Glitch-Stopping Circuits: Hardware Secure Masking without Registers

Zhenda Zhang[1], Svetla Nikova[1] and Ventzislav Nikov[2]

[1] COSIC, ESAT, KU Leuven, Belgium,
firstname.lastname@esat.kuleuven.be
[2] NXP Semiconductors, Belgium,
venci.nikov@gmail.com

Abstract. Masking is one of the most popular countermeasures to protect implementations against power and electromagnetic side channel attacks, because it offers provable security. Masking has been shown secure against d-threshold probing adversaries by Ishai et al. at CRYPTO'03, but this adversary's model doesn't consider any physical hardware defaults and thus such masking schemes were shown to be still vulnerable when implemented as hardware circuits. To addressed these limitations glitch-extended probing adversaries and correspondingly glitch-immune masking schemes have been introduced. This paper introduces glitch-stopping circuits which, when instantiated with registers, coincide with circuits protected via glitch-immune masking. Then we show that one can instantiate glitch-stopping circuits without registers by using clocked logic gates or latches. This is illustrated for both ASIC and FPGA, offering a promising alternative to conventional register-based masked implementations. Compared to the traditional register-based approach, these register-free solutions can reduce the latency to a single cycle and achieve a lower area cost. We prove and experimentally confirm that the proposed solution is as secure as the register-based one. In summary, this paper proposes a novel method to address the latency of register-based hardware masking without jeopardising their security. This method not only reduces the latency down to one clock cycle, but also improves the area cost of the implementations.

Keywords: AES · Hardware Secure Masking · Glitch-Stopping Circuits

1 Introduction

Every cryptographic implementation leaks certain information about the sensitive variables used for the computations through the so-called side-channels. These leakages can be of different nature: power consumption, electromagnetic, timing, micro-architectural, etc. In this paper we focus on leakages based on power and electromagnetic side-channels. The threat model is captured by the corresponding side-channel attacks (SCA) which are exploiting those power leakages by passively measuring the power consumption of millions of executions. Then using the collected power traces the attacker apples statistical methods, like differential power analysis [KJJ99], to recover the sensitive variables (e.g., the AES key) using divide and conquer revealing the key byte per byte. SCA can be very harmful if there is no proper countermeasure implemented. The known countermeasures fall into the following two categories, which however, can be jointly implemented:

- *Leakage hiding* tries to either remove the variations caused by the computations, or to create artificial noise in the circuit. The means to achieve hiding are shielding, power balancing, noise makers, dummy operations, dual-rail technology.

- *Noise amplification* leverages the existing noise in the given side-channels to make the measurements harder. Shuffling and masking are the two methods used to achieve noise amplification.

Through the years masking earned the reputation of being the most effective practical countermeasure. It is especially relevant because of its provable security. To prove the security of a cryptographic implementation, first the side-channel adversary's model has to be defined. The simplest model is the d-threshold probing model in which the adversary is only allowed to probe the values of d wires within the circuit [ISW03]. However, the d-probing model considers only ideal circuits and excludes any physical hardware defaults. To address this limitation, the glitch-extended probing model was first introduced by Reparaz et al. [RBN+15] at CRPYTO'15, followed by the formal definition of the robust probing model by Faust et al. [FGP+18] at CHES'18. The glitch-extended d-threshold probing model again allows the adversary to probe d wires within the circuit but in addition also all the input values of the probed wires are given to the adversary. In other words, the adversary can learn more than d values. We stress that although glitches are undesired (for security and power reasons) they are ubiquitous part of every CMOS technology. Masking with $d+1$ shares provides a successful defense against (glitch-extended) d-probing adversary if no information can be revealed for the sensitive variable for any choice of the d probes.

Early masked schemes didn't consider physical hardware defaults and Mangard et al. [MPO05] at CHES'05 showed that such schemes leak information due to glitches. It became clear that glitches in hardware designs can cause unintended re-combinations of masked values due to varying delays in the circuit, so the question was how to build resistant circuits. Mangard et al. also proposed two strategies which Lammers et al. [LMM23] named as *glitch-free circuits* and *glitch-immune circuits*.

Two main physical defaults have been identified: glitches and transitions. When a circuit stores a value in a memory element, the power consumption depends on the old (i.e., already stored value) and the new value, which overwrites the memory state. This overwriting effect is called a transition. The propagation delay across different data paths within the combinational logic is not uniform. A single gate may receive input signals that arrive at different times. As a result, the gate's output may undergo temporary changes. This process continues until the gate's input signals stabilize, after which the gate will produce a stable output. From a logical circuit perspective, these stable signals progressively propagate through the circuit from the input registers to the output registers throughout a cycle. Such short-lived effects of the signal are known as glitches.

A circuit is considered glitch-free if it experiences no internal glitches. One branch of research focused on investigating ways to eliminate glitches in circuits with the goal of achieving power equalization and thus leakage hiding. Early attempts to build glitch-free circuits have focused on dual-rail logic with pre-charging (DRP) and its extension WDDL [TV04]. All DRP circuits are usually asynchronous since, otherwise, they introduce time delay due to the pre-charging phase. Early-propagation in DRP logic, where a DRP logic's output stabilizes without all input signals being determined, causes side-channel leakages and such have been illustrated for both DRP and WDDL. Further extensions like DRPnoEE and AWDDL considered monotonic Boolean gates, but still standalone (without additional masking) those solutions were shown to be vulnerable.

In parallel to glitch-freedom, another branch of research focused on the glitch-immune circuits, namely masking (i.e., noise amplification) that maintains their security in the presence of glitches. The first glitch-immune circuit is proposed in 2006 [NRR06], where the Threshold Implementations (TI) methodology is introduced. TI outlines two principal guidelines for constructing glitch-immune masking:

- To render a circuit resistant to SCA in the presence of glitches, it is necessary to incorporate additional registers. The primary role of these additional registers is to

prevent the propagation of glitches between sub-circuits.

- Each masked logic block between two registers (starting and ending points) must adhere to the *non-completeness* requirement as defined by TI. This ensures that each such sub-circuit is glitch-immune.

Since then, many glitch-immune masking schemes have been proposed on the basis of, or expanding upon, the TI design principles, for example HOTI, CMS, DOM, HPC1, HPC2, and HPC3 [BGN+14, RBN+15, GMK16, CGLS21, KM22]. To reduce the latency overhead, GLM [GIB18] omitted share compression after each nonlinear operation but at the cost of increased intermediate share count for higher-degree S-boxes. More recently, LLTI, a low-latency masking technique based on TI, offers comparable area requirements to GLM while eliminating online randomness [AZN21]. Note that glitch-immunity refers to a circuit which achieves glitch-extended probing security. We are now ready to define glitch-stopping circuits as follows.

Definition 1. A *glitch-stopping circuit* (GSC) refers to a synchronous digital circuit that (1) incorporates glitch-immune masking schemes and (2) stops the propagation of glitches between consecutive blocks of combinational logic.

Meanwhile since DRP-like solutions standalone still exhibited data-dependent early signal propagation leading to potential leakage [MS16], research shifted to strengthen such solutions by applying glitch-immune masking in addition. The authors of [MS16] proposed the integration of WDDL and first-order TI masking. More recently LMDPL [SBHM20], self-synchronized masking (SESYM) [NGPM22] and self-timed masking [SBB+22] were proposed. Those solutions have been applied as asynchronous logic for part of a cipher implementation, while globally the cipher is a synchronous design. All these three recent solutions highlight the importance of data synchronization layers even in glitch-free designs, especially synchronizing in long combinational data paths inside such asynchronous circuits. Considering these works, which integrate DRP logic and masking, we redefine glitch-free circuits as follows.

Definition 2. A *glitch-free circuit* (GFC) refers to an asynchronous digital circuit that (1) incorporates glitch-immune masking schemes and (2) uses DRP logic and its extensions to eliminate any internal glitches.

A notable aspect of our investigation is the measurement of latency. Latency can be measured in two different ways: cycle-wise or time-wise. The choice between the ways to measure depends on the specific use cases and constraints. In scenarios where there are no limitations on clock frequency, time-wise measurement, which is the delay from the input of a circuit to its output, is preferable. However, in instances where the clock frequency is constrained, and the implementation block cannot be clocked faster than the rest of the integrated circuit (IC), cycle-wise measurement becomes more relevant. Recognizing the dual meaning of the term, we will provide latency measurements in both cycle-wise and time-wise formats, offering a comprehensive view of the circuit's performance under different conditions.

This work: Figure 1 illustrates the categories of hardware secure masking which includes GSCs as defined in Definition 1 and GFCs as defined in Definition 2. When glitch propagation is prevented by registers, we denote such circuit by `GSC_R`. Such circuits are equivalent to the conventional "glitch-immune circuits". We recall that during the last 10 years glitch-immune masking (like TI, DOM, HPC, etc.) became the only recognised way to do secure masking in hardware. Since additional registers are used for security purposes such masking schemes have the drawback of increased cost and latency. However, `GSC` can be instantiated not only with registers, and this paper focuses on building such instances

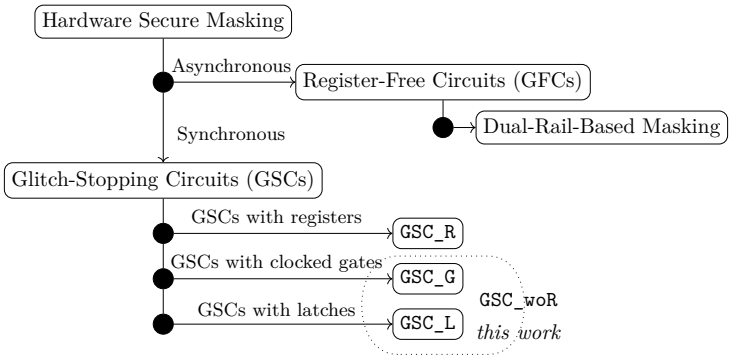


Figure 1: Categories of Hardware Secure Masking

without registers. Thus, our motivation is to keep the same level of security while we aim to reduce the cost in area and to improve the latency of secure HW implementations. In particular, we propose the usage of *clocked logic gates* or *latches* to build glitch-stopping circuits (`GSC_G` and `GSC_L`, respectively). We refer to those new solutions as `GSC_woR` and we show that they can stop glitch propagation between logic blocks as effectively as `GSC_R`, thus maintaining the security order of digital circuits against SCAs. The design and implementations for `GSC_woR` are feasible for digital logic circuits in both application-specific integrated circuits (ASIC) and field-programmable gate arrays (FPGA) environments. The key contributions of this paper are as follows:

- We introduce a novel approach to build glitch-stopping circuits that do not rely on registers `GSC_woR` while keeping the synchronous nature of the design. This method employs clocked logic gates or latches, which can stop the propagation of glitches as effectively as registers do.

- The novel `GSC_woR` require minimal changes in the clock configuration. Namely, additional clocks are added, that play a crucial role in ensuring the correct timing and synchronization of signals in the circuits. We provide complete procedure of how to make the required changes in the circuit, for both ASIC and FPGA, in a way that the security claims of the original circuit are preserved. This allows us to prove that `GSC_woR` maintain the same security as `GSC_R` under the glitch-extended probing model.

- A comprehensive comparison is provided between the proposed `GSC_woR`, the conventional `GSC_R` and state-of-the-art glitch-free circuits. This analysis covers area efficiency and latency in timing and in cycles. We show that `GSC_woR` can reduce the latency to a single cycle, reduce area and improve the overall time-wise latency.

- We performed a leakage evaluation of the proposed `GSC_woR` instantiated with four Xilinx FPGA primitives. This physical security evaluation provides empirical evidence that two out of the four primitives effectively implement glitch-stopping gates and thus achieve the first-order security.

2 Preliminaries

In this section we introduce all the notations and fundamental concepts of sequential logic circuits and their role in developing secure hardware implementations by either preventing glitches or their propagation.

