On the Appropriate Handling of Metastable Voltages in FPGAs*

Thomas Polzer[†], Robert Najvirt[‡], Florian Beck and Andreas Steininger[§]

*Vienna University of Technology/Embedded Computing Systems Group,
A-1040 Vienna, Treitlstrasse 3, Austria*

[†]*tpolzer@ecs.tuwien.ac.at*

[‡]*rnajvirt@ecs.tuwien.ac.at*

[§]*steininger@ecs.tuwien.ac.at*

Received 10 July 2015

Accepted 3 August 2015

Published 16 November 2015

The significant process, voltage and temperature (PVT) variations seen with modern technologies make strictly synchronous design inefficient. Asynchronous design with its flexible timing is a promising alternative, but prototyping is difficult on the available FPGA platforms which are clock centric and do not provide the required functional primitives like mutual exclusion or Muller C-elements. The solutions proposed in the literature so far work nicely in principle but cannot safely handle metastability issues that are inevitable even at some interfaces in asynchronous designs. In this paper, we propose reliable implementations of the fundamental function blocks required to safely convert potential intermediate voltage levels that result from metastability into late transitions that can be reliably handled in the asynchronous domain. These are high- and low-threshold buffers as well as a Schmitt-trigger. We give elaborate background analysis for the proposed circuits and also present the associated routing constraints to make the Schmitt-trigger circuit work properly in spite of the uncertain routing within FPGAs. Furthermore, we propose a procedure for an “*in situ* reliability assessment” of the specific Schmitt-trigger element under consideration, which also applies to metastability containment with high- or low-threshold buffers only. Our proof of concept is based on experimental results for both Xilinx and Altera FPGA platforms.

Keywords: Asynchronous; FPGA; metastability; Schmitt-trigger; low-threshold; high-threshold; measurement.

[§]Corresponding author.

*This research was supported by the SIC project (grant P26436-N30) of the Austrian Science Fund (FWF).

This is an Open Access article published by World Scientific Publishing Company. It is distributed under the terms of the Creative Commons Attribution 4.0 (CC-BY) License. Further distribution of this work is permitted, provided the original work is properly cited.

1. Introduction

Asynchronous circuits are receiving more and more interest since they provide a natural way of handling the significant process, voltage and temperature (PVT) variations seen with modern ASIC technologies and they lend themselves to low-power design. Their operation is not based on a rigid global clock but rather on local handshakes, which, in an abstract sense, form control loops for the flow control. Naturally, this different design paradigm also necessitates different basic function blocks. While synchronous designs are dominated by flip flops, in an asynchronous circuit, we rather find Muller C-elements, latches, delay elements and mutual exclusion (mutex) elements, apart from the combinational gates like NAND and NOR, which we find in both approaches. These “asynchronous specific” elements are often burdensome to implement in a custom ASIC (as they are rarely part of a standard library), and in an Field Programmable Gate Array (FPGA) they are simply not available, which makes prototyping of asynchronous circuits tedious. The popular FPGA architectures are all clearly optimized for synchronous designs and the lack of asynchronous basic blocks has already been recognized in several publications where workarounds for their implementation have been proposed. However, a notorious problem that has not always been correctly addressed is metastability. The standard solution in synchronous designs for handling potential metastability problems is the use of synchronizers and it is relatively well understood how to properly design them. In the asynchronous world, metastability can occur as well, but, as will be outlined later in this paper, its handling is fundamentally different and therefore calls for different measures. More specifically, the mission is to safely turn an intermediate voltage level that usually results from metastability into a well-defined HI or LO level by means of high- or low-threshold buffers or a Schmitt-trigger. Again, these are not available in FPGAs. In this paper, we will analyze this conversion in more detail, investigate which of these measures is required when and we will propose a safe and systematic way of implementing the required functions in an FPGA. A specific contribution will be a novel circuit structure for the implementation of a Schmitt-trigger. On this foundation, a value safe solution to handling metastability without upsets in asynchronous circuits can then be implemented in the FPGA prototype.

2. Background

Fundamentally, every state-holding element in digital logic is prone to metastability. For simplicity, however, let us consider a simple storage cell constructed from two cross-coupled inverters as an example here, for which the logic level at its output represents its internal state. This element has two stable states (“HI” and “LO”) and moving from one state to the other requires some energy which is usually provided by applying an appropriate input signal. The way this input is provided depends on the specific type of storage element, like RS-latch, D-latch, flip-flop or Muller C-element. In any case, ultimately there is a pulse applied to the storage cell that is strong

enough to make it flip. In this context, “strong” implies that the pulse is long enough and its voltage level high (or low) enough to make the state change, i.e., it must have sufficient energy. Should the pulse be too weak, it will not be able to make the state flip. However, since the energy of the pulse is a continuous quantity, it will always be possible to find a pulse that has just the right energy to start making the cell flip, but then turns out too weak to fully flip it. In that case, the cell will remain undecided about which state to assume for an essentially unbounded time, although the slightest disturbance will finally make the state move to HI or LO. This state is referred to as the metastable state and an often cited analogy is a ball balancing on the top of a hill. It has been shown that experiencing this metastable state is unavoidable whenever a discrete output decision (like the state to assume) is based on a continuous input quantity (like the pulse energy) and it is further known that the duration of this state cannot be bounded.¹ During this metastable state the storage element will present an intermediate voltage level (i.e., a voltage between HI and LO) at its output. Depending on their individual actual threshold voltage, this level may be (correctly!) interpreted by subsequent inputs as HI or LO or it may cause subsequent elements to propagate the metastable state of the signal or, in case of sequential elements, become metastable as well. This means that two inputs receiving the same metastable signal may decide differently about its logic state, which is obviously very dangerous. There are two fundamental ways of handling this situation:

One option is to simply wait until the cell has clearly decided for one or the other state, i.e., until the metastable state has decayed. Then one can be sure to read a well-defined voltage level that is uniquely understood by all subsequent inputs. This approach is called *value-safe* and its drawback is that the maximal time one has to wait for the metastability to decay is unbounded, thus detection of correct and stable values is needed to dynamically determine the waiting time.

In practice, one is often forced to have a result within a given time limit, like in a synchronous system, where all computations need to be finished within a clock period. In such a context, one cannot wait for an unlimited time. What is done instead is to allow for a certain resolution time, and then just use the result, accepting the fact that with a non-zero probability the state is still undecided at that point. Considering the possible ambiguous interpretation of the metastable voltage, this *time-safe* solution implies accepting the computation to fail with a certain probability. In case this delayed output decision leads to an incorrect or metastable value being captured by a successor stage (next sequential element), we have experienced what is called a metastable upset.

