DTA-PUF: Dynamic Timing-aware Physical Unclonable Function for Resource-constrained Devices

IOANNIS TSIOKANOS, JACK MISKELLY, CHONGYAN GU, MAIRE O'NEILL, and GEORGIOS KARAKONSTANTIS, Institute of Electronics, Communications and Information Technology (ECIT), Queen's University Belfast, UK

In recent years, physical unclonable functions (PUFs) have gained a lot of attention as mechanisms for hardware-rooted device authentication. While the majority of the previously proposed PUFs derive entropy using dedicated circuitry, software PUFs achieve this from existing circuitry in a system. Such software-derived designs are highly desirable for low-power embedded systems as they require no hardware overhead. However, these software PUFs induce considerable processing overheads that hinder their adoption in resource-constrained devices. In this article, we propose DTA-PUF, a novel, software PUF design that exploits the instruction- and data-dependent dynamic timing behaviour of pipelined cores to provide a reliable challenge-response mechanism without requiring any extra hardware. DTA-PUF accepts sequences of instructions as an input challenge and produces an output response based on the manifested timing errors under specific over-clocked settings. To lower the required processing effort, we systematically select instruction sequences that maximise error-rate. The application to a post-layout pipelined floating-point unit, which is implemented in 45 nm process technology, demonstrates the effectiveness and practicability of our PUF design. Finally, DTA-PUF requires up to 50× fewer instructions than existing software processor PUF designs, limiting processing costs and resulting in up to 26% power savings.

CCS Concepts: • **Security and privacy** → **Embedded systems security**; **Hardware-based security protocols**; • **Hardware** → *Emerging tools and methodologies*; *Timing analysis*; *VLSI design manufacturing considerations*

Additional Key Words and Phrases: Device identification, dynamic timing analysis, FPU, hardware security, IoT, low cost PUF, pipeline, resource-constrained devices, software-based processor PUF, timing errors

ACM Reference format:
Ioannis Tsiokanos, Jack Miskelly, Chongyan Gu, Maire O'Neill, and Georgios Karakonstantis. 2021. DTA-PUF: Dynamic Timing-aware Physical Unclonable Function for Resource-constrained Devices. *J. Emerg. Technol. Comput. Syst.* 17, 3, Article 32 (August 2021), 24 pages.
https://doi.org/10.1145/3434281

The presented research effort is partially supported by the UK Engineering and Physical Science Research Council (EPSRC) under Grant No. EP/N508664/-CSIT2, and by the European Commission (H2020-EU) under Grants No. 688540 (UniServer) and No. 732631 (OPRECOMP).

Authors' addresses: I. Tsiokanos, J. Miskelly, C. Gu, M. O'Neill, and G. Karakonstantis, Institute of Electronics, Communications and Information Technology (ECIT), Queen's University Belfast, Belfast, UK, emails: {itsiokanos01, jmiskelly08, c.gu,m.oneill, g.karakonstantis}@qub.ac.uk.

1550-4832/2021/08-ART32 $15.00
https://doi.org/10.1145/3434281

1 INTRODUCTION

The continuous scaling of transistor sizes and technology advances are driving the demand for low-power portable devices [36, 64]. The **International Data Corporation (IDC)** [3] estimates that there will be 41.6 billion devices connected to the Internet, generating 79.4 **zettabytes (ZB)** of data in 2025. Therefore, with the advent of **Internet of Things (IoT)** era, secure communication among computing devices is of prime importance [6, 23]. Traditional security methods (e.g., cryptography) [65, 75] require intense computations and thus are undesirable for low-power, resource-constrained platforms. As such, research has been conducted into new lightweight and low overhead techniques for addressing hardware weaknesses [45, 55, 79] as early as possible at design cycle [12, 23]. ***Physical Unclonable Functions*** **(PUFs)** [35] are one security primitive, which has been proposed in this space.

1.1 Physical Unclonable Function Overview

PUFs are security primitives that derive entropy from low level manufacturing process variation in physical components. PUFs are good candidates to address the security problems of resource-constrained computing platforms, especially in regard to device **identification (ID)** and authentication. PUF signatures are produced by a challenge-response protocol: a unique response (output) is generated for a specific challenge (input). The form of the challenge and response is dependent on the design and desired properties. The main target of a PUF design is to generate sufficient entropy that each instance gives a unique response, while simultaneously ensuring the responses are reliable—within a range of acceptable limits—under different environmental conditions [67].

