

On Self-Timed Circuits in Real-Time Systems

Markus Furringer

Department of Computer Engineering
Embedded Computing Systems Group
Vienna University of Technology
1040 Vienna, Treitlstr. 3
Email: furringer@ecs.tuwien.ac.at

Abstract—While asynchronous logic has many potential advantages compared to traditional synchronous designs, one of the major drawbacks is its unpredictability with respect to temporal behavior. Having no high-precision oscillator, a self-timed circuit’s execution speed is heavily dependent on temperature and supply voltage. Small fluctuations of these parameters already result in noticeable changes of the design’s throughput and performance. Without further provisions this jitter makes the use of asynchronous logic hardly feasible for real-time applications.

We investigate the temporal characteristics of self-timed circuits regarding their usage in real-time systems, especially the Time-Triggered Protocol. We propose a simple timing model and elaborate a self-adapting circuit which shall derive a suitable notion of time for both bit transmission and protocol execution. We further introduce and analyze our jitter compensation concept, which is a three-fold mechanism to keep the asynchronous circuit’s notion of time tightly synchronized to the remaining communication participants. To demonstrate the robustness of our solution, we perform different tests, and investigate their impact on jitter and frequency stability.

I. INTRODUCTION

Asynchronous circuits elegantly overcome some of the limiting issues of their synchronous counterparts. The often-cited potential advantages of asynchronous designs are – among others – reduced power consumption and inherent robustness against changing operating conditions [1], [2]. Recent silicon technology additionally suffers from high parameter variations and high susceptibility to transient faults [3]. Asynchronous (delay-insensitive) design offers a solution due to its inherent robustness. A substantial part of this robustness originates in the ability to adapt the speed of operation to the actual propagation delays of the underlying hardware structures, due to the feedback formed by completion detection and handshaking. While asynchronous circuits’ adaptive speed is hence a desirable feature with respect to robustness, it becomes a problem in real-time applications that are based on a stable clock and a fixed (worst-case) execution time. Therefore, asynchronous logic is commonly considered inappropriate for such real-time applications, which excludes its use in an important share of fault-tolerant applications that would highly benefit from its robustness. Consequently, it is reasonable to take a closer look at the actual stability and predictability of asynchronous logic’s temporal behavior. After all, synchronous designs operate on the same technology, but hide their imperfections with respect to timing behind a strictly time driven control flow that is based on worst-case

timing analysis. This masking provides a convenient, stable abstraction for higher layers. In contrast, asynchronous designs simply allow the variations to happen and propagate them to higher layers. Therefore, the interesting questions are: Which character and magnitude do these temporal variations have? Can these variations be tolerated or compensated to allow the usage of self-timed circuits in real-time applications?

In our research project ARTS¹ (Asynchronous Logic in Real-Time Systems) we are aiming to find answers to these questions. The project goal is to design an asynchronous TTP (Time-Triggered Protocol) controller prototype which is able to reliably communicate with a set of synchronous equivalents even under changing operating conditions. TTP was chosen for this reference implementation because it can be considered as an outstanding example for hard real-time applications. In this article we present results based on our previous and current work. We will investigate the capabilities of self-timed designs to adapt themselves to changing operating conditions. With respect to our envisioned asynchronous TTP controller we will define a timing model, and also study the characteristics of jitter (and the associated frequency instabilities of the circuit’s execution speed) as well as the corresponding compensation mechanisms. We implement and investigate a fully functional transceiver unit, as required for the TTP controller, to demonstrate the capabilities of the proposed solution with respect to TTP’s stringent requirements.

The work is structured as follows: In Section II we give some important background information on TTP, the research project ARTS, and the used asynchronous design style. Section III discusses related work and the basic jitter terminology. Afterwards, Section IV discusses temporal characteristics of QDI circuits and presents some case studies. The main requirements, properties, and implementation details of the asynchronous time reference generation unit are presented in Section V, right before experimental results are shown in Section VI. The article concludes in Section VII with a short summary and an outlook to future work.

¹The ARTS project receives funding from the FIT-IT program of the Austrian Federal Ministry of Transport, Innovation and Technology (bm:vit, <http://www.bmvit.gv.at/>), project no. 813578.

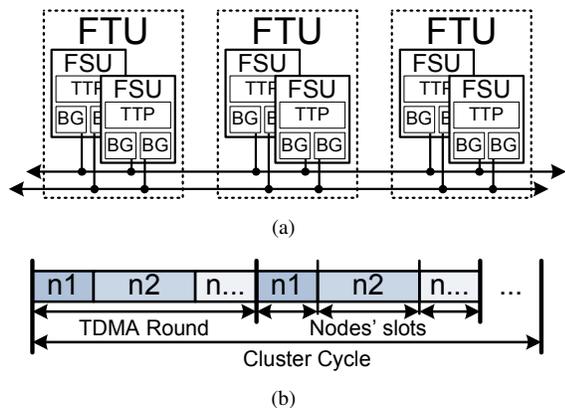


Figure 1. TTP system structure (a) and TDMA communication scheme (b).

II. BACKGROUND

A. Time-Triggered Protocol

The Time-Triggered Protocol (TTP) has been developed for the demanding requirements of distributed (hard) real-time systems. It provides several sophisticated means to incorporate fault-tolerance and at the same time keep the communication overhead low. TTP uses extensive knowledge of the distributed system to implement its services in a very efficient and flexible way. Real-time systems in general and TTP in particular are described in detail in [4], [5].

A TTP system generally consists of a set of Fail-Silent Units (FSUs), all of which have access to two replicated broadcast communication channels. Usually two FSUs are grouped together to form a Fault-Tolerant Unit (FTU), as illustrated in Figure 1(a). In order to access the communication channel, a TDMA (Time Division Multiple Access) scheme is implemented: As illustrated in Figure 1(b), communication is organized in periodic TDMA rounds, which are further subdivided into various sending slots. Each node has *statically* assigned sending slots, thus the entire schedule (called Message Descriptor List, MEDL) is known at design-time already. Since each node a priori knows when other nodes are expected to access the bus, message collision avoidance, membership service, clock synchronization, and fault detection can be handled without considerable communication overhead. Explicit Bus Guardian (BG) units are used to limit bus access to the node's respective time slots, thereby solving the babbling idiot problem. Global time is calculated by a fault-tolerant, distributed algorithm which analyzes the deviations of the expected and actual arrival times of messages and derives a correction term at each node.

The Time-Triggered Protocol provides very powerful means for developing demanding real-time applications. The highly deterministic and static nature makes it seemingly unsuited for an implementation based on asynchronous logic, as a precise notion of time (which is usually provided by a crystal oscillator in synchronous systems) is missing. Hence, these properties also make TTP an interesting and challenging topic for our exploration of predictability of self-timed logic.

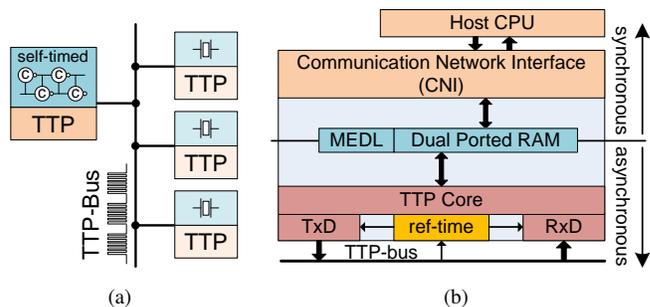


Figure 2. ARTS system setup (a) and TTP-node block diagram (b).

B. ARTS Project

The aim of the ARTS (Asynchronous Logic in Real-Time Systems) research project is to investigate the temporal predictability and stability of asynchronous (QDI, quasi delay-insensitive) hardware designs. More specifically, we want to compile models for the timing uncertainties of hardware execution times and extend these to make quantitative statements on the timing behavior of self-timed circuits. Our hope is that the theoretical and experimental analyses will also provide indications for improving the temporal stability of self-timed circuits.

The central concern for the project is the predictability of asynchronous logic with respect to its temporal properties. We therefore investigate jitter sources (e.g., data dependencies, voltage fluctuations, temperature drift) and classify their impact on the execution time. Using an adequate model allows us to identify critical parts in the circuit and implement measures for compensation. Another issue concerns timeliness itself, as without a reference clock we do not have an *absolute* notion of time. Instead, we will use the strict periodicity of TTP to continuously re-synchronize to the system and derive a time-base for message transfer.

The tangible project result shall be an asynchronous implementation of a TTP controller operating in an ensemble of conventional, synchronous controllers as illustrated in Figure 2(a). It is evident from the explanation in Section II-A that TTP's static bus access schedule as well as its clock synchronization and data transmission protocol rely on the stability of the constituent nodes' local clock sources. Therefore it seems quite daring to implement the controller logic in an asynchronous design style. However, moving such a deeply synchronous application to an asynchronous implementation is an interesting and informative challenge of its own, and—if we are successful—a very convincing case study for demonstrating the temporal predictability of asynchronous logic. In this context we need to solve two fundamental problems, whereby it is mandatory to derive quantitative boundaries for the attained properties in both cases:

- 1) We have to make our design operate stable enough to meet TTP's stringent requirements on execution times and jitter. Conceptually this issue has to do with the fact that control flow in self-timed circuits is flexible and not strictly time driven, as in the synchronous paradigm.