Synchronous designs by their nature demand for the time safe approach. Here the arrival of a data transition close to the active clock edge may create a marginal pulse that brings the flip flop to the metastable state. In a proper design, this does not occur within the synchronous timing domain, but at the clock domain boundaries such events are inevitable. Here synchronizers are commonly used, most often they

are built from a chain of flip flops. Their basic principle is to extend the permitted resolution time beyond a single clock cycle, thus making a metastable upset less probable.

$$FR = \lambda_{\text{dat}} \cdot f_{\text{clk}} \cdot T_w \cdot e^{-t_{\text{res}}/\tau_c}. \quad (1)$$

Equation (1) describes the expected rate of upsets (FR) for a system with clock frequency f_{clk} , data changing with an average rate of λ_{dat} and a flip flop with parameters T_w and τ_c . Here it can be seen that there is an exponential dependence of FR on t_{res} . It also becomes apparent that FR can never become zero no matter how conservatively t_{res} is chosen. Also, one should be aware that a choice of large t_{res} implies a performance penalty. In the face of the ample PVT variations seen in recent technologies, the conservative design of a synchronizer for a given target FR may render this penalty significant. Furthermore, considering that modern SoCs comprise a multitude of uncorrelated clock domains, the number of synchronizers becomes substantial, making this tradeoff between FR and performance penalty even worse.

Asynchronous design, in contrast, employs the value-safe approach, particularly in the context of the delay-insensitive timing model.² Here a handshake loop between the communication partners adapts the pace of processing data items to the current speed of the processing hardware. Still, “external” activities that are not aligned with the handshake cycle need to be synchronized to it. Here the mutex circuit is the element of choice. Unlike the synchronizer, it can perform its task of deciding which of its requests occurred earlier without introducing a systematic risk of upsets. The reason why the mapping from a continuous space (relative time of arrival of requests) to a discrete one (identification of the earlier one) is now safely possible is simply because the mutex operates in a value safe fashion, i.e., its maximal decision time is unbounded. This makes the use of the asynchronous style attractive, where the application does not demand a time safe solution.

It should be noted here that another metastability manifestation is possible, namely oscillation. This phenomenon is observed if the pulse applied to the input of the storage loop is shorter than the round trip delay of the loop. In that case, the pulse can “cycle” within the loop, creating an oscillating output voltage. The precondition for this, however, is that the loop delay is dominated by pure delay Δ rather than inertial (RC) delay. In fact, it has been established that depending on the ratio of Δ/RC , a storage element will either oscillate or show the intermediate voltage in case of metastability, but a given circuit cannot exhibit both behaviors.³ Normally, storage cells are designed to avoid oscillation, but for storage loops built in FPGAs, more specifically those realized by feedback loops involving multiple look-up tables (LUTs) and/or having unfortunate routing, one may also experience oscillations. In this paper, however, we will assume that the routing is sufficiently optimized in all cases and hence focus on the intermediate voltages only.

3. Related Work

FPGAs have early been recognized as attractive targets for the implementation of asynchronous logic (see, e.g., Ref. 4), since they represent an appealing prototyping platform for this design paradigm, whose delay insensitive nature can furthermore easily accommodate their significant and unpredictable routing delays. However, delay insensitive behavior can typically be achieved on the level of basic building blocks only while inside these blocks, critical race conditions can result in glitches leading to malfunction. As a consequence, the hazard free implementation of a Muller C-element,^{5–8} mutex⁹ and threshold gate¹⁰ in FPGA technology has been a major concern in many publications. In all these publications, however, potential metastability of these elements has not been addressed. A notable exception is Ref. 11 where, in the context of a gated clock implementation for an asynchronous GALS wrapper, a synchronizer design is proposed that more or less successfully tries to mask out intermediate voltage levels. This synchronizer, however, is not generic; it handles one-sided (down-) transitions of the data only, for up-transitions the synchronizer would fail and synchrony between data and clock is assumed in the given context. Moreover, no rigorous argumentation and/or measurements of the claim of metastability containment are provided.

A mutex implementation has been proposed by Seitz that makes sure that no false output is generated even in case the mutex gets metastable internally (recall that this cannot be avoided).¹² The trick here is to use a low-threshold inverter connected to the output of the mutex' actual bistable element. It works as follows: Imagine a mutex core cell built from two cross-coupled NAND gates as shown in Fig. 1. When one request R_i is activated (HI), the mutex will activate (pull to LO) the corresponding internal grant GR_i . In case both requests are activated concurrently, the mutex becomes metastable with both its internal grant outputs being at an intermediate voltage. The low-threshold inverters will consistently interpret this voltage as HI and hence issue a LO for both external grant outputs G . This means no grant is activated during metastability resolution, which is a safe solution. Only after metastability has resolved, one of the grant outputs will be activated as intended. In Seitz's implementation, the low-threshold inverter is realized by a transistor circuit that relies on one output providing the power supply for the other one, thus building an extra level of safety.

Fig. 1. Traditional mutex implementation according to Seitz (Ref. 12).

Fig. 2. Possible metastability behaviors of a Muller C-element, outputs of low- (Q_l) and high-threshold buffers (Q_h) connected to z (color online).

We have investigated more generally the effect of using low- and high-threshold buffers for containing metastable voltages.¹³ As shown in Fig. 2, there are the following possible reactions of a high-threshold filter (here we use a buffer to simplify the explanation) to a metastable input:

Case A. The input was LO before: The metastability raises the voltage in the storage loop to an intermediate level, but it stays below the (high) threshold, so the output remains LO until the metastability resolves. In case it finally resolves to LO, we do not see any reaction at the buffer output, the metastability has been successfully suppressed. Should it resolve to HI we see a clean transition, which is late as it occurs only after metastability has resolved.

Case B. The input was HI before: When going down to the intermediate metastable level, the voltage crosses the threshold and immediately causes a falling transition at the buffer's output. In case the metastability finally resolves to LO, the voltage level continues to fall without any threshold crossing, so overall we see a clean transition with nominal delay. However, should the metastability resolve to HI, we see another transition at the buffer output. This means we have experienced a negative glitch whose width equals the duration of the metastable state.

In a similar fashion, the possible behaviors of a low-threshold buffer can be deduced: no transition nominally delayed transition, late transition or positive glitch. Note that the glitch always occurs in one specific case, namely an initially HI output going metastable and resolving back to HI, in case of a high-threshold buffer and a LO – metastable – LO sequence in case of the low-threshold buffer.

