender. The timing diagram in Fig. 7 shows the behavior of CTRL.

Recall that we require the sender and receiver *oscillators* to be well-behaved even when control bits are unstable. Specifically, we require that (i) oscillator frequencies are always within $[s^-, f^+]$, and (ii) frequency mode changes occur within T_{osc} time ((C2) to (C5)). This is why we resorted to starved-inverter ring oscillators that guarantee such behavior [21]; we designed the sender and receiver starved inverter rings at transistor level following [22]. Note that we do not need the full control logic overhead typically required to drive the starved inverter cells, since we only need two speeds: slow and fast. Hence, the control logic of the oscillator takes a single bit and adjusts the delay of the starved inverters according to fast or slow mode. As the receiver oscillator OSC_{rcv} additionally drives the control logic CTRL its load is higher than the load driven by the sender oscillator. The effect is that OSC_{rcv} has slightly slower fast and slow modes. The difference does not matter as long as the oscillator speeds lie within their theoretical bounds. One can keep the imbalance very small by decoupling the oscillators from the load with buffers.

Signals md_{rcv} and md_{snd} are used as control signals of the rings, which run at roughly 2 GHz and 2.3 GHz for input 0 and 1, respectively.

Extracting delay and frequency parameters from the standard cell library we get $\Delta = 1$ in Corollary 8, i.e., ClockedTh is provably correct for $N \geq 2$. This fits to the bounds given in Table I.

B. Frequency Stability of Tunable Oscillators

Typically, accuracy of oscillator frequencies is stated as a two-sided error, i.e., if the nominal frequency of the oscillator is f and it has a relative frequency error of at most r, then at any time its momentary frequency is between $(1-r)f$ and $(1+r)f$.

Recall from (C5) that we require that the fast oscillator mode is always faster than the slow oscillator mode. For a $2-2.3$ GHz clock we must therefore tune the clock within an error r that satisfies the condition $2 \cdot (1+r)^2/(1-r)^2 \leq 2.3$, i.e., $r \leq 3.49\%$ is a sufficient bound on the frequency error. In case these error margins would be too restrictive, we could choose a clock with larger gap between fast and slow modes, e.g., $2 - 2.5$ GHz. Depending on the outcome of the timing analysis (see also Corollary 8), this may require a larger buffer size N.

For comparison, the accuracy requirements for the oscillators used in [6] are as follows. If both the sender and receiver oscillator run at (roughly) the same nominal frequency, $\Delta p < g/S$ is proven to be sufficient for correctness of the design, where Δp is the relative phase change per clock cycle, $S = 4$ the number of synchronizer stages, and $g = 0.1$ the guard band. However, the proof assumes a perfectly stable receiver clock. If receiver and sender oscillator may drift, the above inequality becomes $2\Delta p(1 + \Delta p) < g/S$. This is equivalent to a frequency error of less than 1.24%.

C. Gate level and SPICE Simulations

We first ran gate-level VHDL simulations of designs of our ClockedTh controller with delay and setup/hold parameters from the ASIC design. The starved-inverter rings were simulated by forward Euler integration of a first order ODE model,