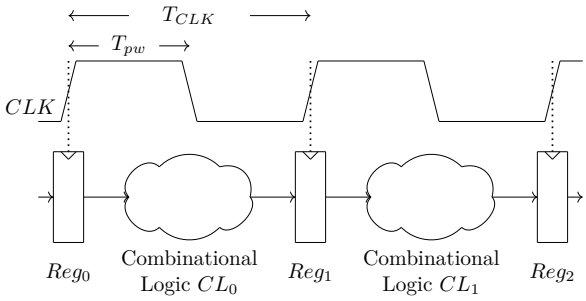


Figure 2: A synchronized circuit and its clock signal [WH10]



(a) A D-latch

(b) D-latch timing diagram

Figure 3: A D-latch that is level-sensitive to its clock signal [Wik24]

2.1 Sequential Logical Circuits

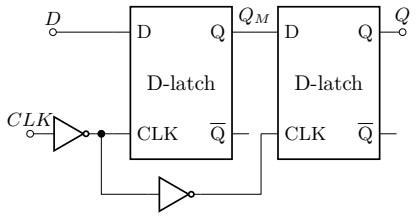
In digital design, *sequential logic* circuits provide outputs based on the current input values and the preceding input states. These preceding input states are stored in memory elements, usually referred to as *registers*. Conceptually, a register is characterized by a single input and output, governed by a clock signal. Combinational logic circuits connect the stages of registers. *Combinational logic* circuits consist of *(Boolean) gates* connected by wires carrying the signals. Most common circuits are *synchronized*, meaning a mix of combinational and sequential logic where the latter is driven by a global *clock signal*.

An example of sequential circuits is *pipelining*, as depicted in Figure 2. Pipelining is a technique aimed at optimizing resource utilization, enhancing functional *throughput*, and ultimately accelerating the operation of data paths in digital circuits. In this example, the pipeline has two stages of combinational logic, separated by registers. Every cycle, a new input is stored in Reg_0, after two cycles the output becomes ready at Reg_2. Thus, the latency for each input is two cycles. However, the circuit does not need to wait for the output to be ready before processing the next input. The throughput, quantifying the number of operations a circuit can process per cycle, is 1 bit/cycle in this example.

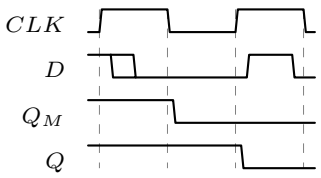
2.1.1 Registers

A digital signal has two states or voltage *levels*: the *low* state is denoted as 0, and the *high* state as 1. The clock signal is a special digital signal used to synchronize all gates in the circuit. The *rising edge*, or *positive edge*, of the clock signal is the transition from 0 to 1. Similarly, the *falling edge*, or *negative edge* of the clock signal is the transition from 1 to 0. Registers are commonly instantiated as flip-flops and sometimes as latches.

Latches: A *latch* operates as a level-sensitive memory element, while in contrast, a *flip-flop* operates as an edge-sensitive memory element. As depicted in Figure 3, when the clock signal controlling a D-latch is 0, the input is ignored, while the output is locked and maintained to the last input state. This state of the latch is referred to as the *hold*

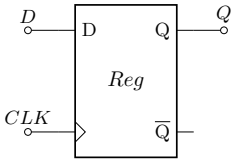


(a) A primary-secondary D-flip-flop (Register)

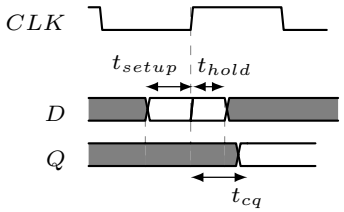


(b) D-flip-flop timing diagram with its internal value

Figure 4: A primary-secondary D-flip-flop that is triggered on the rising edge of the clock [Wik24, RCN03]



(a) A D-flip-flop (register)



(b) Timing characteristics

Figure 5: Definitions of set-up time, hold time, and propagation delay of a D-flip-flop [Wik24, RCN03]

mode since it effectively stores the data. When the clock signal becomes 1, the input propagates and passes continuously to the output. This is the *transparent* mode, which ensures real-time tracking of the input signal. The hold mode makes latches an essential building block for maintaining data in digital logic circuits, but any glitch on the input will propagate to the output when the clock signal is high, hence latches should be in transparent mode only when the input signal is stable.

Flip-Flops: D-flip-flops (FF) operate by sampling the input data typically at the rising edge of the clock. They can also be triggered on the falling edge, though such implementations are less common. A flip-flop is normally built based on the *primary-secondary*, also known as *master-slave*, configuration of two cascaded latches, as shown in Figure 4.

When CLK is 1 and before its falling edge, the primary D-latch is in hold mode and the secondary D-latch operates in transparent mode. The primary latch ignores any new input and holds its last input data constant at its output Q_M. The secondary latch, now transparent, receives the data Q_M and transfers it to its output Q, allowing the data to proceed to the combinational logic following the FF. Conversely, when CLK is 0 and before its rising edge, the primary latch accepts data from the D input and transfers it to its output Q_M. Meanwhile, the secondary latch is in hold mode, maintaining a stable output of its previously stored value. Consequently, the FF changes its output Q when the rising edge of the clock signal occurs. Unlike latches, either one of the FF's two latches always blocks any glitch inputs, preventing them from propagating to the FF's output.

2.1.2 Clock Signal Characteristics and Timing Constraints in Synchronous Circuits

The clock controls the updating of the sequential memory elements within an IC or a chip. As illustrated in Figure 2, a synchronous circuit is synchronized by a clock signal that is connected to all registers. The *clock period* (T_{CLK}) is the period between two rising clock edges. In other words, this is the sum of the rise time, high time, fall time, and low time

of the clock signal, as depicted on Figure 2. Within this clock period, or during one *cycle*, stable signal propagation through the circuits must be ensured, setting a minimum limit for T_{CLK}. The *frequency* of a synchronous circuit is defined as $1/T_{CLK}$, and consequently, it possesses a maximum limit. It is important to note that in complex system-on-chips (SoCs), different clock frequencies may coexist, being distributed to various parts of the integrated circuit.

Duty Cycle: *Pulse width* (T_{pw}) of a signal is the duration from its rising edge to its falling edge. The ratio of the positive pulse width T_{pw} and the total clock period T_{CLK} is defined as the *duty cycle* of the clock. Expressed as a percentage, the duty cycle is given by $D = T_{pw}/T_{CLK} \times 100\%$. Clock signals in modern circuit designs typically have a 50% duty cycle, although this is not always the case. Advanced electronic design automation tools (EDA) facilitate modifications of the duty cycle.

Timing Considerations of a flip-flop: The timing metrics of a flip-flop are critical for correctly registering an input to its output, as illustrated in Figure 5. The *setup time* is the interval during which the input data D must remain stable before the clock's rising edge. This period typically corresponds to the time required for the input signal to propagate through the primary latch. The *hold time* represents the duration that D cannot change for the data to transition to the secondary latch and the FF output. Propagation delay t_{cq} is the time the signal takes for the input change to be reflected at the output after the clock edge.

If the setup or hold time requirements are not met, the output of the FF is unpredictable and may even be unstable: the output may operate normally, take an invalid level, or oscillate. Therefore, setup and hold times define the uncertainty time windows in which the FF output is not deterministic. That is why in synchronous digital design clock periods are chosen such that no meta-stability of the FFs ever happens. In synchronous circuit design, the rising edge of the clock signal indicates that all signals have reached a valid and stable state. This is guaranteed by setting the clock period, during the design time, to the worst-case delay between two FFs. Usually, the timing requirements are controlled and ensured via the EDA tools.

The global clock signal has a tree structure from a single source and thus it is glitch-free. Note that if there are glitches in the clock signal then flip-flops might become meta-stable and hence the glitches on the data signals might not be stopped by a flip-flop. We will further consider the depicted in Figure 2 circuit to be a circuit without a side-channel attack resistance.

2.2 SCA Resistant Circuits

In this section, we consider the leakage model that captures the effect of glitches and we discuss the state-of-the-art for SCA resistant circuits.

Let's recall that a *glitch-extended probe* on a wire z models the impact of glitches by capturing all stable signals that contribute to the wire z. In other words, such a probe collects all flip-flop values leading to the wire z. Similarly, the *transition-extended probe* on a register w models the impact of transitions by recording two consecutive signals stored in w. In the rest of the paper, the main concept we will use is the glitch-extended probing model. In this model, it is assumed that an attacker can instantiate a glitch-extended probe from a wire z in the circuit and trace the wire values backwards to all registers which store signals contributing to z. However, tracing back beyond the register (into the preceding circuit) is not feasible as glitches are stopped by the register. Since both the glitch-free and the glitch-immune circuits include glitch-immune masking the glitch-extended probing model offers a framework to establish security proofs for both of them.

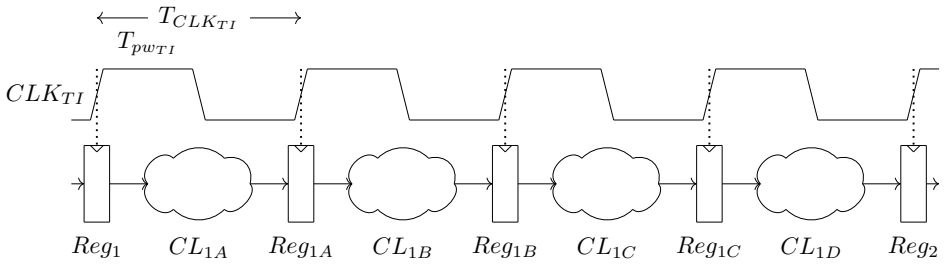


Figure 6: Glitch-stopping circuit with four-stages TI

2.2.1 Glitch-Immune Circuits

The authors of TI [NRR06] were the first to provide first-order protection against side-channel attacks in the presence of glitches. The TI methodology involves segmenting the combinational logic into smaller sub-circuits, as illustrated in Figure 6. Moreover, TI mandates non-completeness for each masked sub-circuit or gadget and requires that the masking must be both correct and uniform to guarantee first-order composability. Subsequently, the notion of *composability* was introduced [BBD+16]. Namely, a secure gadget is composable if and only if all possible combinations of such gadgets maintain probing security under the same security model. Various notions of composability, including NI, SNI, and PINI, have been introduced [BBD+16, CS20]. Secure masking schemes in hardware have also evolved to incorporate composability, resulting in the development of schemes such as HPC1, HPC2, and HPC3. We will not explore the specifics of these well-established and understood methods.

Figure 6 illustrates an example of a glitch-stopping circuit. For illustrative purposes, let's assume that the second combinatorial logic circuit CL_1 from Figure 2 is made side-channel protected by dividing it into four stages CL_{1A-1D} and applying first-order TI masking. The inserted additional registers expand the one-stage unprotected logic to four stages of glitch-stopping circuits, thus increasing from one cycle latency with T_{CLK} to four cycles with $T_{CLK_{TI}}$.

2.2.2 Glitch-Free Circuits

Dual-rail logic encodes each value {1} or {0} to a tuple {1,0} or {0,1} respectively, while {0,0} is known as empty value and {1,1} being not allowed. It is not allowed to have a transition from one valid encoding to another unless {0,0} is inserted in between. This return-to-zero is achieved by forcing a reset to the circuit i.e., pre-charging. The DRP logic e.g., WDDL [TV04], operates in two phases: a pre-charge phase setting all wires to an empty value, followed by an evaluation phase processing valid inputs to yield an output. WDDL is designed for power equalization and to reduce SCA leakages. However, WDDL's limitations in leaking secret information by unmatched wire delays and high routing uncertainty between rails led to its gradual replacement by more flexible glitch-immune masking techniques.

It has been noted that countermeasures relying solely on asynchronous logic are inadequate without masking, which led to the exploration of a hybrid approach. The research in [MS16] aimed to determine whether integrating a glitch-free WDDL-based circuit with TI, but without using registers, could achieve a design secure against SCA. The authors pointed out that "although placing registers between the shared non-linear functions was initially introduced to avoid the propagation of glitches, it also synchronizes the start of their evaluation to be independent of the timing of the previous stage." In their asynchronous circuit, the authors employed registers to isolate shared non-linear functions. However, as [MS16] reveals, this approach results in a design significantly larger than its

synchronous counterpart and faces latency issues due to the interleaved pre-charge and evaluation phases, ultimately compromising some advantages of asynchronous design due to the synchronization constraints.