In a value-safe environment, a late transition can be accepted and only the glitch constitutes an undesired behavior. According to the above analysis it can be avoided by using a low-threshold while the input is HI and a high-threshold while the input is LO. This resembles the behavior of a Schmitt-trigger. Consequently, the Schmitt-trigger is identified as the method of choice by Polzer *et al.* for transforming the intermediate metastable voltage into well-defined levels without producing glitches.¹³

4. Proposed High- and Low-Threshold Implementation in FPGA

As outlined in the previous chapter, a single-sided threshold can be used to protect one direction of a transition from glitches (low-threshold for down- and high-threshold for up-transitions). This can be useful for function blocks in which one input transition is critical only. A prominent example is the mutex, where the activation of the request is critical, while the release is uncritical, recall Sec. 3. Generally, FPGAs do not provide explicit high- or low-threshold functionality for their internal gates/LUTs. However, “high-threshold” actually applies to any threshold voltage higher than the intermediate voltage presented in the metastable state. So if the threshold voltage of a gate input does not incidentally match its source’s intermediate voltage, it will work as high- or low-threshold filter anyway. Under the assumption that the gates’ threshold voltages are somewhat centered but spread around the intermediate voltage, one may construct a high-threshold buffer by ANDing the same signal connected to several inputs — thus effectively selecting the highest of the inputs’ thresholds — and a low-threshold by ORing several inputs. This has already been proposed in the literature (e.g., in Ref. 14). Mapping this concept to FPGAs would imply using a LUT as a threshold buffer and defining an AND or OR over all its inputs (which are connected to the same signal) to reach the desired high- or low-threshold, respectively.

Given the high number of infrastructure elements like buffers, switches and multiplexers present in the signal paths of modern FPGAs, it would be naive to assume that the metastable output of one sequential cell, be it a flip flop or a Muller C-element, will be directly seen by the input of a subsequent LUT. But how do those infrastructure elements change the picture? To better understand this, let us try to establish a model.

Figure 3 illustrates a very general case: The output of the sequential cell (i.e., the one that is suspected to become metastable) is propagated over a number of such infrastructure elements (let us simply call them buffers in the following) before it reaches the subsequent LUT. When we convey that output to two inputs of the same LUT (like in case we plan to AND or OR them to attain high- or low-threshold), we have, in the most general case, a common sub-path and then a fork to two disjoint sub-paths. All these sub-paths may or may not comprise a number of buffers.

Fig. 3. Model comprised of buffers and fork.

4.1. Propagation model — analog voltages

To model the propagation of an intermediate voltage over a buffer, we can look at the characteristics of a buffer as shown in Fig. 4.

Here we see that for a clear LO at the input, the buffer will deliver a clear LO at its output and in the same way a HI output for a HI input. However, there is an intermediate window of input voltages in which the buffer behaves like an amplifier. More precisely, an input voltage of V_m will produce an output voltage V_M right between the borders for a clear HI and LO, namely V_H and V_L . This output range $[V_L, V_H]$ can be projected back to an input range $[V_l, V_h]$ that should be avoided when a clean output is desired. Note that this model is extremely general by assuming nothing more than amplification of an intermediate input voltage (with a voltage

Fig. 4. Buffer characteristics (color online).

shift on the input and output side); so it is reasonable to claim that other types of infrastructure elements will also exhibit this type of behavior.

When operated in its linear analog range, the buffer characteristics can be approximated by the linear slope, yielding

$$V_{\text{OUT}} = V_M + (V_{\text{in}} - V_m) \cdot A,$$

with A being the amplification, i.e., slope of the characteristics (typically 5 for current technologies). When cascading two such stages, the output of the first stage becomes the input of the second one, yielding

$$V_{\text{OUT},2} = V_{M,2} + [V_{M,1} + (V_{\text{in}} - V_{m,1}) \cdot A_1 - V_{m,2}] \cdot A_2.$$

Here it becomes apparent that the mismatch between “midpoint” output voltage $V_{M,1}$ of stage 1 and “midpoint” input voltage $V_{m,2}$ of stage 2 (i.e., the voltage that brings stage 2 to the midpoint between V_H and V_L) is relevant. We will further call it “offset voltage” $V_{\text{off},1,2} = V_{M,1} - V_{m,2}$ and simplify the above equation as

$$V_{\text{OUT},2} = V_{M,2} + [V_{\text{off},1,2} + V_{\text{off},0,1} \cdot A_1] \cdot A_2.$$

Note that we have used $V_{\text{off},0,1}$ to express how far the input voltage V_{in} deviated from the midpoint. This allows us to express the transfer behavior of an n -stage buffer chain as

$$V_{\text{OUT},n} = V_{M,n} + \sum_{i=1}^n \left(V_{\text{off},i-1,i} \cdot \prod_{k=i}^n A_k \right)$$

and when extracting the first element, namely the one containing the input voltage, from the sum, we get

$$V_{\text{OUT},n} = V_{M,n} + (V_{\text{in}} - V_{m,1}) \cdot \prod_{k=1}^n A_k + \sum_{i=2}^n \left(V_{\text{off},i-1,i} \cdot \prod_{k=i}^n A_k \right). \quad (2)$$

This shows a linear relation between V_{OUT} and V_{in} in the form of

$$V_{\text{OUT}} = k \cdot V_{\text{in}} + d.$$

with proportionality factor

$$k = \prod_{k=1}^n A_k.$$

Note that for a given circuit with a given path of buffers k and d are constants (we disregard PVT variations here). According to this linear relation, the interval of size $V_{\text{CRIT}} = |[V_L, V_H]|$ can be projected to the input interval of size $V_{\text{crit}} = |[V_l, V_h]|$ as

$$V_{\text{crit}} = \frac{V_{\text{CRIT}}}{\prod_{k=1}^n (A_k)}, \quad (3)$$

which means the critical interval at the input that will make the voltage undetermined at the output is shrunk by the gain of each buffer stage along the path.

To obtain a relation on how the midpoint of the last buffer output (or the threshold of the LUT input, respectively) is transformed back to the input of the buffer chain (i.e., where the “threshold” $V_{\text{th,eff}}$ of the first input actually lies), we need to set $V_{\text{OUT},n} = V_M$ and set $V_{\text{in}} = V_{\text{th,eff}}$ in Eq. (2). After some transformations, this yields

$$V_{\text{th,eff}} = V_{m,1} - \sum_{i=2}^n \frac{V_{\text{off},i-1,i}}{\prod_{k=1}^{i-1} A_k}. \quad (4)$$

Here we can observe that while the threshold $V_{m,1}$ of the first stage in the chain has an immediate impact, the offset (and hence threshold) of every subsequent stage is divided by the gain of all its predecessors. This gives evidence that the earlier stages have a much higher influence on the decision about the effective threshold seen by the LUT.