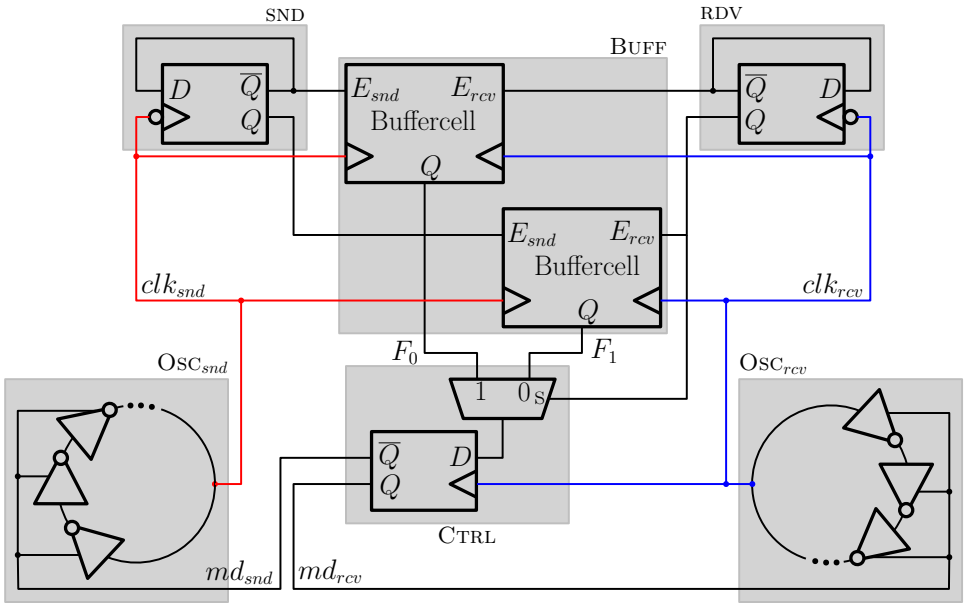


Fig. 5: Implementation of the system with buffer size $N = 2$. Clock regions are marked red (sender) and blue (receiver).

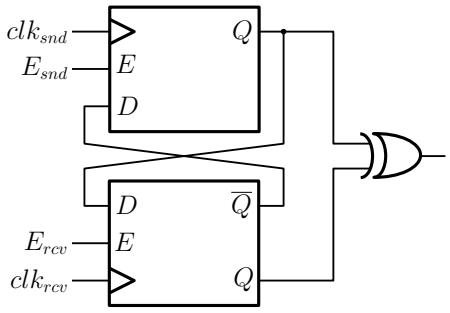


Fig. 6: Implementation of a buffer cell that can only be set by the sender and only be reset by the receiver.

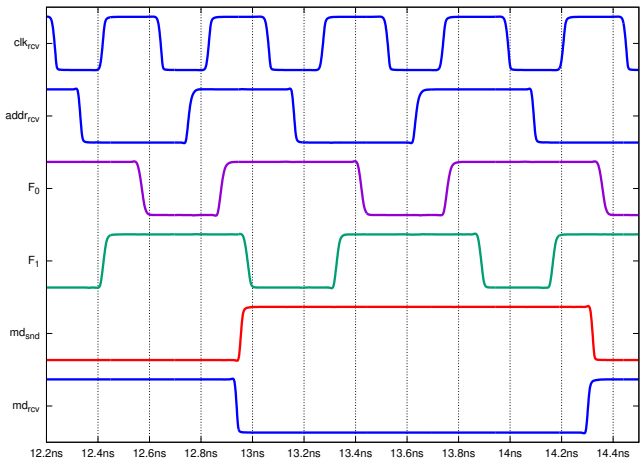


Fig. 7: Timing diagram of the controller CTRL. The address $addr_{rcv}$ decides which full-flag is sampled into the register of the controller at a rising clock transition.

where current clock rates are independently uniformly distributed in each integration step to account for drift. The high respectively low frequency of the starved inverter rings where set to 2.3 GHz respectively 2 GHz. Potential in-/metastability of signals was simulated by X in a worst-case manner; this includes flip-flops with setup/hold violations, full/empty-flags, and oscillator mode signals. Simulated traces were 5 ms (10^7 clock cycles) long and all in accordance with the proven correctness results. We stress that signals md_{rcv} and md_{snd} were unstable (X) almost all the time due to the conservative gate model assumptions, yet no buffer over-/underruns were encountered; cf. Fig. 8.

We then ran Spice simulations for the ClockedTh design: The design was implemented in Spice using standard cells and parameters of the UMC 65 nm library combined with an implementation of a tunable ring oscillator. The oscillator runs at speed 2.09 GHz in slow mode and 2.42 GHz in fast mode. Taking into account timing constraints and propagation delays of the elements we can use a ring buffer of size two, according to Corollary 8.

When simulating the design for 500 ns (about 1100 clock cycles) no faulty behavior could be detected. However, the simulation confirms what we stressed previously. In almost 50% of the cases the setup time of ffs (see Fig. 4) is violated due to late transition of the full flags. Still the controller behaves correct and the two oscillators run synchronously.

Fig. 9a shows the full flags of the buffer. Sender and receiver alternatingly access cell 0 and cell 1. Fig. 9b shows the clock signals produced by the sender and receiver oscillators. When stabilized, the sender is ahead by slightly more than a clock cycle. Both run on average with a frequency of roughly 2.28 GHz. Fig. 9c shows the mode signals of the sender and receiver which are computed by ClockedTh.