- 2) We have to provide a stable local time reference for bit timing and bus access. This issue originates from the fact that in synchronous systems the clock sources can also be used as time reference – which is missing in a self-timed approach.

Figure 2(b) shows the internal structure of the envisioned asynchronous TTP node. It is very similar to the existing TTP communication chip from TTTech Computertechnik AG [6], our project partner. The interface to the controlling host-CPU (CNI, Communication Network Interface) will remain synchronous, thus existing soft- and hardware-solutions for TTP can be used without modifications. However, the “central” parts (bus access, receiver-unit, transmitter-unit, macrotick generation, TTP-services, etc.) of the controller will be replaced by asynchronous implementations. Similar to the existing FPGA-based controller, a dual-ported RAM separates the CNI from the TTP-core, thereby forming not only a temporal firewall [5], but also splitting the synchronous from the asynchronous parts². It should be noted that the asynchronous controller implementation is intended solely as an academic case study and not as a prototype for an industrial design. However, a successful project outcome might considerably increase acceptance and even open new fields of applications for asynchronous designs.

The main challenge relies in the method of resynchronization, as the controller will use the data stream provided by the other communication participants to dynamically adapt its internal time reference. The chosen solution is to use a free-running, self-timed counter for measuring the duration of external events of known length (i.e., single bits in the communication stream). The so gained reference measurement can in turn be used to generate ticks with the period of the observed event. This local time base should enable the asynchronous node to derive a sufficiently accurate time reference for both low-level communication (bit-timing, data transfer) as well as high-level services (e.g. macrotick generation). The disturbing impact of environmental fluctuations is automatically compensated over time, because periodic resynchronization will lead to different reference measurements, depending on the current speed of the counter circuit.

C. Asynchronous Design Style

Our focus is on QDI (quasi delay insensitive) circuits, as they exhibit more pronounced “asynchronous” properties than bounded delay circuits. More specifically, we use the level-encoded dual-rail approach (LEDR [7], [8], used in Phased Logic [9], e.g.), which represents a logic signal on two physical wires. We prefer the more complex 2-phase implementation over the popular 4-phase protocol [2], [10], as it is more elegant and we have already gained some practical experience with it. LEDR periodically alternates between two disjoint code sets for representing logic “HI” and “LO” (two phases φ_0 and φ_1 , see Figure 3(a)), thus avoiding the need to insert NULL tokens as spacers between subsequent data items.

²Both host-CPU and TTP controller share a global time-base, thus concurrent access to the dual-ported RAM can be controlled to avoid collisions.

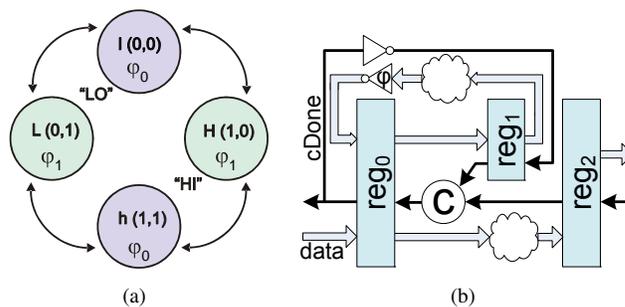


Figure 3. LEDR coding (a) and exemplary LEDR circuit structure (b).

In this approach, completion detection comes down to a check whether the phases of all associated signals have changed and match. As usual, handshaking between register/pipeline stages is performed by virtue of a *capture done* signal. Figure 3(b) shows an exemplary LEDR circuit with two sequential registers ($reg_0 \rightarrow_2$) and a feedback loop ($reg_0 \rightarrow_1 \rightarrow_0$). Direct feedback (i.e., without a shadow register like reg_1) is not possible, as race conditions and deadlocks may occur when a register issues its own acknowledges and requests. Also notice the phase inverter in the feedback path.

If external single-rail inputs are to be fed into LEDR circuits, special “interface gates” [9] must properly align (“synchronize”) these to the LEDR operation cycles. Concrete implementations and variations of such interface gates are described in [11].

The two-rail coding as well as the adaptive timing make LEDR circuits very robust against faults and changing operating conditions, unfortunately at the cost of increased area consumption and reduced performance. In principle, LEDR circuits belong to the class of QDI circuits, and typically the mandatory delay constraints are hidden inside the basic building blocks, while the interfaces between these modules are considered delay insensitive (i.e., unconstrained with respect to their timing delays).

D. Design Flow

For efficiently implementing complex asynchronous circuits it is of utmost importance that there exists a sophisticated software tool support. This is the main reason why the special area of automated synchronous-to-asynchronous converters gains more and more interest. This procedure, which is also called *desynchronization*, has the major advantage that designers need not care about the fallacies and pitfalls of asynchronous design directly. The systems can be described in a synchronous fashion, and an (optimally fully) automated tool chain converts it accordingly. A good overview to existing design flows using automated circuit conversion has recently been presented in [12]. The approach taken at our department [13] is also a form of desynchronization, and is explained in more detail in the next few paragraphs.

The general idea behind automated circuit conversion is to use a suitable circuit representation (one which designers are familiar with), and let a software tool convert the circuit

into an asynchronous representation. In the process, the tool needs to identify concurrency, as well as temporal³ and causal dependencies, and must of course guarantee functional and temporal equivalence between the input and output circuits. Using an arbitrary synthesis tool, the synchronous design is compiled into a gate-level netlist consisting only of D-flip-flops and the supported combinational gates. Based on this netlist our tool replaces the all sequential components with asynchronous registers, and all logic gates with their (dual-rail) asynchronous counterparts. After analyzing all dependencies, the tool automatically derives an initial token configuration, inserts shadow registers where necessary and finally produces the asynchronous netlist. This file can now be used for simulation or compilation (technology mapping, placement and routing, ...) with any suitable software. The presented approach has the advantage that it is almost totally automated. In addition, designers are not restricted to a special asynchronous design style. Our tool is currently capable of generating LEDR as well as NCL (Null Convention Logic) circuits.

III. STATE OF THE ART

A. Related Work

As already mentioned, the focus of this article will be on how to attain a stable time reference in the context of self-timed logic. From an abstract point of view this problem results from our attempt to insert a self-timed TTP node into an ensemble of otherwise fully synchronous nodes. For the purpose of our project, however, this is an important part of the deal, and situations like this may be encountered in practice as well. Let us first review some common options for building a time reference [14]:

- 1) *Crystal Oscillators*: This approach exploits mechanical vibrations paired with the piezo-electric effect, which attains highest precision at high frequencies. One of the severe drawbacks of crystal oscillators is their incompatibility with standard process technology. They need to be attached externally, which is area consuming, costly, and unreliable (as soldering contacts may break in harsh environments). Another drawback is the relatively long startup time of crystal oscillators (in the range of about ten milliseconds). Furthermore, susceptibility to mechanical vibrations, humidity and shock are considerably higher compared to alternative solutions.
- 2) *RC Oscillators [15]*: Here the time constant associated with charging a capacitance over a resistor is used as a time reference. While resistors and capacitors are very cheap components and can be integrated on silicon, they suffer from high fabrication variations as well as relatively large temperature and supply voltage dependencies. RC oscillators provide a good alternative to external crystal resonators, as long as high frequency and high precision are not major concerns.

³Temporal means the order of events, the exact timing is of course different because of the different design style.

- 3) *Integrated Silicon (Ring-)Oscillators*: In this approach the oscillations produced by a negative digital feedback loop, usually a ring spanning an odd number of inverters, are exploited [16]. The implementation is fully compatible with the CMOS fabrication process, but the produced frequency is determined by the delay path through the closed loop and hence heavily dependent on fabrication variations, supply voltage, and temperature. Different circuit structures are conceivable, from a simple chain of inverters to more complex solutions, as for example a free running self-timed circuit based on micropipelines [17].
- 4) *Distributed Clock Generation [18]*: For the use in embedded systems, distributed algorithms can be implemented to generate clock signals in a fault-tolerant distributed way. Each node can have its own clock source (i.e., an instance of the distributed algorithm) that remains in synchrony with the others within some known precision bounds. This approach can be viewed as a complex distributed silicon ring oscillator, inheriting the properties of the method above, but being more complex and robust due to the desired fault tolerance.

Obviously, the most straightforward solution in our setting is the use of the self-timed circuit's natural processing cycles as a time reference. Being able to exploit the oscillations of self-timed circuits as a replacement for the crystal oscillator directly meets our project goals. We do not consider solution (4) as it is overly complex for our application.