From this model, we can now draw the following conclusions:

- (i) The buffer chain provides a lot of (potential) shifting and amplifying of the initial input voltage’s offset against the midpoint (Eq. (2)).
- (ii) The first stages in the chain are likely to decide about the interpretation of the input voltage (above or below threshold). The later stages receive a highly amplified signal which is very likely to be clear HI or LO already (Eq. (4)).
- (iii) Once one stage has lifted the output voltage clearly above V_H or below V_L locally, then the logic level is well defined for the subsequent stages (if they behave according to the specification). This is not reflected in the model for simplicity.
- (iv) For given buffer parameters, we can project back from the output to determine the range $[V_l, V_h]$ of input voltages that will produce an intermediate output voltage. Here the initial range $[V_L, V_H]$ is divided by the gain of each stage along the path. As a result, a long chain is less likely to still exhibit undecided voltage levels at its output (Eq. (3)).
- (v) If the metastable output voltage V_{meta} of the sequential element happens to fall within this interval, then the metastable voltage has been conveyed to the LUT input, otherwise we already see a clear HI or LO level there. Note that V_{meta} is typically a fixed value with rarely any random fluctuation, so for a given design with given circuit parameters one either always experiences the analog LUT input voltage in case of metastability or never.
- (vi) In that case, the contributions of the individual stages have happened to cancel each other in such a way that the LUT still receives an intermediate input. Only in this case the interpretation of the value by the LUT matters.

At this point, it is also interesting to examine how the forking paths impact this situation: If the decision is already clear at the forking point, then the forks will consistently convey the respective logic levels. Otherwise, the decision is made

somewhere downstream, but again not necessarily at the LUT. In that case, the LUT may receive contradicting interpretations. However, with the above analysis, even a few stages in the common path will render this case very unlikely.

4.2. Propagation model — path delays

Independent from the question of how analog voltages are propagated through the chain of buffers, there is also some delay involved that affects transitions more or less in the same way, irrespective of whether they are viewed on analog and digital level. This delay results from wire delays, switching delays and from limited rise and fall times (i.e., the time from starting a transition until it reaches the threshold for a clear HI or LO). In the following, we will not distinguish between these different sources of delay and just assume that we will experience some delay in the signal transitions and that this delay may be dependent on the type of transition (falling or rising). We can expect the common path to impose the same delay Δ_{cmn} for both ends of the fork, but individual delays Δ_a and Δ_b on the individual paths. The latter will result in a skew $S_{ab} = \Delta_b - \Delta_a$.

4.3. Projections on the effect of AND and OR gating

Our analysis in Sec. 4.1 suggests that intermediate voltages will hardly ever reach the input of a LUT. However, this claim still needs to be validated. So in the following, we will first describe the setup of validation experiments and then anticipate their outcome, once for the case that intermediate voltage can reach the LUT and once for the case that it does not. This will finally allow us to draw the right conclusions from the observations we will make in our experiments.

Experiment setup: We will connect (over routing paths and buffers hidden inside the FPGA) all four inputs of the LUT to the output of the sequential element. As it is not possible to faithfully convey glitches that may show up at the LUT output, over the FPGA's I/O buffers and pins due to their bandwidth limitations, we have employed an analysis circuit implemented internally on the FPGA. More specifically, we use the circuit from Ref. 15 for detecting *late transitions* allowing a very detailed analysis of metastability behavior.^a While this circuit is designed specifically for analyzing flip flop targets, it can be adapted for the purpose of Muller C-element targets.¹⁶ Its principle is to measure the upset rate produced by the target flip flop for a given resolution time. Basically, this indicates the existence of late transitions. By clever bookkeeping about previous samples, many further details (actually all those illustrated in Fig. 2) about the behavior can be extracted (see Ref. 15). For the purpose of the approach presented here, it suffices to find out whether any glitches are present. So in fact, one does not need the full-fledged implementation from Ref. 15 — a suitably

^aNote that we cannot observe the analog behavior of the flip flop output directly since we rely on digital measurements inside the FPGA.

reduced version will do the job, namely allowing us to clearly judge which type of threshold behavior we have. According to Fig. 2 with a high-threshold we see negative glitches (only!) and with low-threshold we see positive glitches.

The measurement circuit is very sensitive to routing delays. To achieve optimal results, the circuit must be rigorously constrained. Therefore, the most important elements, like the device under test or the detector flip flops, are manually placed into adjacent slices. The routing delay of all other critical signals, like clock domain crossings, are controlled by specifying their maximum allowed delays. For more details, see Sec. 5.4.

Case 1. Intermediate LUT input voltage: Let us assume that the chain of buffers still does not resolve an intermediate output voltage produced by the sequential element to a clean HI or LO and so the LUT inputs can decide. Let us further assume that the LUT's input thresholds are well distributed over the range $[V_L, V_H]$, so when using all four inputs to judge the same signal level, we will find some of them, namely those with a low-threshold, classifying the input as HI while others, those with a high-threshold, rate it as a LO.

Figure 5 summarizes the postulated effects of different thresholds for signal transitions (metastability passing through a phase of intermediate output voltage can be viewed as a very slow signal transition). For a buffer with a high-threshold, the delay from the beginning of a falling transition to the threshold crossing that marks the change of the output is quite small (Fig. 5(a), red curve) while it increases for decreasing thresholds (green and blue curves). For a rising edge on the other hand (Fig. 5(b)), the effect is reversed, therefore for a high-threshold, the propagation delay is higher than for a low-threshold, respectively. Depending on the slew rate of the signal, the effect is more or less pronounced, so for deep metastability it will be clearly visible.

When studying the conjunction or disjunction of multiple inputs, each input reacts as described previously. To create an output transition, the following rules must be met:

- For a rising transition at the output of an AND gate, all inputs must have seen a rising transition.

Fig. 5. Effects of different input thresholds (a) Falling edge. (b) Rising edge (color online).

- For a falling transition at the output of an AND gate, the first falling transition on any input suffices.
- For a rising transition at the output of an OR gate, the first rising transition on any input suffices.
- For a falling transition at the output of an OR gate, all inputs must have seen a falling transition.