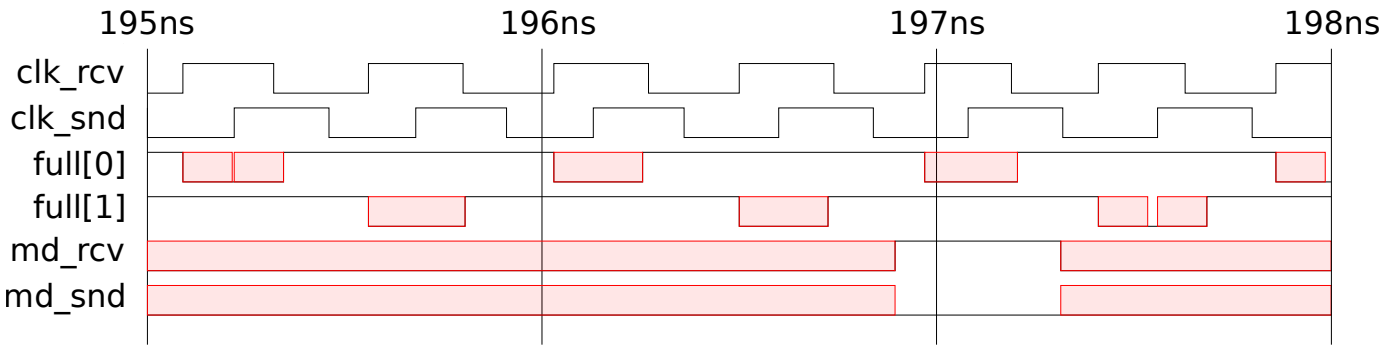
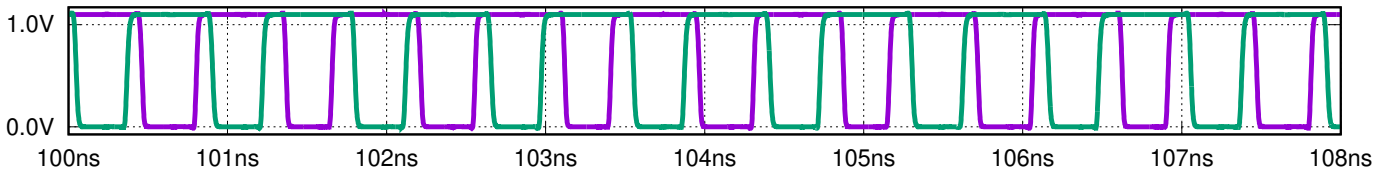
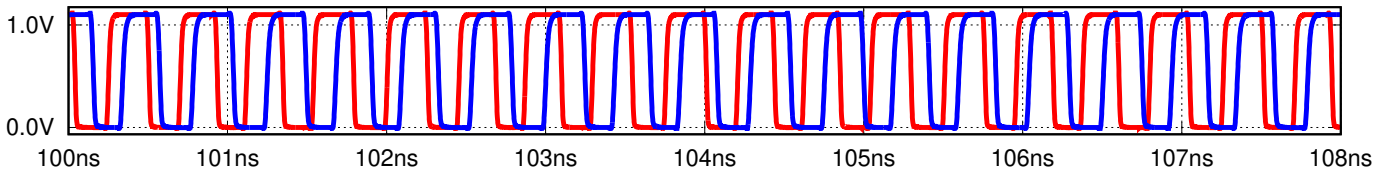


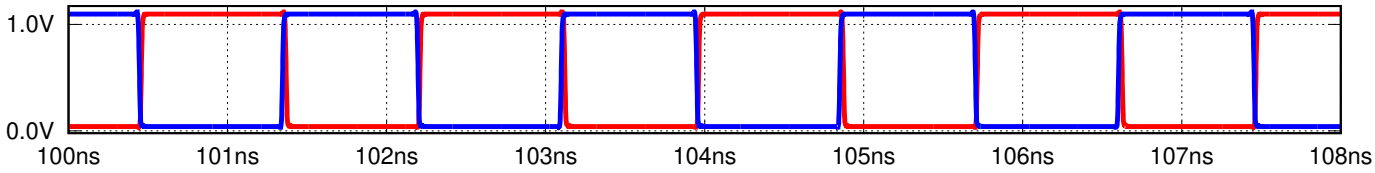
Fig. 8: Gate-level simulations for link with ClockedTh.