Another important alternative for generating precise time references are self-timed oscillator rings, which seem to be perfectly suited for the chosen asynchronous design methodology. In contrast to (3) they are based on Sutherland's micropipelines instead of a simple chain of cascaded inverters. A lot of research has been conducted on self-timed oscillator rings. For example, in [19] a methodology for using self-timed circuitry for global clocking has been proposed. The same authors also used basic asynchronous FIFO stages to generate multiple phase-shifted clock signals for high precision timing in [17]. Furthermore, it has been found that event spacing in self-timed oscillator rings can be controlled [16], [20]. The Charlie- and the drafting-effects have thereby been identified as major forces controlling event spacing in self-timed rings [17], [21].

B. Jitter

In synchronous systems we have the abstraction of an equally spaced time grid to which all transitions are aligned, and all deviations from this ideal behavior are commonly subsumed under the term jitter. Often jitter is associated with a synchronous clock source like a crystal oscillator, where it is obviously an undesired effect. Consequently, attempts have been made to identify the different sources and effects of jitter in order to mitigate the most relevant ones.

Literature generally distinguishes deterministic and random (indeterministic) jitter, as illustrated in Figure 4. The term

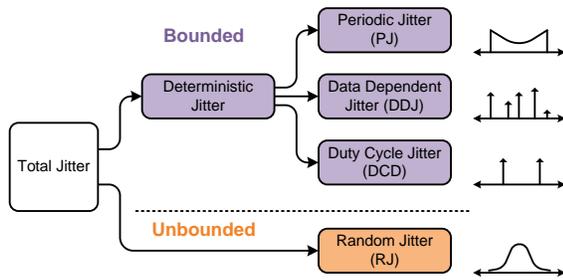


Figure 4. Jitter classification scheme [22].

random thereby refers to the statistical and random characteristics of jitter, and by that the magnitude is unbounded. In contrast, *deterministic* jitter sources have well-defined origins, are always bounded in magnitude, can be predicted, and are thus reproducible⁴. The following list shortly explains the most common sources of jitter [22]–[24]. Notice that they are *not* mutually exclusive – even worse, measurements mostly indicate a combination of several if not all of these types.

- *Data Dependent Jitter (DDJ)* is added to a signal according to the sequence of processed data values. Crosstalk, intersymbol interference, simultaneous switching noise, etc. are common sources of DDJ. In a jitter histogram, DDJ can often be identified as multiple separated peaks.
- *Bounded Uncorrelated Jitter (BUJ)* subsumes deterministic jitter sources that are not caused by data-dependencies. Fabrication and process variations are suitable examples.
- *Duty Cycle Dependent Jitter (DCD)* has its origin in differences in the slopes of rising and falling signal edges. High and low pulses of a periodic signal appear to have different lengths, which manifests as two distinct peaks in the jitter histogram. A similar effect can be observed (even in case of matching slopes) if the decision threshold for binary values is not at 50%.
- *Periodic Jitter (PJ)* is induced by periodic external events, such as switching power supply noise, and is per definition uncorrelated to any data-values. It results in pronounced peaks in FFT plots, for which reason it is also called sinusoidal jitter. In jitter histograms, the characteristic curve of PJ often looks like a bathtub.
- *Random Jitter (RJ)* can be seen as the (statistical) sum of multiple uncorrelated random effects (e.g., thermal or supply voltage noise), which is one of the main reasons for the Gaussian-like characteristics in jitter histograms.

With this abstract classification in mind, concrete manifestations of jitter can be defined [23], [24] for periodic signals.

- *Timing Jitter* is the deviation of a signal transition from its ideal position.
- *Period Jitter* is the deviation of a signal’s period from its nominal value.
- *Cycle-to-Cycle Period Jitter* is the variation in cycle-periods of adjacent cycles.

⁴Notice that random effects may also have well-defined origins and be reproducible, but this only accounts for their statistical parameters.

- *Long Term (Accumulated) Jitter* is defined as deviation of the measured *multi-cycle* time-interval from the nominal value. Especially random jitter accumulates over time, and thus its absolute value increases when observing long time intervals.

In a practical circuit we typically observe a superposition of the diverse types of jitter. It is therefore an intricate task to distinguish them in a measurement, even though powerful support by special jitter oscilloscopes is available.

IV. TEMPORAL CHARACTERISTICS

A. Stability Characteristics

Measuring jitter effects in asynchronous circuits differs from the synchronous case, because there are no specified reference values available for period or timing in general. After all, it is a desired property of asynchronous logic to adapt its speed of operation to the given conditions. As there is no dedicated clock in QDI circuits, we need to find another way to measure execution speed and the associated variations in the durations of single execution steps. To this end, the phase of any register is a suitable measure, as it changes exactly once per execution cycle. The inherent handshaking guarantees the *average* rate of phase changes for all coupled registers to be the same. However, due to the fact that LEDR circuits are “elastic”, there may be substantial differences in the execution speeds of adjacent pipeline stages for consecutive cycles. We define *execution period jitter*, or just execution jitter, to be the variation in the durations of phases of a *specific* LEDR-register. In order to classify the frequency stability of the single execution steps and the generated signals, we use Allan variance plots [25], [26], which provide the necessary means for a detailed analysis. Instead of a single number, Allan deviation is usually displayed as graph for gradually increasing durations τ of the averaging window. It therefore combines measures for both short (e.g., execution steps) and long (e.g., generated ticks for bit-timing) term stability in a single plot.

From an abstract point of view, we can categorize jitter into two major groups. On the one hand, *systematic jitter* (DDJ, global voltage and temperature change, e.g.) describes all effects that can be reproduced by our system setup. Consequently, for a given circuit, if we apply the same input transitions in the same state under the same operating conditions, we may expect the delay to be the same as well. If this is not the case, *random jitter* (local voltage and temperature fluctuations, noise, ageing, e.g.) has been experienced. Obviously, the latter cannot be controlled by the system setup. Recalling the different types of jitter from Section III-B, we consider DDJ and PJ to be systematic, while BUJ, DCD and RJ are considered random. Although BUJ and DCD seem to be systematic after all (they show very little dynamics, which are almost constant over a chip’s lifetime), it is hardly possible to influence them by means of system setup. The classification as “random” therefore seems adequate. It can also be expected that they are not a major source of frequency instabilities for QDI circuits. As we cannot

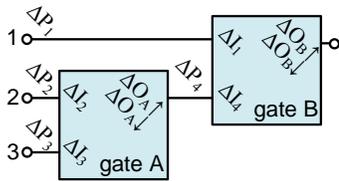


Figure 5. Timing model, example circuit.

control the random effects, we need to focus on systematic jitter when it comes to predictability and reproducibility for our envisioned TTP-controller:

- *Data-Dependent Execution Jitter (DDEJ)* deals with cases where the actual data values induce (systematic) jitter on a signal.
- Consequently, *Data-Independent Execution Jitter (DIEJ)* subsumes all non-data-dependent systematic jitter effects (global changes of temperature and voltage, e.g.). We also classify PJ to be in this group, as the corresponding sources are data-independent and, typically, deterministic.

Keeping the above classification of jitter sources in mind, we now examine sources of data-dependent and random jitter from a logic designer’s point of view. Figure 5 shows an example circuit with two gates *A*, *B*, four interconnect delays $\Delta P_{1,2,3,4}$, two input delays $\Delta I_{1,4}$, $\Delta I_{2,3}$ for each gate, and two output delays $\Delta O_{A,B}$. According to practical experience on FPGAs we also distinguish whether a rising or falling output transition occurs, indicated by the up/down arrows. Notice that even for symmetric functions like AND the input delays ΔI are not necessarily the same for all input terminals. Consequently, a transition on input 1 or 2 has a delay $\Delta 1 = \Delta P_1 + \Delta I_1 + \Delta O_B(\uparrow \text{ or } \downarrow)$, and $\Delta 2 = \Delta P_2 + \Delta I_2 + \Delta O_A(\uparrow \text{ or } \downarrow) + \Delta P_4 + \Delta I_4 + \Delta O_B(\uparrow \text{ or } \downarrow)$, respectively.

A more interesting case occurs when inputs 1 and 2 are connected. A transition on this combined input may need $\Delta 1$ one time and $\Delta 2$ another time, depending on which path is enabled by the involved gates (e.g., if the change on input 1 is sufficient for gate *B* to toggle, independently from *A*’s output). This behavior, in combination with possibly different output delays $\Delta O_{\uparrow\downarrow}$ for opposite transitions, is the main reason for data dependent jitter. Since LEDR designs are *strongly indicating*, which means that the realized logic functions always exhibit worst case performance, the observed data dependencies have their origin mainly in the internal structure of the single LEDR gates, and not in the Boolean expressions that are implemented by them. Out of this analysis it is easy to see that DDJ is systematic, although difficult to predict for complex paths. Obviously, the introduced delays are not constant, but are subject to so-called “PVT-variations”. Fabrication tolerances, local/global supply voltage and temperature variations considerably change the delays. Thereby, deliberate changes in voltage or temperature will cause systematic jitter, while local fluctuations (noise, e.g.) typically result in random jitter. Given that random effects are statistically independent and follow a normal distribution, overall random jitter can be

modeled by (statistically) integrating over all delays along a certain path. In order to derive a quantitative measure for data-dependent jitter, the longest and shortest paths through a given circuit must be examined.