So in essence, ANDing the inputs will bring the highest of the thresholds into effect, thus yielding a high-threshold overall behavior while ORing will result in the lowest threshold becoming decisive and hence form a low-threshold behavior. This is exactly what is sometimes proposed in the literature. So when changing from AND to OR, we should see the transitions move accordingly. However, as we will discuss below, the same will happen to clean transitions (with non-zero transition time), i.e., those that already passed a low- or high-threshold buffer.

However, what still remains for a distinction is the different type of glitch motivated in Fig. 2. If ANDing and ORing really changes the type of threshold, then we can expect to see the type of glitch change accordingly (positive glitch for OR with its low-threshold and negative glitch for AND with its high-threshold). Otherwise, the type of glitch will be determined by a buffer stage earlier in the chain and hence no more be changed by the decision to use AND or OR. In that latter case, we can clearly conclude that ANDing and ORing do not yield the intended effect, however we cannot rule out for sure that the LUT input voltage may have been intermediate, with an intermediate input voltage and all LUT thresholds on the same side, we would observe the same behavior, namely the input voltage being consistently interpreted by all LUT inputs.

Case 2: Different path delays: Let us now follow the alternative argumentation and assume we see consistent logic levels at all LUT inputs, but, due to skew transitions arrive with different delays. On top of that metastability of the sequential element will cause late transitions and glitches. The effective threshold deciding about whether the metastable voltage is HI or LO, however, is somewhere at the beginning of the chain, maybe already the output buffer of the sequential element itself.

Now, interestingly a similar effect of moving transitions through ANDing and ORing can be seen, this time, however, with LUT inputs that have matching thresholds but different delays (skew). Figure 6 illustrates how our late transition detector (LTD) will react to that: To understand the figure, it is important to know that our LTD needs to be calibrated at the start of the measurement. This calibration aligns the time scale to the nominal output delay of the sequential element, which we assume to be the same for rising and falling edges. So what we can observe afterward for a single input is whether, among the late transitions, rising or falling ones experience a larger delay. In addition, when ANDing or ORing two LUT inputs we will pick the respective earlier or later one of the two differently delayed transitions. Figure 6(a) illustrates the case of small skew: The OR output reflects the

Fig. 6. Effects of signal delay on low-threshold buffers. (a) Small difference. (b) Large difference (color online).

earlier rising and the later falling transition, thus amplifying the delay difference initially caused by the low-threshold. The AND, in contrast, counteracts the initial difference, moving the edges closer together. In Fig. 6(b), we see the case of large skew (relative to the initial difference). Here again the OR moves the edges apart, but the AND shows a special behavior: Although locally (per input) the rising edge still occurs before the falling edge (due to the low-threshold), the later rising edge (at input b) occurs after the earlier falling edge (at input a) in the global view, due to the significant skew. This makes the falling edge now appear earlier at the AND output than the rising edge — the former seems to have overtaken the latter. So depending on the size of the skew, we can expect one of these two scenarios. Note that we have assumed equal thresholds for both inputs, so this is an effect of skew only, not of combining high- and low-threshold. In particular, we can also expect to see these effects with clean transitions, i.e., in case the preceding buffers have determined the type of threshold already.

4.4. Measurement results

To confirm or disprove the different scenarios discussed in the previous sections, we have performed physical experiments in FPGAs, namely Altera Cyclone IV EP4CE115. More specifically we have implemented a sequential element that is artificially driven into metastability (on a random base, for details see Ref. 15) and routed its output to all four inputs (a , b , c , d) of a LUT. For our experiments, we configured the LUT to either use a single input only or perform different logic combinations of the inputs, namely pairwise AND of two inputs, pairwise OR of two inputs (in both cases we performed experiments for all permutations of pairs), AND of all four inputs and OR of all four inputs. In all these cases we took care to leave the routing unchanged, which we accomplished by implementing different LUT masks using engineering change orders. In all cases, the LUT output was evaluated by our LTD.¹⁵ As usual with this approach, we manually aligned (time shifted) the measured curves to compensate for unknown calibration delays.

Fig. 7. Delay measurement results (grouped by input/output). (a) Relative delay of the LUT inputs (a , c). (b) Resulting LUT outputs (a , c). (c) Relative delay of the LUT inputs (a , b) and (d) Resulting LUT outputs (a , b) (color online).

Based on our theoretical considerations from above, we can draw the following conclusions from our measurement results (only part of these will be shown in detail due to space restrictions):

Figures 7(a) and 7(c) illustrate the results for channels a , b and c individually, i.e., not combined with any other. Here we see that the rising edge consistently occurs before the falling edge which indicates a low-threshold. This conclusion is confirmed by the fact that we observe positive glitches only ($\uparrow\downarrow$) and not a single negative glitch. Keep in mind that this low-threshold behavior does not necessarily apply to the LUT input but rather to the whole buffer chain overall.

When combining the inputs using an OR function within the LUT, the expected behavior, namely the increase of the response time difference between rising and falling edges was visible in all measurements (see Fig. 7(b) and Fig. 7(d)).

When using an AND function both possible scenarios described above could be observed. Although the fact that in Fig. 7(b) the rising edges are slower than the falling edges could be mistaken for an indication for a high-threshold behavior, we can clearly conclude from the exclusive occurrence of positive glitches that we still have a low-threshold and are just observing the case illustrated in Fig. 6(b) from our analysis section. For the behavior recorded in Fig. 7(d), we have the case from Fig. 6(a), which makes the conclusion to have low-threshold straightforward.

An interesting detail result shown in Fig. 7 is that the OR seems to increase the width of the glitches. The reason is that for the rising edge of the glitch the first arriving transition is sufficient to change the output, while for the falling edge both falling transitions must have arrived. For the AND LUT the glitch is shortened for the same reason.

No matter whether we choose an AND or OR combination for the inputs, in no case do we see negative glitches. This clearly shows that ANDing and ORing is not efficient for selecting the type of threshold. Furthermore, we may see this as a confirmation of our claim that the actual decision about the type of threshold is made by a buffer earlier in the chain and that the LUT already receives clean (albeit delayed, in case of metastability) transitions. However, as outlined above already, this might be explained as well by assuming all LUT inputs to have their threshold at the same side of the intermediate voltage.

To visualize the dependence between the arrival of the input transitions and the output change, we have plotted the results grouped by implemented LUT function and transition polarity in Fig. 8. As can be seen, the changing of the output always

Fig. 8. Delay measurement result (grouped by transition). (a) Falling edges, AND buffer. (b) Rising edges and pulses, AND buffer. (c) Falling edges, OR buffer. (d) Rising edges and pulses, OR buffer (color online).