(a) full flags



(b) clock signals



(c) mode signals

Fig. 9: Ring buffer with two cells. (a) Rising and falling full flags of cell 0 (purple) or 1 (green) show write and read access to the respective cell. (b) Clock signals of the sender (red) and receiver (blue) oscillator. When stabilized both run at 2.28 GHz on average. (c) The mode signals for sender (red) and receiver (blue) side alternate between fast (2.42 GHz) and slow (2.09 GHz) mode.

D. Increasing Initialization Slack

If a sufficiently small δ (i.e., initial clock offset) cannot be guaranteed, the address pointers may "collide". However, if the pointers move apart sufficiently far, the link will resume to operate as intended. Note that the pointers colliding and moving at the same speed (i.e., the clocks running at the same speed) is an unstable equilibrium state, as the control logic aims at "pushing" them apart. Accordingly, this is a metastable state of the link, which can be expected to resolve fairly quickly.

We used a variation of the Spice simulation that allows us to initialize sender and receiver clocks to a specific offset (due to the the machinery simulation does not start exactly at 0 ns). Together with a suitable initialization of the full/empty flags, this simulates one of the clocks being started earlier.

We simulated the link with small initial offsets of the continuous pointers, i.e., $p_s(0) - p_r(0) = C_s(0) - C_r(0) + N/2 \approx 0$, with the goal of finding a good tradeoff between resolution time and precision of the initialization. Fig. 10 shows the pointer offset of the sender and the receiver clock $(p_s(t) - p_r(t) - N/2)$ over time t for different initializations. We see that simulations with an initial offset of 0 ps, 30 ps and 50 ps stay in the metastable state until eventually the sender advances by one clock cycle relatively to the receiver and the simulation stabilizes. Similarly, a simulation with an initial offset of -75 ps stays in the metastable state until the receiver advances by one clock cycle relative to the sender and the simulation reaches the corresponding stable state. Simulations with 30 ps resp. -75 ps offsets resolve after 11 ns resp. 10 ns. Hence, if the designer is willing to wait 11 ps after initialization, it is sufficient to guarantee avoiding this window of 105 ps during initialization. At the given clock speed, this corresponds to a much larger $\delta = 1/f^+ - 105/2$ ps, which in our setting is roughly 360 ps. In general, waiting for a couple of clock cycles after initialization increases the slack δ to being close to a full clock cycle.

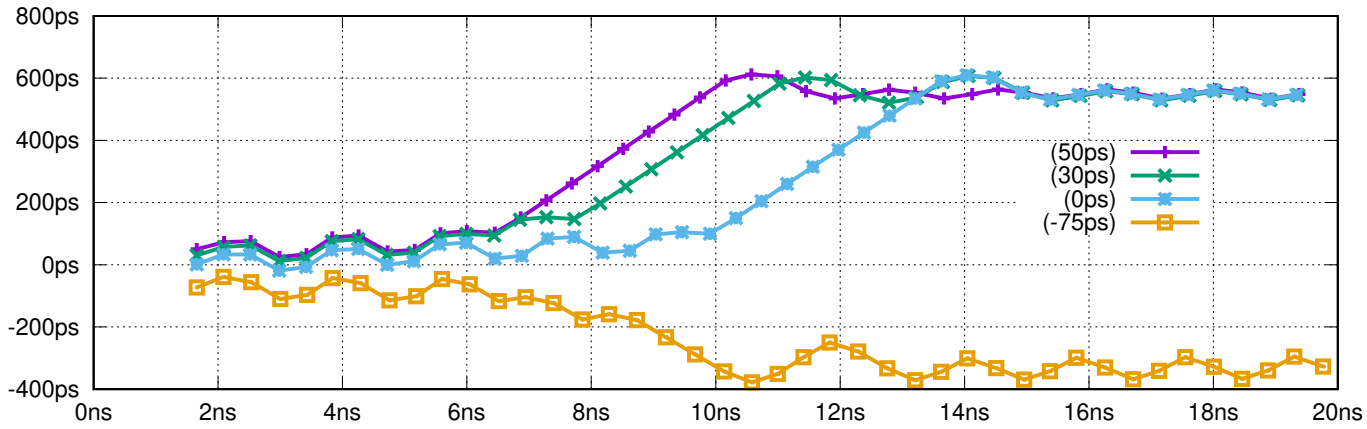


Fig. 10: Offset of the sender and receiver pointers over simulation time. When initializing the pointer offset to 50, 30, 0 and -75 ps, we observe different times to stabilization. According to the analysis, setup and hold times cannot be violated once link is stabilized.