The primary goal of the most recent asynchronous glitch-free designs is to achieve low latency, which means to reduce the number of clock cycles required for computing the entire masked function. However, these designs remain globally synchronous while being locally asynchronous so the efficiency improvement is focused on the masking of S-boxes in ciphers. One such approach is the LMDPL technique [SBHM20], which combines look-up table based first-order masking on monotonic gates with DRP logic. While this method has been shown to ensure first-order security, its performance and area overhead in comparison to similar designs without LMDPL are not clearly established due to the lack of comparative data in the same CMOS technology. Additionally, Nagpal et al. [NGPM22] proposed the SESYM technique. This method utilizes WDDL along with the Muller C-elements for synchronization, ensuring security across different critical components like S-boxes without the need for a fully dual-rail data path. Another approach is called self-timed masking [SBB+22], which modifies monotonic Boolean functions i.e., dual-rail OR and AND gates, and aims at eliminating the early-propagation. Data synchronization in [SBB+22] is done in similar way as in [NGPM22] using the Muller C-element. To our best understanding, the main difference between [SBB+22] and [NGPM22] is the use of AWDDL instead of WDDL logic.

We stress here that the recently proposed GFCs, that incorporate glitch-immune masking with DRPs, have demonstrated a notable resilience against SCAs, even at one order higher than the masking order. This enhanced security is primarily attributed to the dual-rail power-equalization effect.

3 Case Study: Threshold Implementations on AES

In this section, we present two different implementations of the AES S-box with serialized architecture of `AES-128` encryption. Let t be the algebraic degree of the corresponding Boolean function. The first implementation uses first-order $t+2$ TI sharing with five shares of the S-box decomposed in two stages. The second one uses first-order $t+1$ TI sharing (with three shares) of the S-box decomposed in four stages. We will apply those 2 implementations in Section 4 and Section 5.

3.1 Serialized AES Encryption

The `AES-128` encryption is implemented using a serialized architecture with a one-byte-width data path, incorporating an AES S-box instance as described in either Section 3.2 or Section 3.3. Due to the pipelining of the S-box, proper scheduling of operations is essential. Data is stored in s shares, where s equals either 5 or 3. The architecture comprises two state arrays of size $s \times 16$ bytes each, one for data and the other for the key. Data and key bytes in s shares move unidirectionally from one functional unit to the next, with their results being stored in the respective state registers after each clock cycle. The functional units consist of linear layers and the S-box, with registers in the latter capturing intermediate results due to pipelining.

In the AES round control flow, the ShiftRows operation is executed concurrently with the S-Box output processing. The output of the S-Box is written to the state arrays in which it would be stored after performing the ShiftRows operation. In other words, each column is written back diagonally with rotational wrapping around the state matrix. The MixColumns operation is performed as soon as all column bytes are ready. The result of MixColumns is ready in the following cycle. The key addition is performed in the next

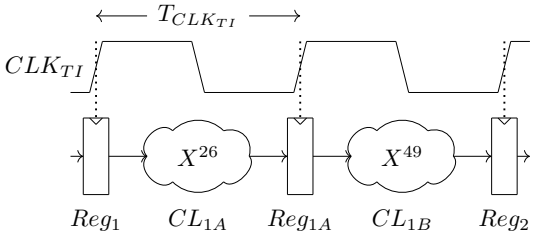


Figure 7: Representation of the two-stage masked AES S-box with five shares.

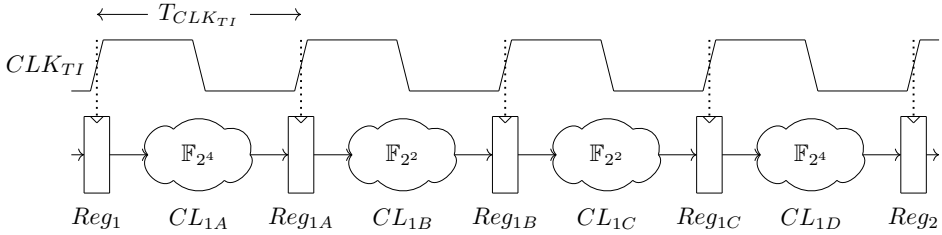


Figure 8: Representation of the four-stage masked AES S-box with three shares.

round iteration, while the final key addition is computed during the read-out operation. For the considered two pipelined designs an AES round fits into 20 cycles [BKN20].

3.2 Two-Stage AES S-box with Five Shares

The implementation of a two-stage AES S-box involves decomposing the inversion in $GF(2^8)$ of algebraic degree seven, into two cubic functions: namely $x^{-1} = x^{254} = (x^{26})^{49}$ [NNR19]. This decomposition facilitates a pipelined two-stage implementation with an interposed single glitch-stopping layer (register). First-order TI with $t+2$ shares [PAB+22] is applied to each of the cubic functions x^{26} and x^{49}. This strategy not only simplifies the design but also saves randomness cost, since the sharing is uniform no additional randomness is required. The architecture of the S-box is shown in Figure 7.

3.3 Four-Stage AES S-box with Three Shares

Canright [Can05] proposed an efficient tower field decomposition of the AES S-box in order to improve hardware costs. Since then this decomposition was widely used to create efficient hardware masking of AES. The tower-field decomposition of the S-box consists of the linear input/output isomorphism; and the nonlinear finite field multipliers, finite field scaling functions, and a multiplicative inversion. The tower-filed computes the inversion by starting from $GF(2^8)$ via $GF(2^4)$ to $GF(2^2)$ each time simplifying it until it becomes a linear operation. The implementation we will use is a modified version of the Design II by Askeland et al. [ADN+22] but with four stages instead. The shared linear stage from the input is merged into the first non-linear stage which results in one stage less than the Design II. The S-box uses a total of 36 random bits since we do not use the changing of the guards method. The architecture of the S-box is shown in Figure 8.

3.4 Timing Overhead of Glitch-Immune Masking Schemes

Revisiting the timing requirements, we compare the total delay of an unprotected circuit denoted as T_{CLK} to the time delay of a `GSC_R` protected circuit, represented by $T_{TI} =$ Number-of-stages $\times\, T_{CLK_{TI}}$. For the case of $t+2$ TI and a cubic function (Section 3.2) the timing overhead of `GSC_R` compared to unprotected cubic function is the time required

for the evaluation of $3 + 2 + 1 = 6$ XORs one after another. When applied to the shared AES S-box with two stages we obtain that the time delay compared to un-shared S-box is the time to pass through $2 \times 6 = 12$ XORs and 1 Register. Note, that when we talk about timing overhead in this section, we will refer to the number of gates but we will mean the time required for their consecutive evaluation.

Let us consider now masked Canright S-box and $t + 1$ TI on quadratic functions (Section 3.3). Taking into account that the overhead of finite field masked-multiplication is $1 + 1 = 2$ XORs, the overhead for the shared AES S-box (Canright with four stages) is $4 \times 2 = 8$ XORs and 3 Registers.

We also can consider masking with $d + 1$ shares (e.g., CMS, DOM) for quadratic functions. In this case the overhead of masked-multiplication is: $log_2(d + 1) \times (1 + 1)$ XORs and 1 Register. Hence, the overhead of a masked AES S-box (Canright with 5 stages) will become $log_2(d + 1) \times 4 \times 2$ XORs and 4 Registers. Similar calculations can be made for AES S-box implemented via Boyar-Peralta [BP11] representation which has the multiplicative depth of 4.

The general, the timing observation for `GSC_R` masked implementations is that the overhead comes from: a) the number of stages (and the required registers); b) the number of XORs used for re-sharing; c) the number of XORs used to compress the expanded number of sharings after multiplication.

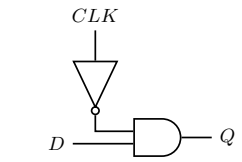
4 Glitch-Stopping Circuits in ASIC

In this section, we introduce glitch-stopping circuits without registers (`GSC_woR`) in contrast with the traditional approach with registers (`GSC_R`). We will show that glitch-stopping circuits can be built with different clocked combinational logic blocks, i.e. `MUX`-gate, `AND`-gate (`GSC_G`) and latch (`GSC_L`). This results in reduced latency and area. We will prove that the proposed `GSC_woR` provides security equivalent to `GSC_R` in the glitch-extended probing model. Different configurations of the clock signals can be used for `GSC_G` and `GSC_L`. We will use both high and low clock signal levels as well as rising and falling clock edges in our descriptions further. However, in cases when it is not advisable to use the falling clock edge then the described solutions can easily be adapted to only use the clock rising edge.

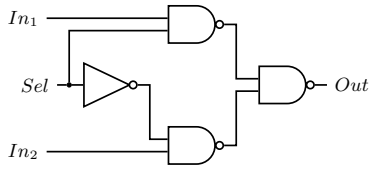
4.1 Clocked `AND`-Gate

Consider a device with `GSC_R` implementation as depicted in Figure 6. This implementation is either represented in hardware description language on register-transfer level (RTL) or gate-level description netlist. Now we preserve all the four sub-circuits $CL_{1\{A-D\}}$ and their critical paths. We replace each register inside the pipeline $Reg_{1\{A-C\}}$ with clocked logic `AND` gate, as depicted in Figure 9a. The `AND`-gate is placed between the data (in/out) signals and the inverted clock signal is used for control. Note that the term *clocked-gate* should not be confused with *clock gating*. Clock gating is a power-saving technique that disables clock signals in a synchronous logic module to reduce dynamic power dissipation.

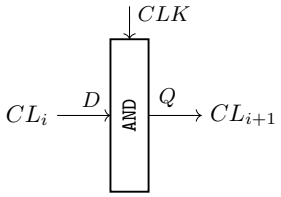
To ensure the same security as `GSC_R`, it is crucial to maintain the timing as shown in Figure 10, where $T_{CLK_{TI}}$ is included for reference (as originally given in Figure 6) to demonstrate the preserved critical path for $CL_{1\{A-D\}}$. Proper synchronization is achieved by introducing two additional clocks with the same period as the main clock $T_{CLK_{Reg}} = T_{CLK_B} = T_{CLK_A} = T_{CLK_C}$, yet with varying duty cycles. Specifically, D_{CLK_A} is set at 25%, D_{CLK_C} at 75%, and $D_{CLK_B} = D_{CLK_{Reg}}$ remains consistent with the global clock at 50%. To be able to refer to the sub-circuit evaluation in each of the four circuit stages we introduce the notion of "virtual cycle" as an analogue CLK_{TI}.



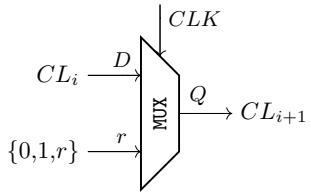
(a) A clocked AND-Gate



(b) A clocked MUX-Gate



(c) Integration of the AND-Gate into GSC



(d) Integration of the MUX-Gate into GSC

Figure 9: AND and MUX instances of the glitch-stopping circuits

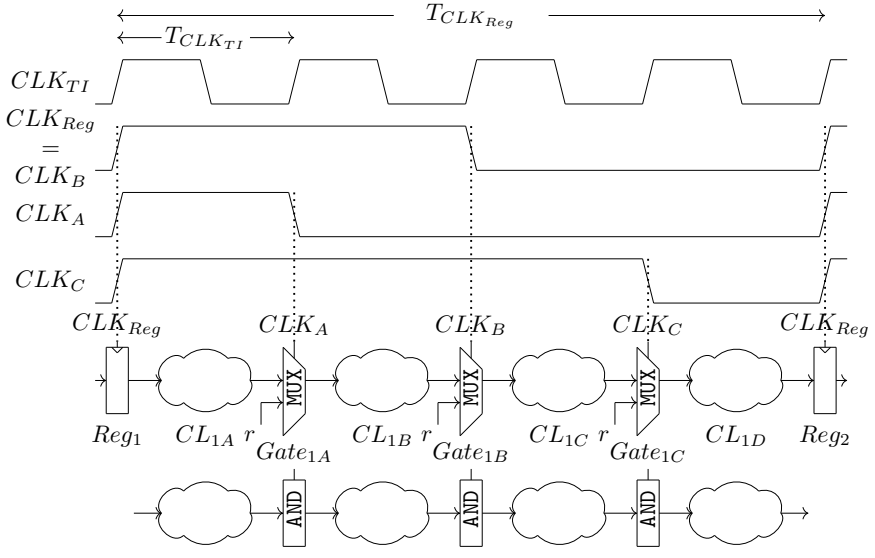


Figure 10: Clocked logic-gate-based glitch-stopping circuit

The three clock signals, CLK_A, CLK_B, CLK_C, manage signal flow through the stages $CL_{1\{A-D\}}$ in the `GSC_G`, ensuring only stable signals passing and stopping glitch propagation. In the first virtual cycle, all three clock signals are high, ending with CLK_A's falling edge. In other words, the positive pulse width $T_{pw_CLK_A}$ align with CLK_{TI}'s first clock period. Immediately the second virtual cycle starts with CLK_A low, while CLK_B and CLK_C remain high. It ends with CLK_B's falling edge, accordingly, $T_{pw_CLK_B} - T_{pw_CLK_A}$ matches with the second clock period of CLK_{TI}. The third virtual cycle has CLK_A and CLK_B low, CLK_C high, and finishes with CLK_C's falling edge. Thus $T_{pw_CLK_C} - T_{pw_CLK_B}$ aligns with CLK_{TI}'s third period. The final fourth cycle occurs with all clocks low, ending with their simultaneous rising edge, thus matching CLK_{TI}'s fourth clock period and corresponding to $T_{CLK_{Reg}} - T_{pw_CLK_C}$.