Fig. 8. (Continued)

directly corresponds to an input change. The only difference between AND and OR LUT function is, whether the first changing or the last changing input is triggering the output change. This is perfectly in line with our theoretical analysis and also confirms that our measurement approach works well.

4.5. Changing the side of the threshold

Our findings so far put us in the position to reliably implement single-sided threshold buffers in an FPGA, actually by leveraging their inherent threshold behavior.^b We have witnessed the Altera Cyclone IV EP4CE115 FPGA to consistently exhibit low-threshold (as discussed above), while our experiments on the Xilinx Virtex 4 FX12 yielded consistently high-threshold for that target. We have seen that ANDing and ORing is not an effective way to select the threshold; in fact there does not seem to be an immediate way of implementing a threshold behavior other than the inherent one in an FPGA at all. When the input of an element already has a certain type of threshold, no provisions can change this already filtered signal to appear as if it was filtered with the opposite threshold. For still implementing threshold filtering for that opposite side, we thus propose the following strategy (explained at the example of a D-flip flop):

- Invert the data input.
- Employ the inherent threshold filtering of the flip flop's output.
- Invert the filtered output.

^b As the FPGA vendors do not specify the thresholds of the internal function blocks, we cannot generalize our statement to other FPGAs. In any case, however, it should always be possible to implement one type of threshold in a given FPGA and use our concepts below. A respective strategy will be given in Sec. 6.

This strategy can be mapped to other bistable functions, an example for the Muller C-element is shown in Fig. 10 in the lower signal path.

5. Proposed Schmitt-Trigger Implementation in FPGA

5.1. Circuit

As already pointed out in Sec. 3, a Schmitt-trigger requires a combination of high- and low-threshold functions that are appropriately controlled depending on the current state. The circuit shown in Fig. 9 illustrates the principle of our proposed circuit based on a multiplexer (mux) switching the currently effective thresholds. The upper path gets selected when the mux output is LO and it is responsible for a clean up-transition, which it can well accommodate due to the high-threshold buffer. As soon as the up-transition is finished, the mux selects the lower path which can perform a clean down-transition.

Unfortunately, in FPGAs we only have one type of thresholds and need to apply the inversion principle as described in the above section for building the other one. In the following, we assume the FPGA has inherent high-threshold filtering like we saw in our Xilinx FPGA — the low-threshold filter requires the inversions. Note that the required inversions of the inputs for low-threshold filtering imply that for the implementation of the Schmitt-trigger we need two instances of the gate experiencing metastable states, the Muller C-element in this example. Figure 10 shows the resulting circuit.

Fig. 9. Concept of the proposed Schmitt-trigger implementation.

Fig. 10. Proposed FPGA implementation of a Muller C-element with Schmitt-trigger output.

Note that while not selected, the Muller C-elements are, by virtue of the reset signals, forced to the state that is appropriate when they become selected. This is necessary since due to routing mismatches and non-determinism in metastability resolution the two Muller C-elements may decide differently for input patterns that are (close to) causing metastability. This could cause glitches at the output while switching the mux.

When implementing the Schmitt-trigger circuit, the delays between the duplicated elements and the mux must be carefully controlled. A vital constraint is that the delay from the mux output to its select input must be smaller than the delay from its output to its data inputs through the reset inputs of the basic storage element duplicates (e.g., flip flops or Muller C-elements).

Concerning complexity of the circuit, apart from the duplication of the bistable element, the mux requires an extra LUT. The explicit high-threshold buffers (indicated by an “H” in the figure) can be omitted (see Sec. 4). The inverter in the lower path can of course also be accommodated in the LUT of the mux. Inverted data and reset inputs of the lower bistable element are usually available in FPGAs or may also be accommodated in the implementation of the element. If both are not applicable, separate LUTs would be required.

5.2. Validation experiment setup

Like our high- and low-threshold buffers, we have tested our proposed concept for the Schmitt-trigger on two different platforms, namely on Altera Cyclone IV EP4CE115 and Xilinx Virtex 4 FX12 FPGAs. As targets we used both D-flip flop and Muller C-element. Since an FPGA does not contain Muller C-elements, they were emulated using D-latches and LUTs to implement the set and reset logic. With this implementation, the delays from the LUTs that calculate the set and reset functions to the latch are very critical. We used tight timing constraints to force the implementation tools to place them in close proximity to each other.

Based on our findings from Sec. 4, we leveraged the inherent threshold behavior of our target FPGAs for one side of the threshold while the opposite side of the threshold was realized by the double-inversion presented above. The Schmitt-trigger circuit was built as described in Sec. 5. By adding an additional control signal to the mux at the output, we were able to select between single sided threshold and Schmitt-trigger output modes, allowing us to observe their metastability behavior without changing the FPGA bitstream.

5.3. Measurement results

Figure 11 shows the results for a flip flop in the Xilinx FPGA. We can observe that, as expected, the high-threshold output either produces late transitions or glitches (the cases where it produces no transitions or nominally delayed transitions are not visible in this measurement approach). For the Schmitt-trigger output, we can

Fig. 11. Measurement results for a D-flip flop with high-threshold (single) and with Schmitt-trigger (schmitt) output (color online).

observe that the glitches completely vanished and we have an increased frequency of late transitions instead. This confirms our statements from Sec. 2. For more details about the interpretation of the graphs please refer to Ref. 15.

The results for the Muller C-elements are similar to the ones achieved for the flip flops and are depicted in Fig. 12 again for the Xilinx FPGA. One can clearly see that the pulses which occur in the high-threshold implementation are filtered out in the Schmitt-trigger version.

5.4. Constraints

In this section, we give additional details on the constraints necessary to implement our Schmitt-trigger circuit in an FPGA. As a case study, we use the Virtex-4 version of the Schmitt-trigger flip flop circuit presented earlier in this paper.

Note that our Schmitt-trigger implementation contains concurring feedback loops, namely (i) the one from the mux output back to its select input and (ii) those leading from the mux output to the reset inputs of the (duplicated instances of the) sequential element and then further on to the mux data inputs. Here it is imperative to take care that path (i) is faster than path (ii). To be on the safe side in achieving this, we added two extra LUTs to the reset path (mux output $\rightarrow$ LUT1 $\rightarrow$ LUT2 $\rightarrow$

Fig. 12. Measurement results for a Muller C-element with high-threshold (single) and with Schmitt-trigger (schmitt) output (color online).

reset signals) while the path from the mux output to its select input is direct. By rigorously constraining the mux output signal (*MAXDELAY* constraint of only 400 ps), the delay of the direct path is definitely smaller than the delay including the two LUTs. To keep the delay of the reset path nevertheless reasonably small, we have constrained the delay of the two LUT outputs to 1 ns with *MAXDELAY*.