As target platform for our prototype implementation we want to use FPGAs, which are not in any way optimized for asynchronous hardware. Which effects must be taken into account when elaborating the delay of LEDR-gates (also refer to Figure 5)? Notice that the above model can be applied hierarchically, thus it can be used for complex elements (composed of basic gates, e.g.) as well.

- 1) Gate delays: We have already seen the delays associated with basic gates in Figure 5. These delays can also be applied to FPGAs, where look-up tables (LUTs) are used as basic gates to implement Boolean expressions.
- 2) Internal interconnects are necessary if a complex gate cannot be realized out of a single LUT. However, depending on the exact placement and routing, these delays are not the same for all “internal” interconnects. By using pre-compiled components (hard macros) it can be assumed that all LEDR-gates of the same type also have approximately the same internal interconnect delays.
- 3) External interconnects (connections between different LEDR gates) are certainly a central source of delays. Not only do they have considerably different values for different (input-)signals, they can also be expected to be distinctively different for all LEDR-gates, even those of the same type.
- 4) Environmental conditions such as temperature drift and voltage fluctuations also influence the circuit’s timing. For this model we assume these properties to affect the entire chip homogeneously.

Since we are operating at gate-level (with relatively large delays), transistor-level effects such as the Charlie- or Drafting-effects [20], as well as SSN (Simultaneous Switching Noise [27]) can be neglected. The latter is covered by DDJ anyway. However, there is a comparable behavior for the Charlie-effect at gate-level as well (which directly follows from the delay assumptions we made): We need to look at the relative arrival times of input-transitions at gates. Assume for Figure 5 that $\Delta 1 = 5ns$ and $\Delta 2 = 9ns$. If both inputs arrive simultaneously, the output is stable after $9ns$. If, however, input 2 arrives $4ns$ before input 1, the additional delay *after 1’s arrival* is only $5ns$.

B. Case Studies

We now present three simple circuits and study their characteristics according to the properties elaborated in the previous section. All three designs consist of a free-running, closed-loop LEDR circuit with two registers (one of which being a shadow register with a phase inverter, as direct feedback is not possible, recall Figure 3(b) from Section II-C). The first two examples are a 4-bit and a 16-bit counter, respectively. Both counters are realized as ripple-carry adders. The third design is a free-running 16-bit LFSR. The characteristic figures of each design are summarized in Table I. Row “Logic Elements”

	4-bit Counter	16-bit Counter	16-bit LFSR
Logic Elements	100	452	276
LEDR-Gates	5	41	3
LEDR-Registers	2*4	2*16	2*16
Logic Depth	3	15	1
Performance	125MHz (8ns)	42MHz (24ns)	100MHz (10ns)
Phase C2C Jitter	3.0ns (403ps)	3.5ns (710ps)	0ns (322ps)
Count Period	127ns	1.6ms	683 μ s
Count C2C Jitter	$\sigma = 113$ ps	$\sigma = 57$ ns	$\sigma = 43$ ns

Table I
EXAMPLE CIRCUITS SUMMARY.

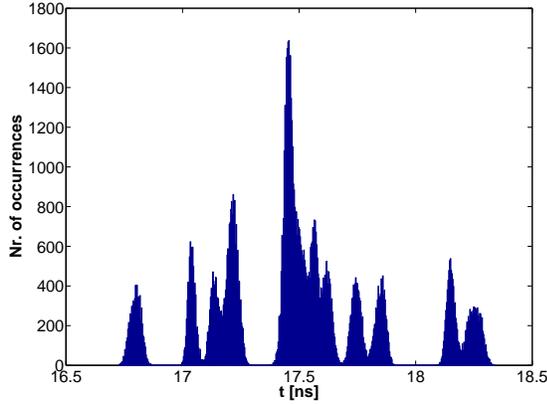
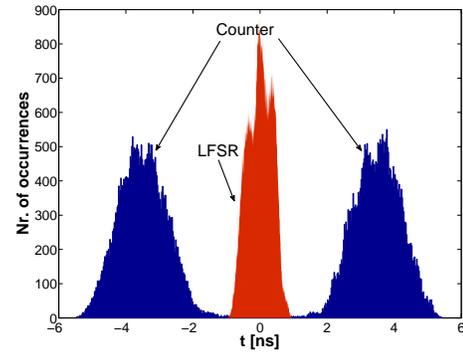


Figure 6. Jitter histogram, 4-bit counter, DDJ.

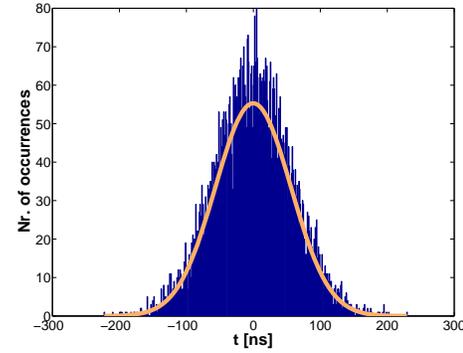
lists the number of logic cells needed on the FPGA. “LEDR-Gates” and “LEDR-Registers” specify how many dedicated combinational LEDR-gates and registers are used, respectively. “Logic Depth” defines the maximum number of LEDR-gates connected in series, and “Performance” gives the average operating speed of the circuit. In row “Phase C2C Jitter” the mean observed cycle-to-cycle period jitter of single execution steps (phases) is listed, followed by the respective standard deviation in parentheses. “Count Period” shows the duration until counter overflow, and “Count C2C Jitter” finally specifies the standard deviation of cycle-to-cycle period jitter of entire counting-periods (i.e., 2^4 and 2^{16} steps, respectively).

4-Bit Counter. To better illustrate DDEJ, we manually inserted some wire delays in order to obtain the more pronounced jitter histogram of Figure 6 (the values of Table I are based on the original circuit). One can see that there are 11 peaks, which are a superpositions of 16 separate peaks, each of which corresponding to one of the 16 possible counter states. It can further be seen that the single humps have (approximately) Gaussian characteristics, but in contrast to real normal distribution they are bounded quite sharply. The distinct peaks are direct consequences of data-dependencies induced by the current state (i.e., the actual counter value) of the circuit.

16-Bit Counter. Figure 7(a) shows the jitter histogram of the counter’s cycle-to-cycle execution period jitter. A 16-bit counter has 2^{16} slightly different execution times, and their superpositions result in one big Gaussian-like distribution *for each phase* (therefore, “Phase C2C Jitter” has a mean value



(a)



(b)

Figure 7. Execution period C2C jitter of 16bit counter and LFSR (a), C2C jitter histogram of counting periods of 16bit counter (b).

other than zero). Again, data-dependencies are responsible for the significant differences in the durations of phases φ_0 and φ_1 : There are minor variations in the propagation delays of gates due to the actual phase-values. These accumulate while passing through the logic stages and manifest as two separate peaks in the jitter histogram. For a counter there is also another, systematic effect which further amplifies these differences: The odd phases φ_1 are strictly coupled with odd counter values (as long as there is an even number of counting steps). Therefore, odd phases have considerably more logic ones in their input values than even phases, hence further aggravating the data-dependencies. In addition, Figure 7(b) depicts the cycle-to-cycle jitter of entire counting periods. No data-dependencies (and thus no phase-dependencies) can be observed any more; The graph shown is the accumulated random jitter over 2^{16} execution steps. One could expect the standard deviation of an entire counter period to be (cf. Table I) $\sigma_{count} = \sigma_{step} \sqrt{2^{16}} \approx 182$ ns, but as all data dependent effects are exactly the same for each complete period, only random jitter remains (which is considerably less, $\sigma_{count} = 57$ ns).

Linear Feedback Shift Register (LFSR). The internal logic just consists of three XNOR-gates, thus being extremely efficient in terms of performance and area consumption. For reasons of performance, our design uses Galois LFSRs, thus having a maximum logic depth of only one gate equivalent. Consequently, the data-dependent effects are considerably less severe because of the reduced logic depth (no accumulation

through logic stages possible). This is also evident in Figure 7(a) and Table I: The LFSR does not show significant differences for phases $\varphi_{0,1}$ (resulting in a mean value of zero for “Phase C2C Jitter”), and the overall width of the histogram is substantially less compared to the counter.

V. ASYNCHRONOUS REFERENCE TIME

In order to allow for reliable TTP communication, the resulting asynchronous controller must have a precise notion of time. As there is no reliable reference time available in the asynchronous case, we design a circuit that uses the TTP communication stream to derive a suitable, stable time-base. We construct an adjustable tick-generator and periodically synchronize it to *incoming* message-bits. In our configuration, the bit-stream of TTP uses Manchester coding, thus there is at least one signal transition for each bit which we can potentially use for recalibration. The Manchester encoding is a line code which represents the logical values 0 and 1 as falling and rising transitions, respectively. Consequently, each bit is transmitted in two successive symbols, thus the needed communication bandwidth is double the data rate. The top part of Figure 8(a) shows three bits of an exemplary Manchester coded signal, whereby the transitions at 50% of the bittime define the respective logical values. This encoding scheme has the advantage of being self-clocking, which means that the clock signal can be recovered from the bit stream. From an electrical point of view, Manchester encoding allows for DC-free physical interfaces.