During the first virtual cycle in the clocked `AND`-gate `GSC_G` design, $Gate_A$ is closed, preventing signal propagation from CL_{1A} to CL_{1B}. $Gate_A$ output is stabilizing at zero. During this virtual cycle also $Gate_B$ and $Gate_C$ are closed, maintaining zero at their outputs. CL_{1A} is evaluated, while circuits CL_{1B-D} receive zero inputs. By design the virtual cycle finishes when CL_{1A}'s signals stabilize, and thus when $Gate_A$ opens at the beginning of the second virtual cycle the signals without glitches pass between CL_{1A} and CL_{1B}. In the second virtual cycle, $Gate_B$ and $Gate_C$ stay closed, sustaining their zero outputs hence $CL_{1\{C,D\}}$ remain unchanged. $Gate_A$ is open allowing CL_{1B}'s evaluation while CL_{1A} remains stable. Again by design, this virtual cycle ends when CL_{1B} stabilizes, leading to the opening of $Gate_B$ and the start of the third virtual cycle. In this third virtual cycle only $Gate_C$ remains closed, keeping its output zero. CL_{1C} undergoes evaluation while $CL_{1\{A,B\}}$ maintain their stable states. This virtual cycle ends with the stabilization of CL_{1C}'s signals, initiating the fourth and final virtual cycle in which CL_{1D} is evaluated. During this period, $CL_{1\{A-C\}}$ remain stable, and Reg_2 at the circuit's end halts further signal propagation. Thus, this strategy ensures stable output values at each stage circuit when the negative edge of the corresponding clock signal occurs.

4.2 Clocked Latch

As described in Section 2.1.1, registers, or flip-flops, consist of cascaded latches connected by the inverted clock signal. We show three ways to configure the clock signals for a glitch-stopping circuit with latches `GSC_L`. The first approach repeats the clock signals of the `GSC_G`. The second approach, depicted in Figure 11, differs from the first by introducing two inverted clocks CLK_P and CLK_N. The third, simpler method, requires only the CLK_N clock but both rising and falling edges of the clock are utilized to control the *Gates*. Notably, inverting the clock signal is not required when using the rising edge.

The `GSC_L`, while structurally similar to the `GCS_G`, diverges significantly in the way circuits $CL_{1\{A-D\}}$ are evaluated due to the inherent ability of the latch to retain its previous value. In the first virtual cycle, where CLK_A is high, the $Gate_A$ effectively halts any glitch progression between $CL_{1\{A,B\}}$. For the clocked `AND`-gate scenario, this results in $CL_{1\{B-D\}}$ having stable, zero-valued inputs. Conversely, in a latch-based setup, these circuits maintain stability by retaining their original data values.

4.3 Clocked `MUX`-Gate

The clocked `MUX`-gate solution is similar to the clocked `AND`-gate approach, with the primary distinction being the connection of the clock signal to the select input *sel* of the `MUX`, as shown in Figure 9b. This setup uses the same additional clocks as the `AND`-gate solution. The key difference lies in the use of a 2-to-1 `MUX` where, apart from the data input, the secondary input is variable. Setting this secondary input to a constant 0 repeats the `AND`-gate solution by zeroizing unprocessed circuits. Alternatively, a constant 1 input results in an identical operation with stable inputs 1. A more sophisticated option involves connecting

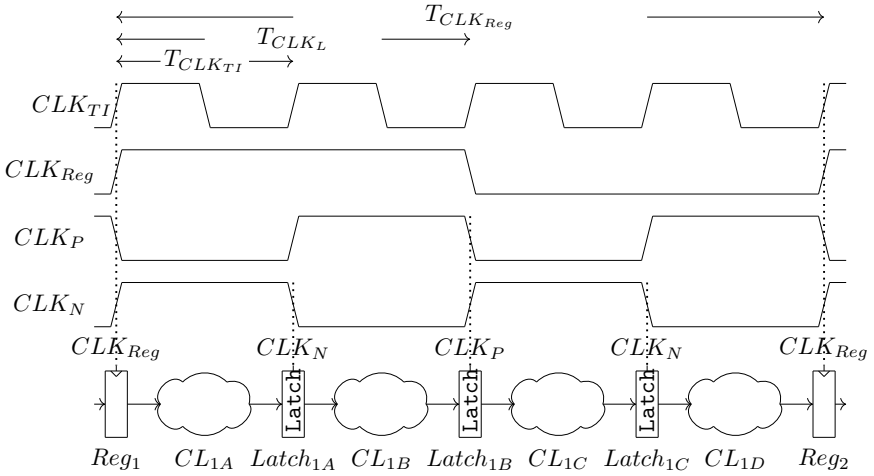


Figure 11: Clocked latch-based glitch-stopping circuit

the secondary input to a randomness source, thereby randomizing the un-evaluated circuits. This last choice adds noise and may result in a more robust solution.

4.4 Comparison of the Glitch-Stopping Circuits

In this section, we compare the four solutions: `GSC_R` given in Figure 6 and the 3 new `GSC_woR` given in Figures 10 and 11. We weigh up their glitch-stopping mechanism and their costs on timing and area.

For the new `GSC_woR` solutions we had to introduce additional clock signals. For example, in the clocked `AND`-gate construction of the four-stage masked circuit, two additional clock signals CLK_A and CLK_C are required. Instead of generating those clocks, a technique for detecting the clock glitches can also be used, as detailed in the recent overview by Askeland et al. [ANN23]. Namely, we will use in a different context the so-called parallel delay line circuit (PDL), as illustrated in Figure 12. The PDL derives from the main clock two lines of circuits, i.e. a fast and a delayed line, which are XOR-ed at the end of the path. We can use the output of the PDL (the XOR-ed value) as the trigger for the clocked-gate instead of the clock falling (or rising) edge as in Figures 10 and 11. The delay lines are using as many buffers (or back-to-back inverters) as needed to achieve the right timing until the moment comes when we need the trigger's edge. However, it might be not so straightforward to achieve the right timing, as indicated by [NT22]. When implemented in ASIC such delay lines have to be tunable in order to mitigate the dynamic timing variations caused by changes of the process, voltage, temperature and ageing. Without proper calibration in silicon, false positives may occur which in our case will result in glitches propagating via the clocked-gates. Such a glitch propagation violates the basic glitch-stopping assumption and might cause a leakage. Since from practical point of view such a calibration process requires additional efforts, we opted it out and favoured a solution with additional clocks.

Apart from the triggering/clocking mechanism the new solutions – `GSC_G` and `GSC_L` – differ from traditional `GSC_R` in their circuit evaluation process. In the pipelined register-based approach, all four circuits $CL_{1\{A-D\}}$ are evaluated simultaneously at each CLK_{TI} cycle. In contrast, in the new solutions, only one stage circuit (out of the four) will be evaluated at the virtual cycle corresponding to the CLK_{TI}. Circuits before the one currently being evaluated are preserved with their already computed values, while the subsequent circuits use predefined input values. For instance, in the clocked `AND`-gate solution, all circuits except for CL_{1A} are zeroized during the first virtual cycle and remain stable until their turn for evaluation arrives. In all four `GSC` solutions, glitches can occur

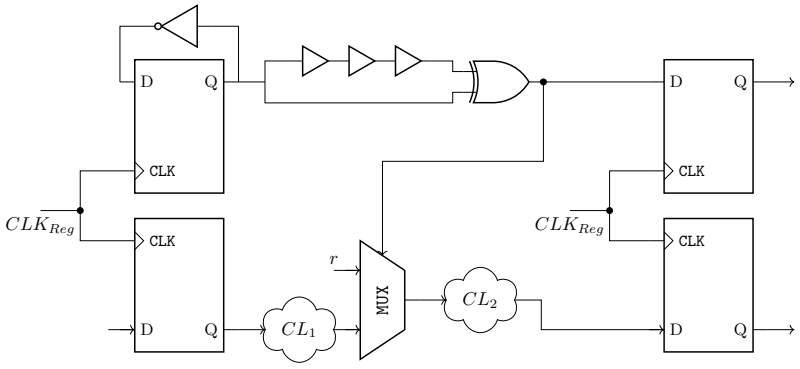


Figure 12: Use PDL-1 [ANN24] to generate trigger signal for the clocked-gate in two-stage masked circuit `GSC_woR`

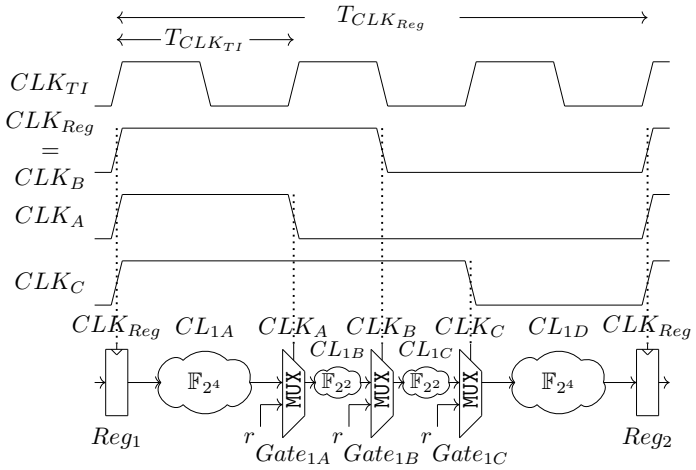


Figure 13: Clocked logic-gate-based glitch-stopping circuit with optimized virtual cycles.

during the evaluation of a sub-circuit within its designated (virtual) cycle. However, these sub-circuits are designed to be glitch-immune, adhering to the non-completeness property. This holds because `GSC_woR` preserved the original `GSC_R` masking scheme properties and in particular the non-completeness.

When comparing various glitch-stopping gate solutions as shown in Figures 3, 4, 9a and 9b, the `AND`-gate emerges as the simplest option with a cost of one AND and one NOT, with a delay on the data line of one AND. Next in simplicity is the `MUX`-gate with a cost of three NANDs and one NOT, and delay on the data line of two NANDs. The latch-gate, slightly more complex, requires four NANDs and two NOTs, also with a delay of two NANDs. Comparatively, the register (flip-flop) solution is more costly, doubling the latch cost while maintaining a similar signal delay.

All three new solutions offer better area efficiency than the register-based approach. In terms of cycle-wise latency, all these new solutions require only one single cycle for evaluation compared to four cycles in traditional register-based designs. However, it is important to note that the advantage in cycle reduction is mainly applicable to round-based architectures, as data pipelining is not feasible with the new solutions.