V. Conclusion

We provided a digitally controlled implementation of a synchronously accessed buffer-based link, where both sender and receiver each have their own tunable clock. This can be seen as a distributed phase-locked loop, as we guarantee a fixed bound on the absolute time difference between the clocks, based on feedback derived from measuring the phase difference via keeping book of buffer accesses. Our design is novel in that we neither rely on analog or asynchronous design nor incur synchronizer delay, yet deterministically guarantee correct operation. By accepting an un- or metastable control signal for the oscillators when the buffer is roughly half full, we can completely dispense with synchronizers in the control loop. As this eliminates the associated delay from the control loop, it leads to relaxed timing constraints compared to synchronizer-based solutions. As a result, our link implementation can operate with a minimal buffer size of 2 under fairly weak requirements on the frequency stability of oscillators, yet guarantee correctness deterministically. We complemented our formal claims with VHDL and Spice simulations of UMC 65 nm ASIC implementations.

For the link to operate correctly upon initialization, it may be the case that δ, the initial clock offset between the sender and receiver clock, needs to be fairly small (cf. Corollary 8). If the resulting constraint is too tight, one can violate this constraint, possibly resulting in the two address pointers "meeting" each other. However, this is a metastable state of the control loop: If the two pointers move apart sufficiently far, operation will go back to the intended mode and push the pointers apart. Note that once the link has stabilized, the resulting total clock difference between the sender and receiver clock is unknown. One could now use the operational link to let the sender communicate its current clock value to the receiver (prefixing the encoding e.g. by a 1, while the buffer cells where initialized to 0).

However, the most practical compromise may be to avoid this complication and simply relax the initialization constraint without removing it entirely, as discussed in Section IV-D. Simulating the link with varying initial pointer offsets, we demonstrated a reasonable tradeoff between the time the link stays in an unstable state (max 11 ns) and the precision of the initialization (in two clock cycles avoid a window of 105 ps), cf. Figure 10.

One limitation of the proposed system is that it is restricted to a single link. In follow-up work [23], the ideas presented here are combined with a gradient clock synchronization algorithm [24], [25] that tightly bounds the phase offset between adjacent nodes. This retains the advantages of small buffers and latency while maintaining deterministic correctness. Future work needs to flesh this concept out into a fully-fledged design, which subsequently is to be tested in silicon. Here, suitable oscillators are more challenging to devise, because the scalability of the system is directly affected by the parameters of the oscillators. Ultimately, the result will be an alternative approach to clocking synchronous systems with far better scalability properties than classic designs, which derive time from a single reference.

Acknowledgment

We thank Attila Kinali and Prof. Ran Ginosar for pointing out that our link controller is in fact an "all-digital PLL."

References

[1] P. Teehan, M. Greenstreet, and G. Lemieux, "A Survey and Taxonomy of GALS Design Styles," *IEEE Design & Test of Computers*, vol. 24, no. 5, pp. 418–428, 2007.
[2] W. S. Coates and R. J. Drost, "Congestion and Starvation Detection in Ripple FIFOs," in *ASYNC*, 2003, pp. 36–45.
[3] R. Mullins and S. Moore, "Demystifying Data-Driven and Pausible Clocking Schemes," in *ASYNC*, 2007, pp. 175–185.
[4] R. Najvirt and A. Steininger, "How to Synchronize a Pausible Clock to a Reference," in *ASYNC*, 2015, pp. 9–16.
[5] T. Chelcea and S. M. Nowick, "Robust Interfaces for Mixed-Timing Systems," *IEEE Transactions on VLSI Systems*, vol. 12, no. 8, pp. 857–873, 2004.
[6] W. J. Dally and S. G. Tell, "The Even/Odd Synchronizer: A Fast, All-Digital, Periodic Synchronizer," in *ASYNC*, 2010, pp. 75–84.
[7] R. Dobkin, R. Ginosar, and C. P. Sotiriou, "High Rate Data Synchronization in GALS SoCs," *IEEE Transactions on VLSI Systems*, vol. 14, no. 10, pp. 1063–1074, 2006.
[8] S. Jackson and R. Manohar, "Gradual Synchronization," in *ASYNC*, 2016, pp. 29–36.