Figure 8(a) further illustrates the properties that our design needs to fulfill. As already mentioned, Manchester coding uses two symbols to transmit a single bit, thus the “feature-size” τ_{ref} of the communication stream is half the actual bit-time τ_{bit} . It can also be seen that the sampling points need to be located at 25% and 75% of τ_{bit} , respectively. We intend to achieve this quarter-bit-alignment by doubling the generated tick-frequency ($\tau_{gen} = \frac{\tau_{ref}}{2}$). Consequently, each rising edge of signal *ref-time* defines an optimal sampling point. As our circuit is implemented asynchronously, the generated reference signal will be subject to jitter. Furthermore, temperature and voltage fluctuations will also change the reference’s signal period. It is therefore necessary to make the circuit self-adaptive to changing operating conditions.

A. Concept and Requirements:

A key problem in our project is how to provide a reference time for the transmission and reception of a digital Manchester-coded bit stream on the communication channel. For already explained reasons we want to use some type of self-clocked loop for this purpose. In order to derive a suitable concept for a solution, let us first compile the requirements:

- 1) With a typical 1MBit/s transmission rate the bit length τ_{bit} is $1\mu s$, but the Manchester coding exhibits a “feature size” of $0.5\tau_{bit}$, i.e., 500ns (see Figure 8(a)). As a consequence we have to sample (at least) twice per τ_{bit} , ideally at 25% and 75% of the bit time.

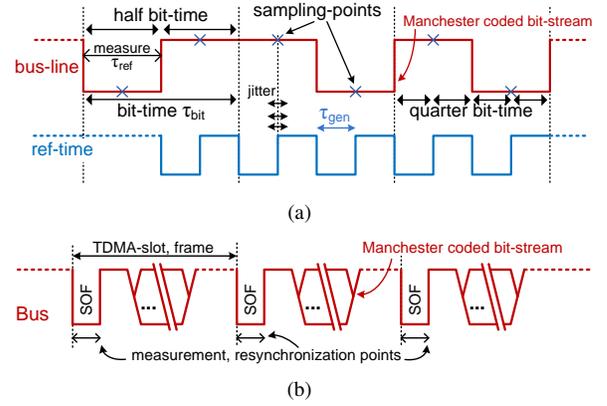


Figure 8. Manchester code with sampling points (a), TTP-slots, resynchronization (b).

- 2) The time reference needs to remain synchronized with the other local references in the system. This requires periodic re-synchronization even in the synchronous case. It is therefore mandatory to have a reference whose timing can be precisely adjusted.
- 3) In the synchronous case the resolution of the adjustment is determined by the local clock generator. With a typical clock frequency of 40MHz we have a resolution R of some 25ns. It seems reasonable to strive for a similar resolution in our case.
- 4) Re-synchronization is performed every TDMA round in the synchronous case (state correction). As we expect the asynchronous reference to be considerably less stable than a crystal clock, we have to perform re-synchronization more often and provide a rate correction as well.
- 5) For the purpose of our study we want to consider all provisions to compensate for the non-ideal behavior of our reference. Among these are the elimination of long-term effects by virtue of periodic re-calibration, masking of random effects by means of averaging, and avoidance of systematic effects by means of design measures. These points are discussed in more detail in Section V-B.

Requirements (2) and (3) spoil our hope to use the operation cycles of the complete asynchronous controller as a time reference – these are neither fast enough nor adjustable. Therefore we decided to use a separate, small circuit as a time reference that is not dependent on the controller’s control flow. A counter suggests itself here to count up to a threshold, whose adjustment already implements the rate correction desired in (4). The price for this decoupling is the need for an explicit synchronization of the operation cycles of the remaining controller logic to this reference.

We will exploit the deterministic nature of TTP and use features of known length in the periodic data stream provided by the other communication participants as a reference to

periodically adjust our local timing⁵. According to (4) we have to adjust the reference as often as possible. We can take advantage of the start of frame (SOF) sequence (HI followed by LO with a length of $0.5\tau_{bit}$ each) for our measurement. More specifically we use the first HI as our “reference half-bit” (see Figure 8(b)). Measuring just a half-bit cell instead of a much longer interval clearly increases the quantization error. However, longer intervals tend to become dependent on the system configuration (number of involved nodes, configured message length, etc.), thereby considerably complicating the measurement circuitry because more control logic is necessary. This increased complexity not only downgrades performance (which in turn increases the quantization error), but also introduces more jitter and makes timing analysis/predictions substantially harder.

The proposed procedure comprises two phases: (i) A *measurement phase* m during which the reference counter’s threshold is determined by starting at 0 at the beginning of the reference half-bit and simply stopping the counter at the observed half-bit’s end. (ii) A *reproduction phase* r during which the observed half-bit length is periodically reproduced by having the counter wrap around to 0 as soon as it reaches the threshold determined above (with a proper initial alignment of $0.25\tau_{bit}$ according to (1)).

B. Properties:

The above procedure implies a *state correction* of the local time, as the internal time is corrected upon detection of the start of frame. In addition a *rate correction* is achieved by adjusting the threshold for every bit. The latter allows for a very tight matching between the current sender’s actual bit length and the period of our reference counter (that is subject to variations caused by changing operating conditions, e.g.). *Random* effects are automatically averaged by periodically counting to the measured threshold value. The temporal proximity of measurement and associated reproduction phases is beneficial, as it facilitates an effective compensation of long term variations (long with respect to the frame length). In other words, the disturbing impact of environmental fluctuations is automatically compensated over time, because the periodic re-synchronization events will lead to different reference values depending on the current speed of the counter circuit.

The obvious questions that arise are, which properties does our solution have with respect to frequency stability, and how can changing environmental conditions be dealt with. The following list summarizes all effects that must be taken into account and discusses their impact on our design. To this end, we need to define some parameters for a simple quantification. τ_{bit} has already been introduced as the duration of one Manchester coded bit on the bus. We further define $\tau_{step,m}$ and $\tau_{step,r}$ to be the average durations of execution cycles in *measurement phase* and *reproduction phase*, respectively. τ_{step} is used if the minor difference between these two phases is not

⁵We are well aware that this may become a circular argument in case of all nodes in the system being implemented asynchronously. This is, however, not our intention in the project.

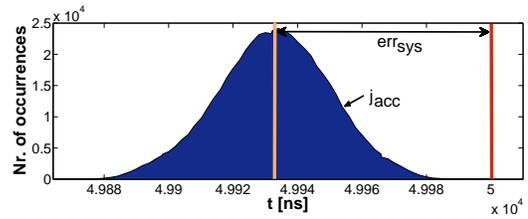


Figure 9. Systematic and random error (jitter).

important. Finally, τ_{ref} denotes the duration of the reference signal to measure. Consequently, $cnt_{ref} = \lfloor \frac{\tau_{ref}}{\tau_{step,m}} \rfloor$ is the average number of execution steps (i.e., the counter threshold) for the measured pulse of length τ_{ref} .

- The *quantization error* can be expressed as $|err_{quant}| \leq 2\tau_{step}$. Since the synchronization circuits presented in [11] provide new data in phase φ_1 only (i.e., only every second phase), the starting and ending transitions introduce a quantization error of up to $\mp 2\tau_{step}$, respectively. By keeping τ_{step} as low as possible, err_{quant} can be improved accordingly. (Using another synchronization circuit may not be possible for all target technologies, but can potentially decrease $|err_{quant}|$ to an upper bound of τ_{step}).
- *Systematic errors* are introduced by data dependent jitter, as mentioned in the previous sections. Two major cases need to be distinguished for our design: (i) The single execution steps while counting up to the measured threshold value show considerable DDJ with respect to each other. (ii) As *measurement* and *reproduction phase* are different states with slightly different register/input values, their average execution speed typically does not match exactly, i.e., $\tau_{step,m} \neq \tau_{step,r}$. While (i) does not affect the overall counting period (because the intermediate DDJ is always the same for each counting-cycle), (ii) introduces errors that can only be compensated by clever circuit design or complex correction measures at logic level. The relative deviation of the generated time reference from its measured value can be expressed as factor $f_{dev} = \frac{\tau_{step,r}}{\tau_{step,m}}$ and should optimally be $f_{dev} = 1$.
- *Systematic, long term effects* are mainly caused by slow changes in temperature or supply voltage. Given these fluctuations are slow enough (compared to one TDMA slot), they are compensated automatically at each resynchronization point (cf. Figure 8(b), because depending on the current speed of the counter circuit there will be a different reference/threshold value).
- *Short term effects* (either random or systematic) that occur faster than a TDMA slot cannot be compensated by our design. In such a case the self-timed circuit accumulates timing errors due to the changed operating speed and will eventually lose synchrony to the remaining cluster.
- *Random effects* cannot be compensated easily. However, when averaging over long periods, statistical outliers become less important and frequency stability improves. For

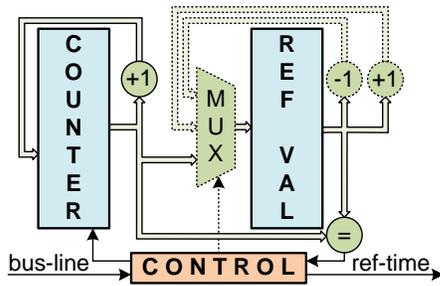


Figure 10. Basic structure of the timer-reference generation circuit.

our design, averaging occurs automatically over the periodic counting-cycles. Consequently, as for quantization errors, large values for cnt_{ref} are desirable (in contrast to systematic errors, where lower threshold values are preferable in case $f_{dev} \neq 1$).