Last but not least, there are potential optimizations that reduce the complexity of the clock configuration and the time-wise latency. Firstly, time-wise latency can potentially be reduced in these new solutions by relaxing setup and hold times and thus slightly shortening the gating-related clocks. However, optimizing different critical paths per

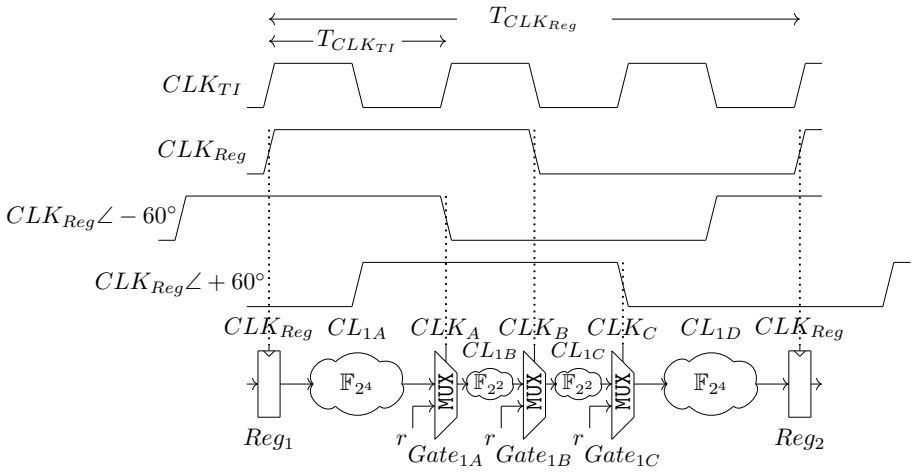


Figure 14: Clocked logic-gate-based glitch-stopping circuit with clocks with phase-shift.

stage instead of a single maximum critical path can further reduce time-wise latency as illustrated in Figure 13. In this example we assume that the sub-circuits in the first and fourth stages have 2 times longer critical paths than those sub-circuits in stages two and three. This is a plausible assumption given the fact that the first 2 sub-circuits contain multiplication over $GF(2^4)$ and affine operations compared to the later 2 circuits which contain multiplication over $GF(2^2)$. Therefore we can optimize the virtual cycles to fit the sub-circuits' critical paths and in that way the whole computation will be $3 * T_{CLK_{TI}}$ instead of $4 * T_{CLK_{TI}}$. Secondly, clocks with the same frequency can be used with phase shift, as shown in Figure 14. The phase shift can be achieved by either using a delay line or a phase-locked loop (PLL) circuit, which is less complex than clocks with different duty cycles.

4.5 Security in the Glitch-Extended Probing Model

In this section, Procedure 4.1 outlines the essential steps for designers to ensure the secure implementation of glitch-stopping circuits. Following this procedure, `GSC_woR` circuit gW is `derived` from `GSC_R` circuit gR. The steps described are critical to guarantee that glitches are effectively stopped across various stages. Subsequently, the security of the `GSC` hinges on the correct application of the masking techniques used. For example, Step 3 mandates the implementation of delays in the direct wires that pass through a combinational logic block from its input to the output. The aim is to mitigate (as an example) the risk that the time, required for the rising edge of CLK_A to reach $Gate_{1A}$ from Reg_1, is bigger than the time needed for the data propagating on a direct wire between them. Adding buffers (or back-to-back inverters) on such direct wire resolves this.

Our methodology for circuit `derivation` remains independent of the used security notion in the glitch extended probing model, such as TI, NI, SNI, or PINI. For simplicity, we collectively refer to these as *α-secure* `GSC`, where α represents any of these notions. Recall that a glitch-extended probe on a wire z models the impact of glitches by capturing all stable signals that contribute to the wire z. For the conventional `GSC_R` case those are flip-flop values, however for the `GSC_woR` cases such an extended probe captures the values of the clocked-gates instead. We are now ready to state our main result.

Theorem 1. *If a gadget gR is glitch-extended d-order α-secure `GSC_R`, then the `derived` gadget gW is also glitch-extended d-order α-secure `GSC_woR` (i.e., with clocked `AND`-gate, `MUX`-gate, and latch).*

Step 1. Start with a secure register-based implementation `GSC_R` named gR.

Optional. Perform leakage assessment tests on gR.

Step 2. Find the register layers and the critical path of gR.

Optional. Calculate the critical path for each combinational logic stages of gR, if the time-wise latency is to be optimised.

Step 3. Preserve the combinational logic of gR and insert delays on the direct wires.

Step 4. Add corresponding clocks for `GSC_L` or `GSC_G` as shown in Figures 10 and 11. Reduce the number of clocks whenever applicable.

Optional. Optimize the clocks if the time-wise latency is the goal.

Step 5. Replace the registers with the chosen type of glitch-stopping latches or gates for `GSC_L` or `GSC_G` respectively in the RTL or netlist level.

Step 6. Connect the gates with the correspondingly introduced new clocks and obtain `GSC_woR` named gW.

Step 7. Check the behavioural and timing correctness of gW.

Optional. Perform leakage assessment tests on gW.

Procedure 4.1: `GSC` design principles

If a composition of gadgets gR is glitch-extended d-order α-secure `GSC_R`, then the `derived` composition of gadgets gW is also glitch-extended d-order α-secure `GSC_woR`.

Proof. Consider a `GSC_woR` circuit gW which is built following the design principles given in Procedure 4.1 starting from a `GSC_R` circuit gR, in other words, the circuit gW is `derived` from the circuit gR. Such a masked circuit gW has clocked-gates instead of registers while it preserves the combinational sub-circuits and their timing conditions exactly as in gR. This guarantees that in gW no glitches propagate between two sequential sub-circuits via the clocked-gates exactly as this is the case for the gR circuit. At the beginning and the end of each virtual cycle all the sub-circuits in gW have stable signals exactly as for gR.

Let us now consider the extended probing model with glitches and compare the information received by the attacker observing both gW and gR. As explained above, gW has no pipelining and the sub-circuits are evaluated consecutively while in gR the pipelining is used and all sub-circuits are evaluated at the same cycle. The sub-circuits in gW that have been evaluated and those which are not yet in evaluation keep stable signals. A glitch-extended probe in a particular virtual cycle retrieves the same information for all sub-circuits which have already been evaluated and the one under evaluation in circuits gW and gR. The sub-circuits in gW which are not yet evaluated have no analogue in gR, however, those sub-circuits are evaluated with constant or random inputs which are independent of any data (sensitive or not) used in the gR circuit. Therefore, a glitch-extended probe over such (not yet evaluated) sub-circuit reveals no information to the attacker.

For both circuits gW and gR, the adversary has the same information and hence, the adversary advantage for both designs will be the same. So, we can conclude that if gR is a glitch-extended d-order α-secure gadget then gW is also a glitch-extended d-order α-secure gadget. Similarly, if gR is a composition of glitch-extended d-order α-secure gadgets then

gW is a composition of glitch-extended d-order α-secure gadgets. This concludes the proof. □

It has been noted that early signal propagation is the main security threat to asynchronous designs, particularly those based on DRP logic. In these glitch-free circuits, the timing information for the initial signal propagation is enough to reveal the secret. However, glitches are present in `GSC_R`, early signal propagations are not only related to the secret value and have not been shown to cause leakage. Thus `GSC_woR` are not vulnerable to early signal propagation similar to the `GSC_R`.

At the end of this section we stress that there are no layout-dependent constrains which a designer has to take into account. Procedure 4.1 (step. 2,3,4) demonstrates that the circuits are preserved and hence their time-wise behavior is preserved as well. The Place-and-Route procedure will make sure that the timing constrains are met and is not going to optimize any of the "gates" since this is prevented by the typical for masking "don't touch" instances.

5 Glitch-Stopping Circuits in FPGA

In this section, we describe how the three new solutions for ASIC presented in Section 4 can be implemented on FPGA taking into account the FPGA specifics. Unlike ASICs which are designed for a specific purpose and manufactured for that use case, FPGAs can be reprogrammed to perform a wide variety of tasks. However, this versatility comes with limitations and high complexity of placement and routing. Although the FPGA primitives can perform the same logical functions as the corresponding ASIC designs, their implementation is more complex. Indeed, the designer has to use the provided FPGA native primitives which might not entirely fit for the glitch-stopping purpose, for example, be more glitchy because they have more inputs. Compared to the register-based implementation of `GSC` which is on algorithmic RTL level, the `GSC` without registers requires deeper intrusion like preserving circuits time, taking care of the additional clocks, using less standard clocked-gates instead of registers. Some of these changes have to be performed on netlist level.

5.1 Primitives in FPGAs for Glitch-Stopping Circuit

FPGAs consist of Configurable Logic Blocks (CLBs) linked by a reconfigurable hierarchy of customizable interconnects. These CLBs can be customized to execute complex combinational functions typically utilizing Look-Up Table (LUT) circuits, flip-flops, and various logic components. This setup enables the implementation of both combinational and sequential digital circuits. Each CLB interfaces with a switch matrix, linking it to a broader routing matrix for flexible connectivity with other CLBs and I/O blocks within the FPGA.

Since we implement and evaluate our designs in the Xilinx Spartan-6 FPGAs on the Sakura-G board, we first introduce the instances of the required primitives in the Spartan-6 FPGAs. These primitives are also present in newer Xilinx FPGA models, accessible through identical instantiation methods. For Intel FPGAs, corresponding primitives adhering to the specifications of these FPGA models should be selected based on their glitch-stopping characteristics.

Look-up table and multiplexers in Spartan-6 Spartan-6 FPGAs utilize six-input Look-Up Table `LUT` circuits as function generators. Each `LUT` features six independent inputs, labelled `A1` to `A6`, and two independent outputs, `O5` and `O6`. These `LUT`s can execute any Boolean function with six inputs, including the `AND`-gate. Additionally, `SLICEL` and

SLICEM slices in Spartan-6 FPGAs incorporate three 2-to-1 multiplexers, named MUXF7, along with the basic LUTs. These multiplexers combine function generators, allowing for up to eight-input functions within a slice when integrated with LUTs. The MUXF7, a straightforward 2-to-1 multiplexer, receives one of its inputs from the LUT output O6 and can be driven by any internal net, making it suitable for use as a glitch-stopping MUX-gate

Latches and storage elements in Spartan-6 Each slice within the Spartan-6 FPGA structure incorporates also eight storage elements. Out of them, four storage elements can be configured either as edge-triggered FFs or as level-sensitive latches. Spartan-6 devices also feature an additional set of four storage elements that can exclusively be set as edge-triggered FFs.

In a normal synchronized circuit, the flip-flops are used to store the intermediate values of the circuit and are instantiated by using FDCE which represents a D-flip-flop with an asynchronous reset. This is done as part of the synthesis and mapping procedure of Xilinx design suits, so it is not exposed to the user. However, we can also explicitly instantiate the latches in the design by using LDCE which is a transparent level-sensitive data latch. We will use LDCE as a glitch-stopping latch.

Beyond the standard roles of such storage elements, they can also be used to implement logic gates. For example, a level-sensitive latch can emulate basic logic gates. Two primitives in Spartan-6 that help in leveraging the latch function as logic gates are the AND2B1L and OR2L. AND2B1L is a 2-input AND gate with one of the inputs inverted. This configuration uses the data input of the latch as the inverted AND-gate input (labelled as DI) and the latch's asynchronous clear input (labelled as SRI) as the other AND-gate input. OR2L is a 2-input OR gate that combines the data input of the latch with the latch's asynchronous preset.

The design purpose of these primitives is to broaden the logic capability by adding an additional external input. Typically, the latch data input is sourced from the output of a LUT located in the same slice. Similar to MUXF7, AND2B1L is a 2-input AND gate with one input inverted implemented on the latch function. We will use either AND2B1L or LUT as a glitch-stopping AND-gate. Thus on our Spartan-6, we have identified four ways to implement the three clocked-gates.

5.2 AES Implementations

In our implementations, we will use MUXF7 only with the second input set to 0, therefore implementing glitch-stopping AND-gate. Further, we will present several implementations for the two AES designs (S-boxes with two and four stages). Namely, for the design with two stages S-box we present four implementations AND2B1L, MUXF7, LUT and LDCE; while for the design with four stages S-box, we present two implementations MUXF7 and LDCE.

Implementations of AES with the Two-Stage S-Box In synchronous logical circuits, the clock signal is routed from its source, such as a Phase-Locked Loop, crystal oscillator, or clock divider, to all relevant components including registers and latches. This process, known as clock tree synthesis, is a crucial step in the circuit design workflow, taking place after the placement phase and before the routing phase.

We start from the GSC_R implementation, as described in Section 3.2, focusing solely on the 2-stage S-box illustrated in Figure 7. The remainder of the implementation is the serialized AES encryption architecture assuming the S-box is evaluated in a single cycle. We need to replace the register between the two stages with a glitch-stopping gate, which could be either a clocked latch or clocked AND. Note that in this simple configuration when only one register has to be replaced, we can maintain the same clock signal as in the

standard register configuration. However, the clock routing should connect this signal to the pin of the glitch-stopping gate as referred to in Figure 9a.