Further constraints are necessary to ensure the functionality of the late transition detection circuits. The delay on the output signals of the unit under test (UUT) flip flops, the delay on the reference and detection flip flops as well as the delay in the synchronizer stages was rigorously constrained, again using a *MAXDELAY* statement. These delays are in the order of 500 ps. To prevent the timing analyzer from erroneously checking the path between the UUT and the detection flip flop, a timing ignore constraint (*TIG*) was used. This is necessary, as the detection clock is shifted in the measurement procedure and the correct phase alignment is satisfied by the calibration run at the beginning of each measurement. Therefore, checking timing constraints based on the clocks on these paths leads to false results. The delay on these signals is, however, not arbitrarily large as it is constrained by *MAXDELAY* statements as already mentioned.

Additionally the UUT, reference and detector flip flops are manually placed to ensure that the same target is measured using the same detectors when the design is recompiled. Using these location (*LOC*) constraints, the designer can select the target to be measured. For the Virtex-4 FPGA, it was additionally necessary to pin the UUT and reference clock buffers manually as the clock generation is quite involved in this FPGA, technology. When using more modern FPGAs, such as Kintex-7, e.g., the locking of the clock buffers is no longer necessary.¹⁷

Listing 1 shows the relevant section of the design constraint file used in the case study circuit.

```

1  INST "ltd_d_ff_inst/uut/ff1"
   LOC = SLICE_X44Y3 | BEL = FFY ;
2  INST "ltd_d_ff_inst/uut/ff2"
   LOC = SLICE_X44Y1 | BEL = FFY ;
3  INST "ltd_d_ff_inst/ltd_inst/ref_inst/q"
   LOC = SLICE_X44Y4 | BEL = FFX ;
4  INST "ltd_d_ff_inst/ltd_inst/det_inst/q"
   LOC = SLICE_X44Y2 | BEL = FFY ;
5  INST "ltd_d_ff_inst/delay_line_inst/clk_uut_bufg_inst"
   LOC = BUFGCTRL_X0Y6 ;
6  INST "ltd_d_ff_inst/delay_line_inst/clk_ref_bufg_inst"
   LOC = BUFGCTRL_X0Y7 ;
7  INST "ltd_d_ff_inst/delay_line_inst/clk_uut_int1"
   LOC = SLICE_X23Y55 | BEL = G ;
8  INST "ltd_d_ff_inst/delay_line_inst/clk_ref_buf_inst"
   LOC = SLICE_X23Y55 | BEL = F ;
9
10 INST "top_inst/ltd_d_ff_inst/ltd_inst/det_inst/q" TNM = "DetectorFF";
11 INST "top_inst/ltd_d_ff_inst/uut/q" TNM = "UUTFF";
12 TIMESPEC TS_DetectorFF = FROM "UUTFF" TO "DetectorFF" TIG ;
13
14 NET "ltd_d_ff_inst/uut/q1" MAXDELAY = 515ps;
15 NET "ltd_d_ff_inst/uut/q2n" MAXDELAY = 515ps;
16 NET "ltd_d_ff_inst/q" MAXDELAY = 400ps;
17 NET "ltd_d_ff_inst/uut/c11" MAXDELAY = 1000ps;
18 NET "ltd_d_ff_inst/uut/c12" MAXDELAY = 1000ps;
19 NET "ltd_d_ff_inst/ltd_inst/ref_inst/q" MAXDELAY = 400ps;
20 NET "ltd_d_ff_inst/ltd_inst/ref_sync1_inst/sync<1>" MAXDELAY = 400ps;
21 NET "ltd_d_ff_inst/ltd_inst/ref_sync1_inst/sync<2>" MAXDELAY = 550ps;
22 NET "ltd_d_ff_inst/ltd_inst/ref_sync1_inst/sync<3>" MAXDELAY = 550ps;
23 NET "ltd_d_ff_inst/ltd_inst/det_inst/q" MAXDELAY = 400ps;
24 NET "ltd_d_ff_inst/ltd_inst/det_sync1_inst/sync<1>" MAXDELAY = 400ps;
25 NET "ltd_d_ff_inst/ltd_inst/det_sync1_inst/sync<2>" MAXDELAY = 550ps;
26 NET "ltd_d_ff_inst/ltd_inst/det_sync1_inst/sync<3>" MAXDELAY = 400ps;
27 NET "ltd_d_ff_inst/ltd_inst/det_sync2_inst/sync<1>" MAXDELAY = 400ps;
28 NET "ltd_d_ff_inst/ltd_inst/det_sync2_inst/sync<2>" MAXDELAY = 550ps;
29 NET "ltd_d_ff_inst/ltd_inst/det_sync2_inst/sync<3>" MAXDELAY = 550ps;

```

Listing 1. Relevant part of the UCF file.

6. Proposed Implementation Strategy

Our experimental results give evidence that the proposed methods for containment of intermediate voltages indeed works as intended. Although we have tested it on Altera and Xilinx FPGAs, the huge variety of different types and vendors does not allow a complete coverage. However, we propose the following implementation strategy that is generally applicable:

- (i) Design your circuit and identify those locations where metastability cannot be avoided by appropriate design provisions, usually at interfaces to other timing or handshake domains.
- (ii) Add the proposed components (threshold filter or Schmitt-trigger) as appropriate. Do not introduce a fork between the protected and protecting elements.
- (iii) Add the required constraints and run compilation of the circuit down to place and route.
- (iv) Lock the location of the protected components together with their protection gates with placement constraints, remove the rest of the circuit.
- (v) Add the validation circuit to the design and check whether the protection works properly. If you assumed the device to have a different filtering threshold than measured, add the inversions in case of a single-sided threshold filter or invert the selection signal of the mux and adapt the reset signals in case of a Schmitt-trigger. Afterward, re-run the validation measurement.
- (vi) Re-establish the original circuit while keeping critical components locked.

It is important to note that having tested a circuit prone to metastability, e.g., mutual exclusion with asynchronous requests for billions of cases without an observable failure does not allow for the extrapolated conclusion that metastability must have been successfully contained. Remember that metastable upsets are very rare events and could require months of measurement for a single occurrence depending on the resolution time. In our verification measurements, rather than rely on statistics, we forcefully drive the protected element into metastability allowing us to directly confirm its occurrence and containment by observation of its effects.