Typically a PUF is implemented on a discrete chip, added as a component to a complex circuit, or implemented on an FPGA. Such PUFs form the vast majority of PUF literature to date. The most well studied designs are the arbiter PUF [31, 44], which generates entropy from the difference in two identically placed delay paths with an arbiter at the end, and the **Ring Oscillator (RO)** PUF [47, 67], which uses the difference in frequency between two nominally identical oscillators to generate the response. Such studies have introduced designs with desirable security properties but that are not always suitable for use in the context of resource-constrained and pre-existing devices due to the need to make circuit design changes and add hardware components to carry out the PUF function.

1.2 Software PUFs

A potential solution to these issues are software PUFs, the entropy of which is extracted from circuitry already present in a system without modification and purely by means of software. These PUFs have the advantage of having no hardware overhead and requiring no design changes to the device hardware. In addition, as they are software-based, it is often possible to deploy such PUFs onto devices that are already in use. Several such designs have been proposed [38, 43, 44, 48, 63]. By their nature, each software PUF design relies on the characteristics of an existing component of hardware in the system. Hence, to allow these PUFs to be usable on a broad range of devices it is necessary to have viable designs using as wide a range of underlying hardware as possible.

The primary example of a software PUF is the SRAM PUF [38], in which the PUF response is formed from the power-on values of cells in SRAM memory modules. Although effective, SRAM PUFs require power cycling to access the PUF response, which is not practical in all systems. Designs for software memory PUFs based on other memory technologies, such as DRAM and NAND flash memory, have also been proposed [10, 25, 49, 61] though like the SRAM PUF specific conditions are required for the PUF to be viable in a given system. For instance, not every electronic device is equipped with DRAM and flash memories.

In an attempt to widen the body of devices that can viably use a software PUF, the concept of PUFs deriving entropy from a processor itself has been explored [43, 44, 48, 63]. Such designs leverage processor delays, which are prone to variations, to derive the PUF challenge-response mechanism. Although effective, and in some cases relatively lightweight, the majority of these designs are not software PUFs and require additional hardware to achieve the PUF challenge response mechanism [43, 44, 63]. The only existing software processor PUF [48] may generate responses without the need of extra circuitry, but it requires onboard precise high-speed pulse generators. Particularly, it requires the characterization of each instruction for various clock reduction levels to achieve a sufficient number of response bits. This complicates the challenge-response mechanism, which consumes a significant amount of time/power to produce unique outputs.

1.3 Contributions and Outline

In this article, we introduce DTA-PUF,[1] a software PUF design that achieves unique and reliable responses with a minimal power overhead compared to existing designs. By leveraging timing errors of structures inherent to microprocessors instead of building additional circuit/logic, we propose a low-power PUF architecture. The basic principle of the proposed DTA-PUF lies in the systematic exploitation of the dynamic timing behaviour of logic, which has never been fully exploited, to minimise processing effort. The proposed PUF design is derived from ubiquitous circuitry and hence can be employed to many platforms—especially to those requiring low-power computations such as IoT devices. The main contributions of our work can be summarized as follows:

- We develop a novel, processor-based, software PUF design leveraging the instruction- and data-dependent timing behaviour of pipelined cores. DTA-PUF derives entropy from the inherent complex manifestation of timing errors under carefully selected overclocked settings. Revealing such properties require pre-fabrication simulations, which we perform at one of the most accurate phases before the actual manufacturing, i.e., at post-layout timing analysis phase.
- We implement a design flow using commercial tools that allow us to systematically select instruction sequences that maximize timing error rates in any target pipelined core. DTA-PUF requires considerably smaller size of input challenge than comparable designs to provide the challenge-respone mechanism, limiting required processing efforts and leading to power savings.
- We demonstrate the concept on a pipelined, out-of-order, IEEE-754 compliant [1] **floating-point unit (FPU)** implemented in 45 nm process technology. The generated PUF responses are evaluated with regard to several statistical quality metrics including uniqueness, min-entropy and reliability. Our results show high values of uniqueness, min-entropy and reliability among the evaluated chips.

The article is organized as follows: Section 2 outlines background information and limitations of existing processor-based PUFs. Section 3 introduces the proposed PUF design, while Section 4 discusses the implementation of the proposed DTA-PUF. Section 5 presents the experimental results and Section 6 discusses related work. Conclusions are drawn in Section 7.