In our implementation, an external 6 MHz clock signal from a waveform generator is provided to the clock pin of the Spartan-6 FPGA. Internally, this 6 MHz clock signal is routed to all the registers, while we have to *manually* set the clock pins of the glitch-stopping gates to be `CLOCK_DEDICATED_ROUTE = FALSE`. Thus, the 5×8 bits of the registers are now replaced by the same quantity of clocked-gates.

Implementations of AES with the Four-Stage S-Box We start with the `GSC_R` implementation described in Section 3.3 and again focus only on the 4-stage S-box as depicted in Figure 8. The FPGA is supplied with a $24MHz$ clock. In this implementation, the Xilinx clock wizard is utilized to generate clock signals with different duty cycles, namely 25%, 50%, and 75% duty cycles. For a latch-based pipeline, the clock outputs are $6MHz$, $12MHz$, and $12MHz$ with 180° phase shift. The clocking wizard in the ISE or Vivado design tools for Xilinx has a GUI interface which allows implementing such clock signals directly. Again as it was the case for the S-box with 2 stages, we have to manually re-route to the corresponding registers clock pins of the the glitch-stopping gates replacing those pipeline internal registers. Similar to the 2-stage S-box scenario, it is necessary to manually set the clocks signal to the pins of the glitch-stopping gates `CLOCK_DEDICATED_ROUTE = FALSE`, which replace the three internal pipeline registers. Thus, the 114 bits of the registers are now replaced by the same quantity of clocked-gates.

5.3 Performance Comparison

In Table 1, we start by listing the AES unprotected S-box [MPL+11] based on Canright [Can05], which is a standard unprotected synchronous circuit with registers. We then compare this with the latest implementations of low-latency glitch-stopping circuits with registers (`GSC_R`) [GIB18] and state-of-the-art GFC, which are asynchronous [SBB+22, SBHM20, NGPM22]. A direct comparison is challenging due to variations in CMOS libraries across different studies. WDDL-based solutions typically eliminate the need for registers, thus reducing area costs, but they still have a substantial overhead in combinational logic due to DRP logic. Additionally, these circuits have a higher logic depth and require a pre-charging phase, which in synchronous designs translates to one cycle. However, GFCs are self-synchronizing and may get better latency in time. Clear advantage for the GFC versus `GSC_R` can be seen when latency is measured in cycles.

We have implemented two specific designs of `GSC_R` using $t + 1$ and $t + 2$ TI, which serve as a baseline for our comparisons. Both implementations are described in Section 3.2 and Section 3.3. All our implementations use `Nangate45` CMOS library. The syntheses are re-run with Synopsys Design Compiler 2017.09 using the command `compile -exact_map` and with the `set_dont_touch` flag applied to all nets. We made this choice because the `compile` command includes fewer default optimizations compared to `compile_ultra`. The command flag `-exact_map` prevents sequential mapping from encapsulating combinational logic into sequential cells. We note that the instantiating of glitching-stopping circuits in the synthesis libraries and on the FPGAs are different. Namely, in the HDL code the `AND`-gates, `MUX`-gates and latches are respectively instantiated with `AND2_X1`, `MUX2_X1` and `DLH_X1` elements for `Nangate45` while for Xiling FPGA (as already explained in Section 5.1) those are instantiated with `AND2B1L`, `LUT`, `MUXF7` and `LDCE`. In the last part of Table 1 we list our results for the implementations of `GSC_R`.

Firstly, the glitch-stopping circuits without registers `GSC_woR` are smaller than the corresponding circuits with registers `GSC_R`. The 2-Stage versus 4-Stage cases also show the trend that the more stages the initial solution has the larger is the area reduction. Secondly, our solutions match the cycle-wise latency of GFC. For latency measured in time, we estimate the maximum frequency based on the longest data arrival time from one

glitch-stopping (sequential) layer to another. In GSC_R, the propagation delay occurs on the data path between registers, triggered by the clock's positive edges. In circuits without registers GSC_woR, glitch-stopping layers operate under specific clock configurations, and their propagation delays span all stages of the combinational logic. The timing report is generated by Synopsys's static timing analysis (STA) tool, although no user-defined clock constraints are applied to the implementations. Note that the STA tool considers latches and registers as sequential elements, compared to AND and MUX gates as logic elements. Thus, the timing estimations of GSC_G and GSC_L are not directly comparable. As a rule of thumb the latency in time can be roughly calculated as the GSC_R masked latency multiplied with the number of stages.

We further elaborate on the impact of GSC_woR on the latency in time. As explained in Section 3.4, the latency overhead in traditional GSC_R is attributed to the XOR operations used for re-sharing and compression, along with the time consumed by registers inserted for separating the circuits. Typically, these implementations involve between 8 to 12 XORs and one to three registers. In contrast, GFC do not have the overhead of the registers. The GSC_woR also eliminates overhead of registers, but the synchronization of the circuits costs time. However, this synchronisation can be further optimized per stage and the overhead can be made similar to the asynchronous GFC case.

As it can be seen in Table 1 and Table 2, both GSC_R and Latch-based GSC_woR designs achieve the same time-wise latency for the S-box. Taking into account that the logic which in GSC_R was spread over 2 or 4 cycles, it is processed in GSC_woR in a single cycle we see that $609/4 = 152$ and $387/2 = 193$ holds. However, AND and MUX-based GSC_woR designs can achieve higher S-box time-wise latency by employing optimization of the virtual cycles as depicted in Figure 13. In the 2-stage AES S-box both stages have nearly equal critical paths hence only a small optimization (211 versus 193) can be achieved. However, the 4-stage AES S-box has 2 middle stages with critical path more than twice shorter than the other 2 stages, thus a frequency of 217 MHz instead of 152 MHz only.

In summary, implementing GSC_woR, as outlined in Procedure 4.1, involves the usage of additional clocks to maintain synchronization of the combinational logic in contrast to the conventional GSC_R. This approach not only improves latency in terms of time and cycles but also offers better area efficiency. However, the GSC_woR design method is more engineering-intensive in executing all the steps in Procedure 4.1. Whether the overall gain in cost compensates for the additional efforts and risks is case dependent. While direct comparison with glitch-free circuits is not straightforward, we anticipate achieving comparable latency in time and equal latency in cycles, but with better area efficiency. Moreover, this method is compatible with standard digital design tools and libraries, making it more accessible for practical implementation.

5.4 Leakage Evaluation

None of the existing formal verification tools like SILVER [KSM20], COCO [GHP+20], etc. can be used to evaluate the new glitch-stopping without registers (GSC_woR) designs. Thus similarly to [SBHM20, NGPM22, SBB+22] the only remaining way to demonstrate the security of our designs is to use Test Vector Leakage Assessment (TVLA) by Goodwill et al. [GJJR11] which is widely used to evaluate the security of cryptographic designs against SCA. Instead of testing against potential attacks, TVLA's goal is to identify, using the power traces, the leakage points of the secrets, irrespective of whether these leakages are immediately exploitable or not. The non-specific t-test is applied which verifies that our implementations do not show first-order leakage. For this, the measured power traces are divided into two sets, where the first set $\mathcal{S}_0$ uses fixed plaintexts and the second $\mathcal{S}_1$ receives random plaintexts. The t-test verifies whether the first-order statistical moments of these two sets are the same. The null hypothesis of the t-test states that "$\mathcal{S}_0$ and $\mathcal{S}_1$ are drawn from populations with the same mean."

Table 1: Comparison of first-order secure AES S-box implementations

Solutions	Area (kGE)	Latency ($cycles$)	Max clock (MHz)	CMOS Library
Unprotected S-box [MPL+11]	0.23	1	-	UMC L180
Glitch-Free Circuits (GFC)				
Self-Timed Masking [SBB+22]	6.07	1	4.8	STCMOSM40
LMDPL [SBHM20]	3.84	1	400	GF28
SESYM [NGPM22]	3.98	1	192	UMC65
Glitch-Stopping Circuits with Registers (GSC_R)				
GLM [GIB18]	60.73	1	356	UMC90
GLM [GIB18]	6.74	2	584	UMC90
5-share ($t+2$ TI) 2-Stage S-Box	21.86	2	387	Nangate45
3-share ($t+1$ TI) 4-Stage S-Box	5.31	4	609	Nangate45
Glitch-Stopping Circuits without Registers (GSC_woR)				
AND-based 2-Stage S-Box	19.41	1	211	Nangate45
MUX-based 2-Stage S-Box	19.45	1	211	Nangate45
Latch-based 2-Stage S-Box	19.63	1	193	Nangate45
AND-based 4-Stage S-Box	4.76	1	217	Nangate45
MUX-based 4-Stage S-Box	4.90	1	217	Nangate45
Latch-based 4-Stage S-Box	5.12	1	152	Nangate45

Table 2: Comparison of propagation delays (ns, Mhz) of the different stages in the masked GSC_R S-boxes synthesized with Synopsys

S-box Instance	Stage 1 (ns, Mhz)	Stage 2 (ns, Mhz)	Stage 3 (ns, Mhz)	Stage 4 (ns, Mhz)
3-share 4-stage AES S-box	1.58	0.73	0.64	1.64
	632	1369	1562	609
5-share 2-stage AES S-box	2.18	2.58	NA	NA
	458	387		

Large absolute values of this t-statistic provide evidence for rejecting the null hypothesis with a certain confidence. In side-channel research, a widely adopted threshold for the t-test is 4.5. If the t-test value computed from the power traces exceeds this threshold, it signifies a potential security vulnerability in the tested implementation. Note that we have used this threshold value for the t-tests, however one can apply the methodology by Ding et al. [DZD+17] to obtain a proper threshold when the number of samples is large, for example, $t = 6$ for 10K samples and $\alpha = 10^{-5}$ or $t = 7.4$ for 13K samples. Since our AES serialized implementations have 20K samples even higher t can be applied.

For the practical experiments, we mainly used a Xilinx Spartan-6 FPGA on a SAKURA-G evaluation board. We collect power traces using a digital oscilloscope at a sampling rate of $500MS/s$. The encryption is performed up to 100 million times receiving either fixed or random masked plaintexts. To do the t-test, we collect 4K sample points for the S-box implementations and 20K samples for the AES with serialized architecture implementations. As usual, we performed TVLA on our design with the PRNG both active and inactive to check that the tool-chain performs as expected. We perform the first-order t-test, the summary of the results is given in Table 3, while the figures for the t-tests are provided in Appendix A.

The FPGA primitives, although perform the same logical functions as the corresponding digital designs, can introduce glitches from internal hardware. For example, an LUT instance on FPGA when initiated to be an AND-gate is a glitchy hardware unit with five or six input signals. TVLA tests were conducted on both a standalone 2-stage 5-share S-box and the

Table 3: Summary of the first-order implementations using the FPGA primitives on Sakura-G

Synthesis	FPGA	Implementation	TVLA	Figure
AND or MUX gates	MUXF7	2-stage 5-share AES S-box	*Pass*	Fig. 15
AND-gate	AND2B1L	2-stage 5-share AES S-box	*Pass*	Fig. 16
AND-gate	LUT	2-stage 5-share AES S-box	*Fail*	Fig. 17
Latch	LDCE	2-stage 5-share serialized AES	*Pass*	Fig. 18
AND or MUX gates	MUXF7	2-stage 5-share serialized AES	*Pass*	Fig. 19
AND-gate	AND2B1L	2-stage 5-share serialized AES	*Fail*	Fig. 20
AND-gate	LUT	2-stage 5-share serialized AES	*Fail*	Fig. 21
Latch	LDCE	4-stage 3-share serialized AES	*Fail*	Fig. 22
AND or MUX gates	MUXF7	4-stage 3-share serialized AES	*Pass*	Fig. 23
MUX gates @30MHz	MUXF7	4-stage 3-share serialized AES	*Pass*	Fig. 24

Table 4: Summary of the serialized AES implementations using the FPGA primitives on Sakura-X

Synthesis on FPGA	Implementation	TVLA	Security Order	Figure
MUXF7	4-stage 3-shares	*Pass*	first-order uni-variate	Fig. 25
MUXF7	6-stage 3-shares	*Pass*	second-order bi-variate	Fig. 26

serialized AES-128 encryption incorporating such an S-box, evaluating the efficiency of AND-gate implementations within LUTs. As indicated in Figures 17 and 21, the implementation through LUT was ineffective in stopping glitches, evidenced by the failure of both t-tests.