Out of these properties, the following essential consequences are derived. The overall systematic error can be written as $err_{sys} = err_{quant} + (1 - f_{dev})cnt_{ref}\tau_{step,m} = err_{quant} + cnt_{ref}(\tau_{step,m} - \tau_{step,r})$. While f_{dev} is almost constant for a given circuit, err_{quant} can be different for each measurement (thus err_{sys} is variable as well). Furthermore, long term accumulated jitter (by definition indeterministic and unbounded) causes additional frequency-inaccuracies. Assuming a normal distribution for jitter induced in every execution step, the accumulated jitter can be approximated as $j_{acc} = N(0, \sigma_{acc}^2)$ with $\sigma_{acc}^2 = cnt_{ref}\sigma_{step}^2$ (where σ_{step}^2 is the variance of a single execution step).

Figure 9 illustrates the above types of error. A signal with a high-period of $100\mu s$ serves as reference for the generation of a $50\mu s$ time-base (solid line on the very right). The reproduced signal is measured and plotted as jitter histogram (Gaussian-like area). err_{sys} is the deviation of the average signal-period from its nominal value at $50\mu s$. On the other hand, j_{acc} manifests as Gaussian jitter with variance σ_{acc}^2 around the mean duration of approximately $49.93\mu s$.

C. Implementation

The basic structure of the circuit is shown in Figure 10. As one can see, the interface of our design is quite simple. There is only one input (*bus-line*, the receive-line of the TTP bus), as well as one output (*ref-time*, an asynchronously generated signal with known period). The dashed components and the MUX have been added to the circuit to allow rate correction on a per-bit basis in combination to the absolute measurement of τ_{ref} during the first bit of a message (SOF, Start-of-Frame). If the control block detects the SOF signature, it resets the free-running counter unit. The asynchronous counter periodically increments its own value at a certain (variable) rate, which mainly depends on the circuit structure, placement and routing, and environmental conditions. After time τ_{ref} , the corresponding end of SOF will eventually be detected by the control-block. As a consequence, the current counter value is preserved in register *ref-val* (reference value) and the counter is restarted.

The controller is now able to reproduce the measured low-period τ_{ref} by periodically counting from zero to *ref-val*, and generating a transition on *ref-time* for each compare-match. In order to achieve the 25%/75%-alignment, we double the output frequency by simply halving *ref-val*. The Manchester coded bitstream further allows to use each bit as additional resynchronization point. We therefore exploit the fact that there is at least one transition per bit in the bitstream. We now can compare the expected and actual points in time when transitions on the bus occur, and slightly adjust *ref-val* by just in-/decrementing it by one, depending on whether the observed transitions were early or late (with respect to our internal expectations). This rate correction significantly increases the reachable accuracy of the time reference generation unit, as the quantization errors are compensated by averaging over time. Furthermore, speed differences which have their origin in the non-matching speeds of measurement and reproduction phase $\tau_{step,m}$ and $\tau_{step,r}$, can also be compensated with this strategy.

In order to increase the achievable precision of our system, it is important to optimize the circuit of Figure 10 accordingly. The following two changes can be applied to significantly increase performance and reduce the gate count. (With both optimizations applied we are able to reduce the duration of a single execution step from approximately $30ns$ down to $14ns$.)

- The counter modules (i.e., the incrementers and decrementers) can be replaced by a simple LFSR. This not only reduces the overall gate count, but also significantly increases the maximum speed. LFSRs can be built using at most three XNOR gates for almost all register widths. While ordinary counters have the advantage of producing strictly monotonic outputs, LFSR generate pseudo random numbers. For our application a deterministic order of states is sufficient, monotonicity of counting states is not a requirement. LFSRs are thus a suitable optimization.
- The absolute measurement of the SOF can be removed. For this to work, however, the *ref-val* needs to be initialized with a value approximately matching the expected operating frequency. If this value is too far off, the module will not be able to synchronize itself to the bit stream correctly. While this optimization also decreases area consumption and at the same time increases the maximum performance (control logic is simpler and 3-way MUX can be replaced by a less complex 2-way MUX), adaption to the correct frequency might take relatively long in case *ref-val* is far off its optimum value. In addition, finding a feasible initialization value requires thorough timing analysis or measurements on the target platform.

VI. EXPERIMENTAL RESULTS

In this section we will present a detailed analysis of the experiments we performed with the proposed circuit from Figure 10. We will vary temperature and operating voltage and monitor the generated time reference under these changing conditions. Simultaneously, we will also evaluate the robustness and effectiveness of the three compensation mechanism implemented in the final design:

- 1) *Low-Level State Correction*: Measuring period τ_{ref} of the SOF sequence retrieves an absolute measure of the reference time. However, as only one measurement is performed per message, quantization errors and other systematic, data-dependent delay variations significantly restrict the achievable precision. The possible resolution depends on the speed of the free-running counter, and is at about 25ns for our current implementation⁶.
- 2) *Low-Level Rate Correction*: As the Manchester code always provides a signal transition at 50% of τ_{bit} , we can continuously adapt the measured reference value *ref-val*. We only allow small changes to *ref-val*: It is either incremented or decremented by one, depending on whether the expected signal transitions are late or early, respectively. The advantage of this additional correction mechanism is that quantization errors and data-dependent effects are averaged over time, thus increasing precision.
- 3) *High-Level Rate Correction*: The software-stack controlling the message transmission unit can add another level of rate correction. As it knows the expected (from the MEDL) and actual (from the transceiver unit) arrival times of messages, the difference of both can be used to calculate an error-term. High-level services and message transmission can in turn be corrected by this term to achieve even better precision. The maximum resolution which can be achieved by this technique depends on the baud-rate, and is half a bit-time.

Remark: We are well aware that the presented results can only be seen as snapshot for our specific setup and technology. Changing the execution platform will certainly change the specific outcomes of the measurements, as jitter and the corresponding frequency instabilities mainly depend on the circuit structure and the used technology. However, from a qualitative point of view, our results are valid for other platforms and technologies as well. The presented model and the proposed circuits are flexible enough to be applied to different technologies. However, the main problem is the necessity to perform concrete measurements for each target technology in order to obtain meaningful quantitative evaluations. Temporal behavior and specific jitter characteristics are always dependent on fabrication variations and the operating environment. While the theoretical model needs to be complemented by thorough measurements, the proposed circuits are capable of tolerating these (statistical) variations because of the continuous calibration of internal timing.

A. Time Reference Generator

Before we start with the message transmission unit, which implements all of the above compensation mechanisms, we want to take a closer look at the basic building block (cf. Figure 10). Clearly, compensation method (3) is not present, as we just investigate the time reference generation unit. This

⁶Notice that FPGAs are not in any way optimized for LEDR circuits. Dual-rail encoding introduces not only considerable interconnect delays, but also significant area overhead compared to ordinary synchronous logic.

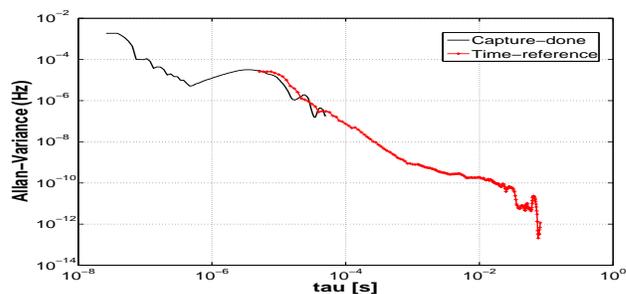


Figure 11. Allan-Variance.

unit does not actually receive or transmit messages, it just generates signal *ref-time* out of the incoming signal transitions on the TTP bus. The measurement setup is fairly simple: There is a (synchronous) sender, which periodically sends Manchester coded messages. The asynchronous design uses these messages to generate its internal time-reference. All measurements have been taken while the bus was idle. This way, we can observe the circuit's capability of reproducing the measured duration without any disturbing state- or rate correction effects. If not stated otherwise, the measurements are taken at ambient temperature and nominal supply voltage.