[9] L. Marino, "General Theory of Metastable Operation," *IEEE Transactions on Computers*, vol. C-30, no. 2, pp. 107–115, 1981.
[10] T. Polzer, T. Handl, and A. Steininger, "A Metastability-Free Multi-synchronous Communication Scheme for SoCs," in *SSS*, 2009, pp. 578–592.
[11] D. Sokolov, A. Mokhov, A. Yakovlev, and D. Lloyd, "Towards Asynchronous Power Management," in *FTFC*, 2014, pp. 1–4.
[12] S. Friedrichs, M. Függer, and C. Lenzen, "Metastability-Containing Circuits," *IEEE Transactions on Computers*, vol. 67, no. 8, pp. 1167–1183, 2018.
[13] D. Ghai, S. P. Mohanty, and E. Kougianos, "Design of Parasitic and Process-Variation Aware Nano-CMOS RF Circuits: A VCO Case Study," *IEEE Transactions on VLSI Systems*, vol. 17, no. 9, pp. 1339–1342, 2009.
[14] M. Függer and U. Schmid, "Reconciling Fault-tolerant Distributed Computing and Systems-on-Chip," *Distributed Computing*, vol. 24, no. 6, pp. 323–355, 2012.
[15] C.-C. Chung and C.-Y. Lee, "An All-Digital Phase-Locked Loop for High-Speed Clock Generation," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 2, pp. 347–351, 2003.
[16] A. Abadian, M. Lotfizad, N. E. Majd, M. B. G. Ghoushchi, and H. Mirzaie, "A New Low-Power and Low-Complexity All Digital PLL (ADPLL) in 180 nm and 32 nm," in *ICECS*, 2010, pp. 305–310.
[17] J. Pan and T. Yoshihara, "A Fast Lock Phase-Locked Loop Using a Continuous-Time Phase Frequency Detector," in *EDSSC*, 2007, pp. 393–396.
[18] A. Bashir, J. Li, K. Ivatury, N. Khan, N. Gala, N. Familia, and Z. Mohammed, "Fast Lock Scheme for Phase-Locked Loops," in *CICC*, 2009, pp. 319–322.
[19] J. Zhou, D. J. Kinniment, C. E. Dike, G. Russell, and A. V. Yakovlev, "On-chip measurement of deep metastability in synchronizers," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 550–557, 2008.
[20] T. Polzer and A. Steininger, "An approach for efficient metastability characterization of FPGAs through the designer," in *ASYNC*, 2013, pp. 174–182.
[21] A. Hajimiri, S. Limotyrakis, and T. H. Lee, "Jitter and Phase Noise in Ring Oscillators," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 6, pp. 790–804, 1999.
[22] S. Suman, K. Sharma, and P. Ghosh, "Analysis and design of current starved ring VCO," in *ICEEOT*, 2016, pp. 3222–3227.
[23] J. Bund, M. Függer, C. Lenzen, M. Medina, and W. Rosenbaum, "PALS: Plesiochronous and Locally Synchronous Systems," in *ASYNC*, 2020, to appear.
[24] F. Kuhn, C. Lenzen, T. Locher, and R. Oshman, "Optimal Gradient Clock Synchronization in Dynamic Networks," in *PODC*, 2010.
[25] C. Lenzen, T. Locher, and R. Wattenhofer, "Tight Bounds for Clock Synchronization," *Journal of the ACM*, vol. 57, no. 2, pp. 1–42, 2010.

Johannes Bund is a Ph. D. student at the Algorithms and Complexity department at MPI for Informatics. He graduated his B. Sc. and M. Sc. studies at the Saarland Informatics Campus. In 2018 he joined Christoph Lenzen's group at MPI for Informatics.

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

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

Moti Medina is a faculty member in the School of Electrical & Computer Engineering at the Ben-Gurion University of the Negev since 2017. Previously, he was a post-doc researcher in MPI for Informatics and in the Algorithms and Complexity group at LIAFA (Paris 7). He graduated his Ph. D., M. Sc., and B. Sc. studies at the School of Electrical Engineering at Tel-Aviv University, in 2014, 2009, and 2007 respectively. Moti is also a co-author of a text-book on logic design "Digital Logic Design: A Rigorous Approach", Cambridge Univ. Press, 2012.