This failure leads to the experiments using other primitives on FPGA, like MUXF7 and AND2B1L, that can build the same logic function of AND-gate. In the 2-stage 5-share S-box implementations, both MUXF7 and AND2B1L show no leakages as illustrated in Figures 15 and 16. However, when the tests are extended to the serialized AES-128 encryption Figure 19 shows that MUXF7 achieves the first-order security, while AND2B1L fails Figure 20. The key differences between the AND2B1L and MUXF7 primitives are that AND2B1L, is a larger unit shared with latches and registers and thus it requires more complex routing. Additionally, the full AES serialized encryption, combining synchronous state arrays and control logics, adds complexity and potential leakage sources. Due to negative results with AND2B1L and LUT, they were not tested on the 4-stage 3-share serialized AES. However, MUXF7 showed no leakage in the 4-stage 3-share implementation (Figure 23), suggesting its suitability as a glitch-stopping gate in FPGA environments.

The test using latches (LDCE) with the 2-stage 5-share S-box in AES serialized encryption showed no first-order leakage (Figure 18), where the clock configuration is the same as the AND-gate. However, the AES serialized encryption with a 4-stage 3-share S-box exhibited leakage (Figure 22), potentially due to complex clock configurations that inadequately prevented glitch propagation ac-ross imperfect clock edges.

For all evaluations on Sakura-G given in Table 3 we use $12MHz$ clock, except for the last one where we show that clocking faster at $30MHz$ (with 25%, 50% and 75% duty cycles) doesn't change the result for the AND or MUX gates.

In order to illustrate transfer-ability of the results to another platform we show in Table 4 that MUXF7 based 4-stage 3-share serialized AES implementation on Sakura-X is still first-order secure.

Finally, in order to demonstrate that the proposed approach is independent of the security order, we used the RTL of the 6-stage 3-share 2nd-order serialized AES implementation by De Cnudde et al. [CRB+16]. The modification from GSC_R to MUXF7 based GSC_woR implementation is second-order secure as shown in Table 4.

In summary, we have shown that MUX or AND glitch-stopping gate can be implemented

on FPGA via `MUXF7`. We leave as a further research topic to show that `LDCE` is also a way to implement latch-based glitch-stopping on FPGA.

6 Conclusions

In this paper, we introduced a method to build secure hardware masking without registers, offering improved performance efficiency and area reduction compared to conventional register-based methods. Our method employs clocked `AND`-gates, latches, and multiplexers to effectively stop glitch propagation. The `GSC_woR` approach is proven, in the glitch-extended probing model, to retain the security level of `GSC_R`. We experimentally confirmed, via TVLA evaluations, the first-order and second-order security of FPGA implementations using `MUXF7` and `LDCE`.

Acknowledgements. This work was supported by CyberSecurity Research Flanders with reference number VR20192203.

References

[ADN+22] Amund Askeland, Siemen Dhooghe, Svetla Nikova, Vincent Rijmen, and Zhenda Zhang. Guarding the first order: The rise of AES maskings. In Ileana Buhan and Tobias Schneider, editors, *Smart Card Research and Advanced Applications - CARDIS 2022*, volume 13820 of *Lecture Notes in Computer Science*, pages 103–122. Springer, 2022.

[ANN23] Amund Askeland, Svetla Nikova, and Ventzislav Nikov. Who watches the watchers: Attacking glitch detection circuits. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(1):157–179, Dec. 2023.

[ANN24] Amund Askeland, Svetla Nikova, and Ventzislav Nikov. Who watches the watchers: Attacking glitch detection circuits. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(1):157–179, 2024.

[AZN21] Victor Arribas, Zhenda Zhang, and Svetla Nikova. LLTI: low-latency threshold implementations. *IEEE Trans. Inf. Forensics Secur.*, 16:5108–5123, 2021.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 116–129. ACM, 2016.

[BGN+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014*, volume 8874 of *LNCS*, pages 326–343. Springer, 2014.

[BKN20] Dusan Bozilov, Miroslav Knezevic, and Ventzislav Nikov. Optimized threshold implementations: Minimizing the latency of secure cryptographic accelerators. In S. Belaid and T. Guneysu, editors, *Smart Card Research and Advanced Applications - CARDIS 2020*, Lecture Notes in Computer Science, pages 20–39. Springer, 2020.

[BP11] Joan Boyar and Rene Peralta. A depth-16 circuit for the aes s-box. Cryptology ePrint Archive, Paper 2011/332, 2011.

[Can05] David Canright. A very compact S-box for AES. In *Cryptographic Hardware and Embedded Systems - CHES 2005 Proceedings*, pages 441–455, 2005.

[CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Transactions on Computers*, 70(10):1677–1690, 2021.

[CRB+16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with d+1 shares in hardware. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 194–212. Springer, 2016.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

[DZD+17] A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsi Fei. Towards sound and optimal leakage detection procedure. In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications - CARDIS 2017*, volume 10728 of *Lecture Notes in Computer Science*, pages 105–122. Springer, 2017.

[FGP+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, (3):89–120, 2018.

[GHP+20] Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. Coco: Co-design and co-verification of masked software implementations on cpus. Cryptology ePrint Archive, Paper 2020/1294, 2020.

[GIB18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. Generic low-latency masking in hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):1–21, 2018.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation, September 2011.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. *IACR Cryptol. ePrint Arch.*, 2016:486, 2016.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KM22] David Knichel and Amir Moradi. Low-latency hardware private circuits. *Proceedings of the 2022 ACM CCS Conference*, 2022.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. Silver - statistical independence and leakage verification. Cryptology ePrint Archive, Paper 2020/634, 2020.

[LMM23] Daniel Lammers, Nicolai Müller, and Amir Moradi. Glitch-free is not enough - revisiting glitch-extended probing model. Cryptology ePrint Archive, Paper 2023/035, 2023.

[MPL+11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of aes. In *EUROCRYPT*, volume 6632, pages 69–88, 2011.

[MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In *Cryptographic Hardware and Embedded Systems - CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[MS16] Amir Moradi and Tobias Schneider. Side-channel analysis protection and low-latency in action - - case study of PRINCE and midori -. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Science*, pages 517–547, 2016.

[NGPM22] Rishub Nagpal, Barbara Gigerl, Robert Primas, and Stefan Mangard. Riding the waves towards generic single-cycle masking in hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):693–717, 2022.

[NNR19] Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Decomposition of permutations in a finite field. In *Cryptography and Communications*, volume 11, pages 379—-384, 2019.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Inform. and Commun. Secur., ICICS 2006, Proceedings*, volume 4307 of *LNCS*, pages 529–545. Springer, 2006.

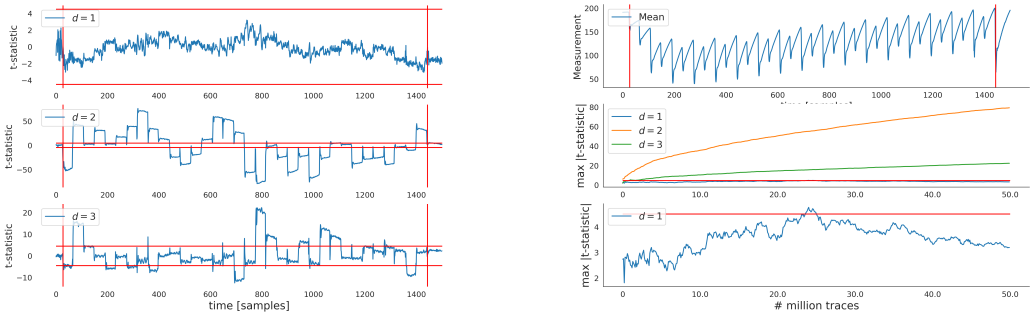
[NT22] Daniel Nemiroff and Carlos Tokunaga. Fault-Injection Countermeasures Deployed at Scale. Technical report, Intel Corporation, 2022.

[PAB+22] Enrico Piccione, Samuele Andreoli, Lilya Budaghyan, Claude Carlet, Siemen Dhooghe, Svetla Nikova, George Petrides, and Vincent Rijmen. An optimal universal construction for the threshold implementation of bijective s-boxes. Cryptology ePrint Archive, Paper 2022/1141, 2022.

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In *Advances in Cryptology - CRYPTO 2015 Proceedings, Part I*, pages 764–783, 2015.

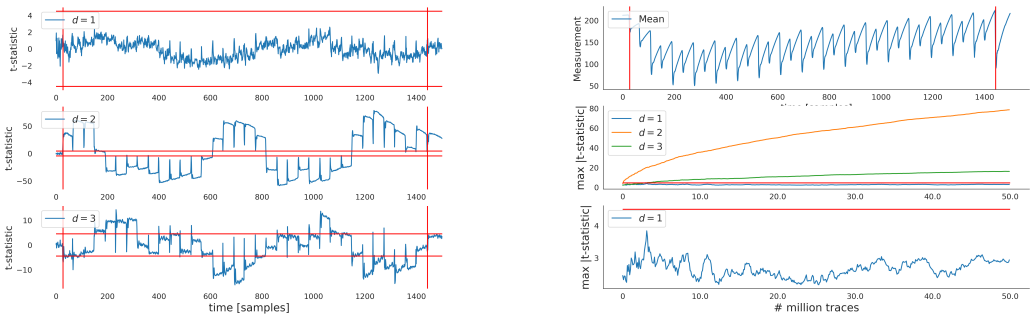
[RCN03] Jan M. Rabaey, Anantha P. Chandrakasan, and Borivoje Nikolić. *Digital integrated circuits: a design perspective*. Prentice Hall electronics and VLSI series. Pearson Education, Upper Saddle River, N.J, 2nd ed edition, 2003.

[SBB+22] Mateus Simoes, Lilian Bossuet, Nicolas Bruneau, Vincent Grosso, and Patrick Haddad. Self-timed masking: Implementing first-order masked s-boxes without registers. *IACR Cryptol. ePrint Arch.*, page 641, 2022.



(a) The final t-test result (b) The maximum t-test value evolving

Figure 15: The 2-stage 5-share AES S-box using `MUXF7`



(a) The final t-test result (b) The maximum t-test value evolving

Figure 16: The 2-stage 5-share AES S-box using `AND2B1L`

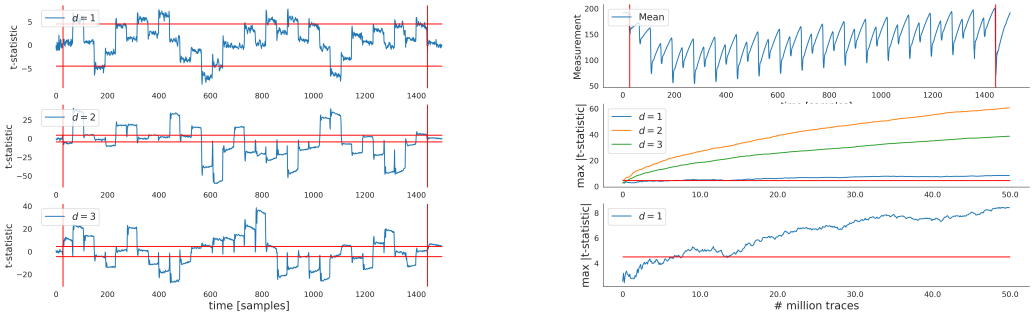
[SBHM20] Pascal Sasdrich, Begül Bilgin, Michael Hutter, and Mark E. Marson. Low-latency hardware masking with application to AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):300–326, 2020.

[TV04] Kris Tiri and Ingrid Verbauwhede. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *Design, Automation and Test in Europe Conference and Exposition (DATE 2004*, pages 246–251. IEEE Computer Society, 2004.

[WH10] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, USA, 4th edition, 2010.