First we take a look at the frequency stability of *ref-time* and *cDone*. The first part of the Allan-plot in Figure 11, ranging from approximately $2 \cdot 10^{-8}s$ to $10^{-4}s$ on the x-axis, is obtained by monitoring the handshaking signal *capture-done* from a register cell. The second part, which starts at $3 \cdot 10^{-6}s$ and thus slightly overlaps with *capture-done*, has been obtained by measuring *ref-time*. Notice that it is no coincidence that both parts in the figure almost match in the overlapping section: Signal *ref-time* is based upon the execution of the low-level hardware and is therefore directly coupled to the respective jitter and stability characteristics. It is obvious from the graph that the stability increases to about $10^{-10}Hz$ for $\tau \approx 10^{-2}s$. Furthermore, the reference signal is far more stable than the underlying generation logic (*cDone*), as periodically executing the same operations compensates data-dependent jitter and averages random jitter. Although the underlying low-level signals jitter considerably due to data-dependent jitter, the circuit's output is orders of magnitudes more stable, as these variations are canceled out during the periodic executions.

One of the major benefits of the proposed solution is its robustness to changing operating conditions, thus we additionally vary the environment temperature and observe the changes in the period of *ref-time*. We heat the system from room temperature to about $83^\circ C$, and let it cool down again. Figure 12 compares *ref-val* to the signal period of *ref-time*. While the ambient temperature increases, *ref-val* steadily decreases from 265 down to 256. The period of *ref-time* makes an approximately 19ns-step (the duration of a single execution step) each time *ref-val* changes. During the periods where the changes in execution speed cannot be compensated (because they are too small), *ref-time* slowly drifts away from the

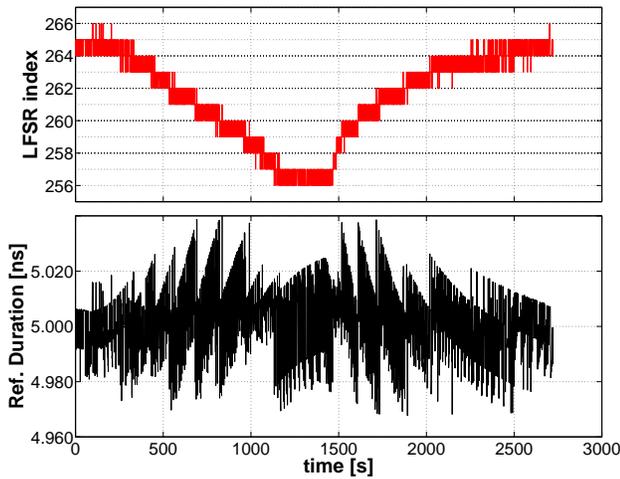


Figure 12. *ref-val* vs. timer reference period for temperature-tests.

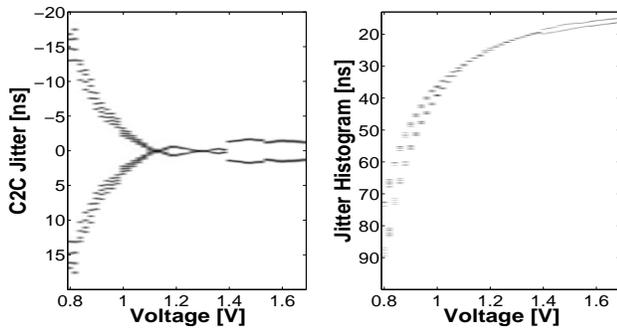


Figure 13. Cycle-to-cycle jitter (left), Jitter histogram (right).

optimum at $5\mu s$. Without any compensation measures the duration of *ref-time* would be about $5180ns$ at the maximum temperature, instead of being in the range of approximately $5\mu s \pm 38ns$ (i.e. the duration of \pm two execution steps), no matter what temperature. Notice that the performance of the self-timed circuit decreases by 3.5% at the maximum temperature, which seems to be relatively low, but it certainly is a showstopper for reliable TTP communication.

Far more pronounced delay variations can be obtained by changing the core supply voltage. We applied 0.8V to 1.68V in steps of 20mV core voltage to our FPGA-board. This time the execution speed of our self-timed circuit increased from about 80ns per step to approximately 15ns per step, as shown in Figure 13(right). This plot illustrates the jitter histogram on the y-axis versus the FPGA’s core supply voltage on the x-axis. Thereby, the densities of the histogram are coded in gray-scale (the darker the denser the distribution). It is evident from the figure that performance increases exponentially with the supply voltage. This illustration also shows other interesting facts: For one, almost all voltages have at least two separate humps in their histograms. These are caused by data-dependencies that originate in the different phases $\varphi_{0,1}$. Furthermore, for low voltages, additional peaks appear in the histograms and the separations between the phases increase as well. This can be

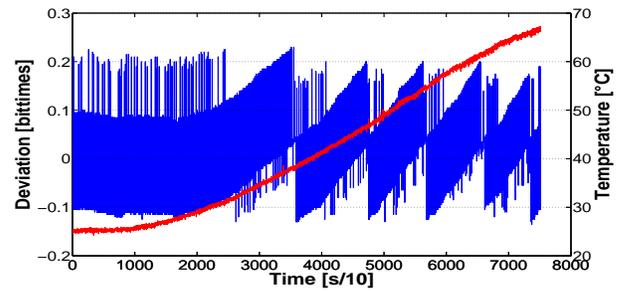


Figure 14. Relative deviation from optimal sending slot and operating temperature.

explained as data-dependent effects caused by different delays through logic stages that are magnified while the circuit slows down. This property is better illustrated in Figure 13(left), where cycle-to-cycle execution jitter is plotted over the supply voltage. The graph appears almost symmetrically along the x-axis, which is caused by the continuous alternation of phases.

We conclude that varying operating conditions not only affect the speed of asynchronous circuits, but also the respective jitter characteristics. In this perspective, slower circuits tend to have higher jitter, which is further magnified by increased quantization errors due to the low sampling rate.

B. Transceiver Unit

The message transmission unit will implement all three compensation methods mentioned at the beginning of Section VI. We intend to use this unit directly in the envisioned asynchronous TTP controller, thus we need to examine the gained precision of this design with respect to timeliness. The interface from the controlling (asynchronous) host to the sub-design of Section VI-A is realized as dual-ported RAM: Whenever the bus transceiver receives a message, it stores the payload in combination with the receive-timestamp⁷ in RAM and issues a receive-interrupt. Likewise, the host can request to transfer messages by writing the payload and the estimated sending time into RAM and asserting the transmit request.

During reception of messages, the circuit can continuously recalibrate itself to the respective baudrate, as Manchester code provides at least one signal transition per bit. However, between messages and during the asynchronous node’s sending slot, resynchronization is not possible. In these phases we need to rely on the correctness of *ref-time*. The Start-Of-Frame sequence of each message must be initiated during a relatively tight starting window, which is slightly different for all nodes and is continuously adapted by the TTP’s distributed clock synchronization algorithm. Failing to hit this starting window is an indication that the node is out-of-sync.

As we are interested in the accuracy of hitting the starting window, we configured the controlling host in a way that it triggers a message-transmission 25 bittimes after the last bit of an incoming message. We simultaneously heated the

⁷We define the internal time to be the number of ticks of signal *ref-time*, i.e., the number of execution steps performed by the bus transceiver unit.

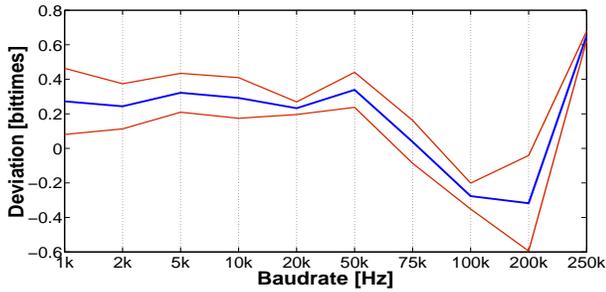


Figure 15. Mean relative deviation from optimal sending point vs. baudrate.

system from room temperature to about 68°C to check on the expected robustness against the respective delay variations. The results are shown in Figure 14, where the deviation from the optimal sending-point (in units of bittimes) and the operating temperature are plotted against time. Similar to Figure 12 one can see that while the circuit gets warmer (and thus slower), the deviation steadily increases. As soon as the accumulated changes in delay can be compensated by the low-level measurement circuitry (i.e., $ref-val$ decreases), the mean deviation immediately jumps back to about zero. We can see in the figure that the timing error is in the range from approximately -0.1 to $+0.2$ bittimes, which will surely satisfy the needs of TTP.

The next property we are interested in is the circuit’s behavior with respect to different baudrates. Although low bitrates have the advantage of minimizing the quantization error, jitter has much more time to accumulate compared to high data rates. It is thus not necessarily true that lower baudrates result in a more stable and precise time reference. On the other hand, if the data rate is too high, it is not possible to reproduce τ_{ref} correctly, and even small changes of the reference value $ref-val$ lead to large relative errors in the resulting signal period. The optimum baudrate will therefore be located somewhere between these extremes. Figure 15 illustrates this by plotting the mean deviations of the optimum sending points versus the bitrate (the “corridor” additionally shows the respective standard deviations). Notice that the y-axis shows the relative deviation in units of bit-times. Therefore, for example, the absolute deviation of the 1kHz bit-rate is more than 50 times larger than that of 50kHz. Clearly, TTP does not support baudrates as low as 1kHz. Reasonable data rates are at least at 100kHz and above (up to 4Mbit/s for Manchester coding). Our current setup allows us to use 100kbit/s for communication with acceptable results. However, we hope to be able to achieve 500kbit/s in our final system setup (with a more sophisticated development platform and a further optimized design).