An FPGA implementation of an asynchronous interface is proposed in Ref. 18. The strategy proposed for critical timing paths is to use try and error for the routing and then compose a physical hard macro to fix the timing. Our approach is more systematic in that we lock the critical circuits by applying LOC constraints and verify their functionality with measurement circuits. Only if the verification fails, we start another iteration with different routing and verification.

7. Conclusion

We have motivated that in view of the significant PVT parameter variations experienced with recent technologies, value-safe designs with their flexible timing are

an attractive choice. Their prototyping (or small volume deployment) in FPGAs, however, suffers from the strong dedication of FPGA architectures to the synchronous design paradigm. While it is fundamentally impossible to completely avoid metastability of storage elements in the general case, its manifestations can be different. In the context of value-safe designs, the conversion from intermediate voltage level to a late transition is crucial for reliable handling of metastability. High-threshold buffer, low-threshold buffer, as well as Schmitt-trigger element are known to properly perform such a conversion, but so far no reliable FPGA implementation has been available.

We have thoroughly investigated how intermediate voltages are propagated in an FPGA and argued that FPGAs inherently present either high- or low-threshold behavior. We have validated our claim by theoretical models as well as comprehensive measurements. Furthermore, we have illustrated how to attain the other type of threshold behavior by appropriate inversion of inputs and outputs. On top of these single-sided threshold functions we have finally elaborated a complete and consistent approach for deploying a Schmitt-trigger in an FPGA and shown, by means of experimental evaluation (for both Xilinx and Altera platforms), that it indeed allows well-controlled metastability handling. We have taken two measures to ensure that our approach provides a safe solution even with the routing uncertainties of FPGAs: First, along with the proposed circuit diagram of the Schmitt-trigger we also gave the associated constraints to guide the synthesis toward a suitable result. Second, we proposed a procedure for an “*in situ* reliability assessment” of the specific Schmitt-trigger under consideration which already includes the relevant routing. Overall this provides a very safe solution, avoiding the compromises often implied by existing approaches. Although we have taken much care in elaborating reliable solutions, and although we have validated them on different FPGA platforms, our circuits must be considered somewhat “experimental” as long as they cannot be backed up by the relevant specification data (worst case of thresholds and output voltages over PVT range, ...) which are, unfortunately, not publicly available. As such we can definitely recommend our approach for experimental studies and prototyping but not for product development, especially not in the safety-critical field.

Our future work will be directed toward applying and extending this concept for building an optimized library of fundamental function blocks for the value-safe design approach.

References

1. L. R. Marino, General theory of metastable operation, *IEEE Trans. Comput.* C-30 (1981) 107–115.
2. J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design — A Systems Perspective* (Kluwer Academic Publishers, Boston, London, 2001).

3. T. Kacprzak, Analysis of oscillatory metastable operation of an RS flip-flop, *IEEE J. Solid-State Circuits* **23** (1988) 260–266.
4. E. Brunvand, N. Michell and K. Smith, A comparison of self-timed design using FPGA, CMOS, and GaAs technologies, in *IEEE Int. Conf. Computer Design: VLSI in Computers and Processors (ICCD)*, October 1992 (IEEE Comput. Soc. Press, Cambridge, MA, 1992), pp. 76–80.
5. D. L. Oliveira, S. S. Sato, O. Saotome and R. T. de Carvalho, Hazard-free implementation of the extended burst-mode asynchronous controllers in look-up table based FPGA, *4th Southern Conf. Programmable Logic*, 2008, San Carlos de Bariloche, 2008, pp. 143–148.
6. C. Pham-Quoc and A.-V. Dinh-Duc, New approaches to design asynchronous circuits on FPGAs, *Int. Conf. Advanced Technologies for Communications (ATC)*, 2009, Hai Phong, 2009, pp. 63–67.
7. C. Pham-Quoc and A.-V. Dinh-Duc, Hazard-free Muller gates for implementing asynchronous circuits on Xilinx FPGA, *Fifth IEEE Int. Symp. Electronic Design, Test and Application (DELTA)*, 2010, Ho Chi Minh City, 2010, pp. 289–292.
8. D. L. Oliveira, L. Faria and E. Lussari, FPGA implementation of robust asynchronous wrappers for globally-asynchronous systems (GALS), *VIII Southern Conf. Programmable Logic (SPL)*, 2012, Bento Goncalves, 2012, pp. 1–6.
9. X. Wang, T. Ahonen and J. Nurmi, Prototyping a globally asynchronous locally synchronous network-on-chip on a conventional FPGA device using synchronous design tools, *Int. Conf. Field Programmable Logic and Applications (FPL)*, 2006, Madrid, 2006, pp. 1–6.
10. M. M. Kim and P. Beckett, Design techniques for NCL-based asynchronous circuits on commercial FPGA, *Euromicro Conf. Digital System Design (DSD)*, August 2014, IEEE, Verona, 2014, pp. 451–458.
11. E. Amini, M. Najibi, Z. Jeddi and H. Pedram, FPGA implementation of gated clock based globally asynchronous locally synchronous wrapper circuits, *Int. Symp. Signals, Circuits and Systems (ISSCS)*, 2007, Iasi, 2007, pp. 1–4.
12. C. L. Seitz, System timing, *Introduction to VLSI Systems*, eds. C. Mead and L. Conway (Addison-Wesley Publishing Company, Inc., 1980), pp. 218–262.
13. T. Polzer and A. Steininger, SET propagation in micropipelines, *23rd Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2013, Karlsruhe, 2013, pp. 126–133.
14. D. J. Kinniment, *Synchronization and Arbitration in Digital Systems* (Wiley, 2007).
15. T. Polzer and A. Steininger, An approach for efficient metastability characterization of FPGAs through the designer, *19th IEEE Int. Symp. Asynchronous Circuits and Systems (ASYNC)*, 2013, Santa Monica, 2013, pp. 174–182.
16. T. Polzer and A. Steininger, Metastability characterization for Muller C-elements, *23rd Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2013, Karlsruhe, 2013, pp. 164–171.
17. T. Polzer and A. Steininger, Measuring the distribution of metastable upsets over time, *Euromicro Conf. Digital System Design (DSD)*, 2015, Funchal, 2015 pp. 189–196.
18. J. Pontes, R. Soares, E. Carvalho, F. Moraes and N. Calazans, SCAFFI: An intrachip FPGA asynchronous interface based on hard macros, *25th Int. Conf. Computer Design (ICCD)*, 2007, Lake Tahoe, CA, 2007, pp. 541–546.