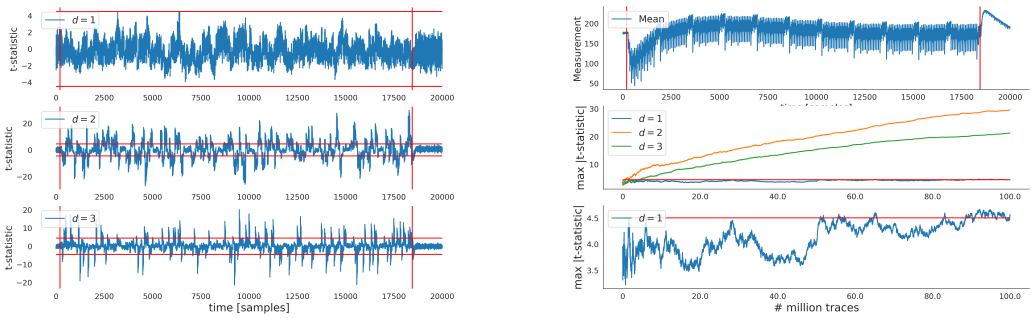
[Wik24] Wikipedia. Flip-flop (electronics) — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Flip-flop%20(electronics)&oldid=1193270436, 2024. [Online; accessed 08-January-2024].

A TVLA Test Results



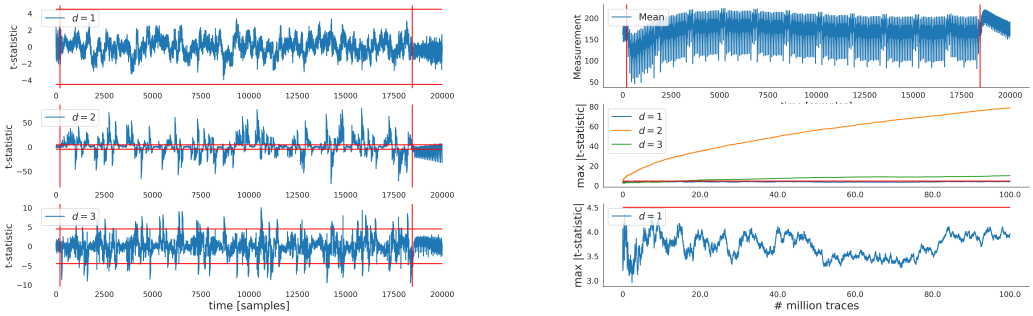
(a) The final t-test result (b) The maximum t-test value evolving

Figure 17: The 2-stage 5-share AES S-box using `LUT`



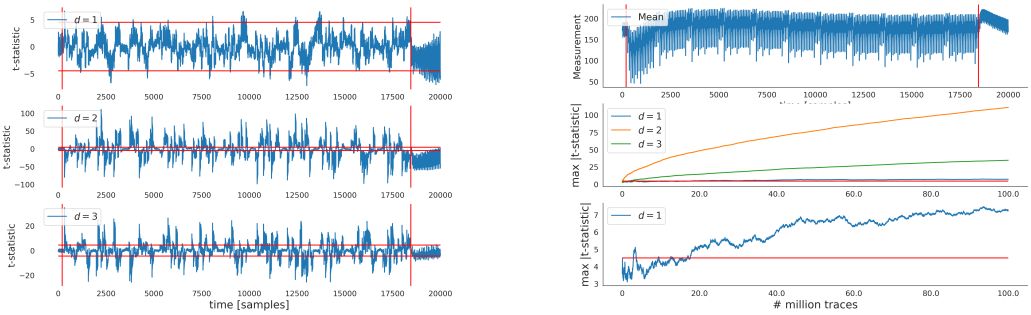
(a) The final t-test result (b) The maximum t-test value evolving

Figure 18: The 5-share serialized AES using the 2-stage S-box in `LDCE`



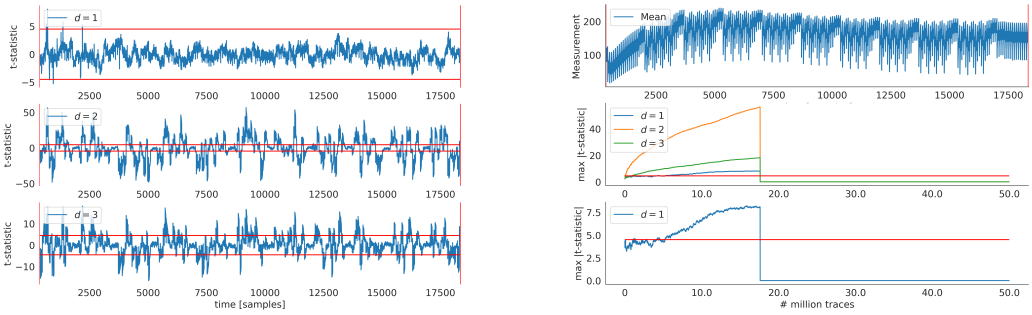
(a) The final t-test result (b) The maximum t-test value evolving

Figure 19: The 5-share serialized AES using the 2-stage S-box in `MUXF7`



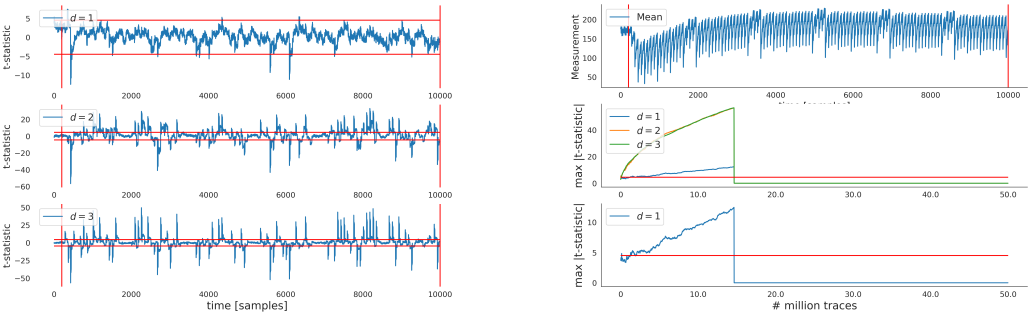
(a) The final t-test result (b) The maximum t-test value evolving

Figure 20: The 5-share serialized AES using the 4-stage S-box in `AND2B1L`



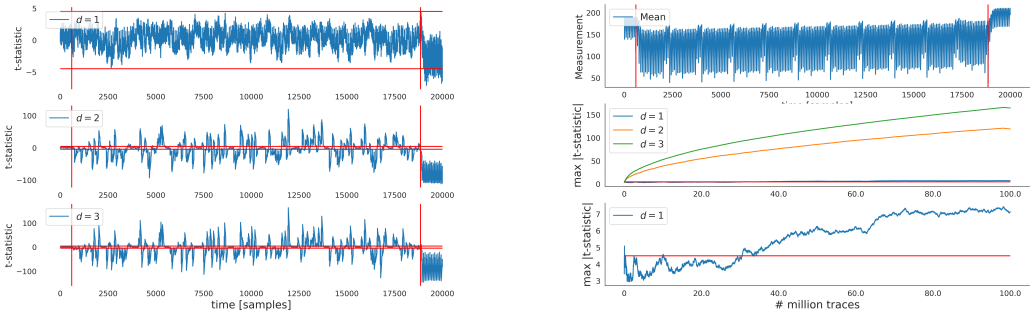
(a) The final t-test result (b) The maximum t-test value evolving

Figure 21: The 5-share Serialized AES using `LUT`



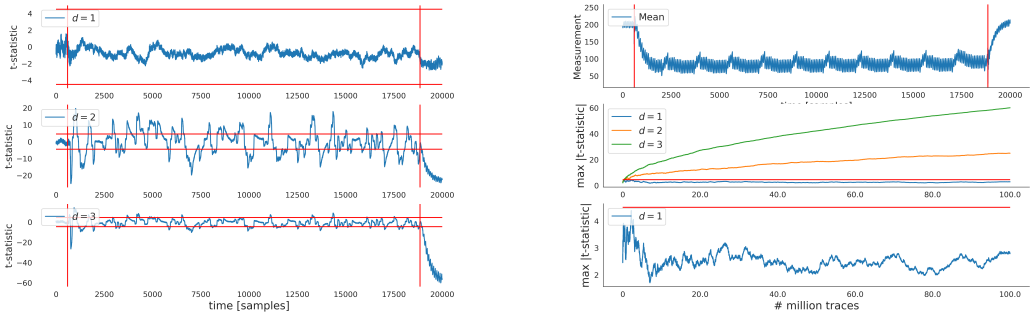
(a) The final t-test result (b) The maximum t-test value evolving

Figure 22: The 3-share serialized AES using the 4-stage S-box in `LDCE`



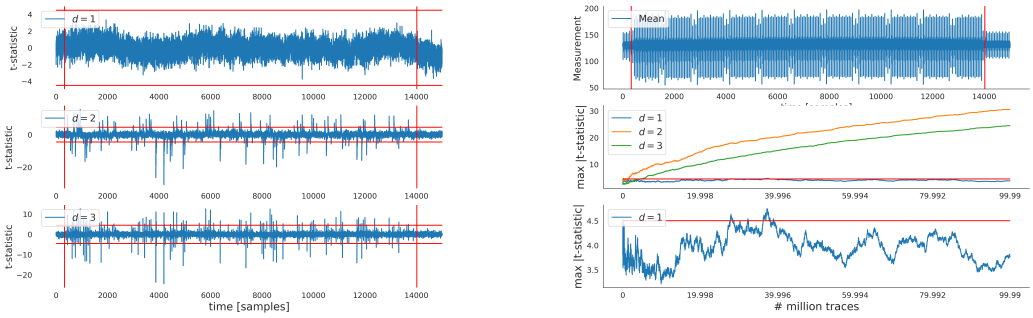
(a) The final t-test result (b) The maximum t-test value evolving

Figure 23: The 3-share serialized AES using the 4-stage S-box in `MUXF7`



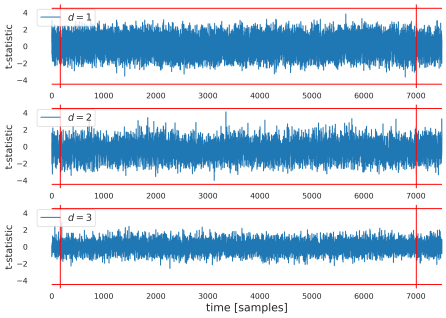
(a) The final t-test result (b) The maximum t-test value evolving

Figure 24: The 3-share serialized AES using the 4-stage S-box in `MUXF7` on Sakura-G with 30Hz input clocks

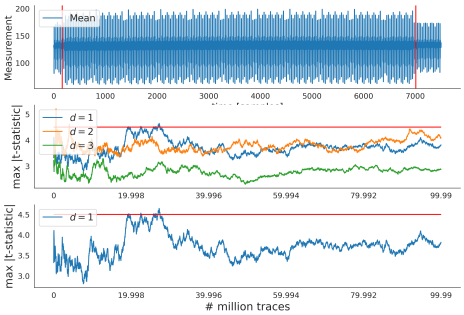


(a) The final t-test result (b) The maximum t-test value evolving

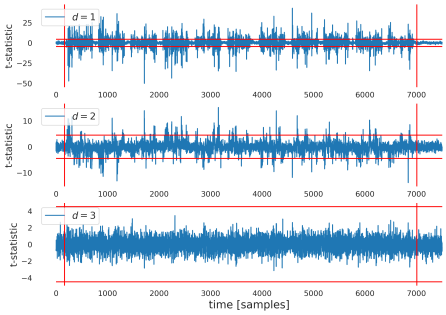
Figure 25: The 3-share first-order serialized AES using the 4-stage S-box in `MUXF7` on Sakura-X



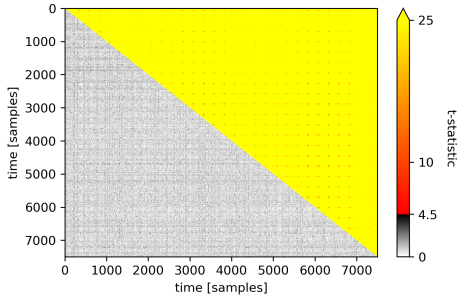
(a) The final uni-variate t-test result: PRNG on



(b) The maximum t-test value evolving



(c) The final uni-variate t-test result: PRNG off



(d) The final bi-variate t-test result: PRNG on (bottom left) and PRNG off (top right)

Figure 26: The 3-share second-order serialized AES using the 6-stage S-box in `MUXF7` on Sakura-X