Finally, we take a look at the accuracy of the generated time reference for different baudrates. Figure 16 therefore shows the mean relative (again in units of bit-times) and the absolute (in seconds) deviations of the actual reference periods from their nominal values. For all baudrates, the relative deviations are within a range of approximately ± 0.01 bit-times, or $\pm 1\%$,

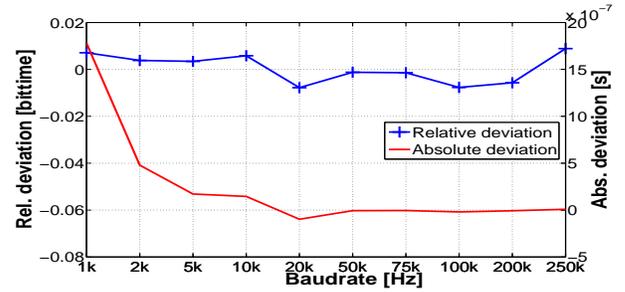


Figure 16. Mean relative/absolute deviation from optimum bit-time.

while the absolute timing errors are significantly larger for baudrates below 50kbit/s.

VII. CONCLUSION

In this article we introduced the research project ARTS, provided information on the project goals and explained the concept of TTP. We proposed a method of using TTP’s bit stream to generate an internal time reference (which is needed for message transfer and most high-level TTP services). With this transceiver unit for Manchester coded messages we performed measurements under changing operating temperatures and voltages. The results clearly show that the proposed architecture works properly. The results further indicate that the achievable precision is in the range of about 1%. This is not a problem while other (synchronous) nodes are transmitting messages, as resynchronization can be performed continuously. However, during message transmission of our node, the design depends on the quality of the generated reference time. Our measurement show that we are able to hit the optimum sending point with a precision of approximately ± 0.3 bit-times (assuming an interframe gap of 25 bits), which should be enough for the remaining nodes to accept the messages.

The presented approach is of course not only limited to TTP, although it clearly is optimized for TTP’s unique properties. For instance, receiver and transmitter modules for all kinds of asynchronous serial protocols (e.g., UART) can be implemented using our solution. The important — and probably limiting — part is, independently of the application, to find suitable resynchronization points or patterns. While these are provided by TTP on a periodic, regular and fixed basis, a start-bit of defined length or a special bit pattern could be used for other communication protocols. Generally speaking, self-timed or (Q)DI asynchronous circuits are difficult to use if strictly timed actions need to be performed, because there are no events of defined duration available. With our approach, however, a basic notion of time can be established even in the absence of a highly stable clock signal.

There still is much work to be done. The presented temperature and voltage tests are only a relatively small subset of tests that can be performed. One of the most interesting questions concerns the dynamics of changing operating conditions: How rapidly and aggressively can the environment change for the asynchronous TTP controller to still maintain synchrony with the remaining system? It should be clear from

our approach that an answer to this question can only be given with respect to the concrete TTP schedule, as message lengths, interframe gaps, baudrate, etc. directly influence the achievable precision of our solution. The next steps of the project plan include the integration of the presented transceiver unit into an asynchronous microprocessor, the implementation of the corresponding software stack, and the interface to the (external) application host controller. Once the practical challenges are finished, thorough investigations of precision, reliability and robustness of our asynchronous controller will be performed.

REFERENCES

- [1] C. J. Myers, *Asynchronous Circuit Design*. Wiley-Interscience, John Wiley & Sons, Inc., 605 Third Avenue, New York, N.Y. 10158-0012: John Wiley & Sons, Inc., 2001.
- [2] J. Sparso and S. Furber, *Principles of Asynchronous Circuit Design - A Systems perspective*. MA, USA: Kluwer Academic Publishers, 2001.
- [3] N. Miskov-Zivanov and D. Marculescu, "A systematic approach to modeling and analysis of transient faults in logic circuits," in *Quality of Electronic Design, 2009. ISQED 2009.*, March 2009, pp. 408–413.
- [4] H. Kopetz and G. Grundsteidl, "TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems," *Symposium on Fault-Tolerant Computing, FTCS-23*.
- [5] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. MA, USA: Kluwer Academic Publishers, 1997.
- [6] *AS8202NF - TTP-C2NF Communication Controller*, Revision 2.1 ed., TTTech Computertechnik AG and www.austriamicrosystems.com.
- [7] M. E. Dean, T. E. Williams, and D. L. Dill, "Efficient Self-Timing with Level-Encoded 2-Phase Dual-Rail (LEDR)," in *Proceedings of the 1991 University of California/Santa Cruz conference on Advanced research in VLSI*. Cambridge, MA, USA: MIT Press, 1991, pp. 55–70.
- [8] A. McAuley, "Four state asynchronous architectures," *IEEE Transactions on Computers*, vol. 41, no. 2, pp. 129–142, Feb 1992.
- [9] D. Linder and J. Harden, "Phased logic: supporting the synchronous design paradigm with delay-insensitive circuitry," *IEEE Transactions on Computers*, vol. 45, no. 9, pp. 1031–1044, Sep 1996.
- [10] K. Fant and S. Brandt, "NULL Convention LogicTM: a complete and consistent logic for asynchronous digital circuit synthesis," in *ASAP 96. Proceedings of International Conference on Application Specific Systems, Architectures and Processors, 1996.*, Aug 1996, pp. 261–273.
- [11] M. Ferringer, "Coupling asynchronous signals into asynchronous logic," *Austrochip 2009, Graz, Austria*, <http://www.vmars.tuwien.ac.at/php/pserver/extern/download.php?fileid=1735>.
- [12] M. Simlastik and V. Stopjakova, "Automated synchronous-to-asynchronous circuits conversion: A survey," pp. 348–358, 2009.
- [13] J. Lechner, "Implementation of a design tool for generation of fsl circuits," Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2008.
- [14] Maxim-IC, "Microcontroller Clock-Crystal, Resonator, RC Oscillator, or Silicon Oscillator?" 2003, application Note 2154. http://www.maxim-ic.com/appnotes.cfm/an_pk/2154.
- [15] F. Bala and T. Nandy, "Programmable high frequency RC oscillator," in *18th International Conference on VLSI Design.*, Jan. 2005, pp. 511–515.
- [16] A. Winstanley, A. Garivier, and M. Greenstreet, "An event spacing experiment," in *Eighth International Symposium on Asynchronous Circuits and Systems, 2002. Proceedings.*, April 2002, pp. 47–56.
- [17] S. Fairbanks and S. Moore, "Analog micropipeline rings for high precision timing," in *10th International Symposium on Asynchronous Circuits and Systems, 2004.*, April 2004, pp. 41–50.
- [18] M. Ferringer, G. Fuchs, A. Steininger, and G. Kempf, "VLSI Implementation of a Fault-Tolerant Distributed Clock Generation," in *21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2006. DFT '06.*, Oct. 2006, pp. 563–571.
- [19] S. Fairbanks and S. Moore, "Self-timed circuitry for global clocking," 2005, pp. 86 – 96.
- [20] V. Zebilis and C. Sotiriou, "Controlling event spacing in self-timed rings," in *11th IEEE International Symposium on Asynchronous Circuits and Systems, 2005. ASYNC 2005.*, March 2005, pp. 109–115.
- [21] J. Ebergen, S. Fairbanks, and I. Sutherland, "Predicting performance of micropipelines using charlie diagrams," mar-2 apr 1998, pp. 238 –246.
- [22] Tektronix, "Understanding and Characterizing Timing Jitter," 2003.
- [23] M. Shimanouchi, "An approach to consistent jitter modeling for various jitter aspects and measurement methods," in *Proceedings of IEEE International Test Conference, 2001.*, 2001, pp. 848–857.
- [24] I. Zamek and S. Zamek, "Definitions of jitter measurement terms and relationships," in *Proceedings of IEEE International Test Conference, 2005. ITC 2005.*, Nov. 2005, pp. 10 pp.–34.
- [25] D. W. Allan, N. Ashby, and C. C. Hodge, "The Science of Timekeeping," 1997, application Note 1289. http://www.allanstime.com/Publications/DWA/Science_Timekeeping/TheScienceOfTimekeeping.pdf.
- [26] D. Howe, "Interpreting oscillatory frequency stability plots," in *IEEE International Frequency Control Symposium and PDA Exhibition, 2002.*, 2002, pp. 725–732.
- [27] B. Butka and R. Morley, "Simultaneous switching noise and safety critical airborne hardware," in *IEEE Southeastcon, 2009. SOUTHEASTCON '09.*, March 2009, pp. 439–442.