2 BACKGROUND AND MOTIVATION

In this section, we provide background information and discuss the most common processor-based PUF designs, analysing the challenges that motivate our work.

[1]DTA-PUF: **D**ynamic **T**iming-**a**ware **P**hysical **U**nclonable **F**unction.

2.1 PUF Designs - Background

PUFs are typically characterised by a set of input challenges and corresponding output responses. These are referred to as **Challenge-Response Pairs (CRPs)**, the form of which vary depending on the PUF design. It is generally assumed for a PUF design being used for hardware ID that some agent, either the system provider or the user, will initially characterise the PUF to derive the full set of CRPs. It is further assumed that this information will be stored somewhere separate from the PUF. This is referred to as "enrolment." When the system needs to be verified a given challenge will be sent to the PUF and the response will be compared to the expected value. This is referred to as "query."

Depending on the number of CRPs that can be generated from a single device, PUFs can be distinguished into weak and strong PUFs:

Strong PUFs are those PUFs that have a very large CRP space. Ideally, a PUF classed as such should have a CRP space that grows exponentially with the resources dedicated to the PUF. PUFs that have only a linearly increasing CRP space are not typically classed as strong PUFs. Strong PUFs are the more well studied class of PUF, but have seen minimal adoption. This is in part due to the proven vulnerability of many strong PUFs to modelling via **machine learning (ML)** [28, 60], though in recent years proposals to mitigate this issue have been made [50].

Weak PUFs are those PUFs that have a relatively small CRP space—in many cases only a single CRP as in the case of SRAM PUFs [38]. Despite being labelled as "weak," these PUFs have seen much more usage in practice as they generally have quite a high cost for an adversary to model [20, 59]. Typically, weak PUFs are jointly considered with cryptographic methods [74] (e.g., encryption) to compensate for CRP scarcity.

2.2 Variability in Nanometer Circuits

Several PUFs leverage intrinsic timing variations of circuits to provide secure protocols [43, 44, 48, 63]. In fact, the microelectronic substrate, on which modern circuits are built, is increasingly prone to variability. Most profound is the variation in circuit parameters of the manufactured chips (1) within die, (2) die-to-die, and (3) over time.

Within-die (intra-die) variations account for the the variations that arise between different devices and interconnects that reside within the same chip. There are different sources of these variations (e.g., process, voltage, temperature, and environmental factors) and may result up to 50% frequency fluctuations [34].

Die-to-die (inter-die) variations refer to the variations that arise between different chips in the same wafer or different wafers. Die-to-die variations are mostly design independent and are mainly related to equipment properties (e.g., wafer placement, manufacturing lithography).

Over time variations. Wires and transistors in integrated circuits suffer substantial wear-out, leading to power and performance changes over time of usage. One of the main sources of such dynamic variations is the phenomenon of device aging, which became much more troublesome for the sub-45 nm nodes [7].

Such variations affect the point of first failure [17] and thus the safe clock frequency between different chips or different cores within the same chip. To mitigate such phenomena, designers apply a global frequency constraint that forces all the manufactured chips to operate at different frequency groups [57] depending on the minimum frequency achieved in their cores. In Section 5.2, we use this performance or speed grouping of chips to represent delay variations effects.

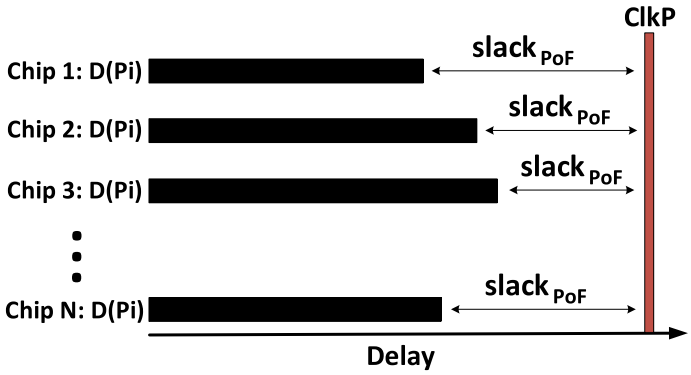

Fig. 1. Delay and point of failure (PoF) variations of the same timing path across variable chips.

2.3 Processor-based PUFs

Those entirely software-based, processor PUFs proposed to date accept instructions as input challenge and take advantage of the fact that the point of failure of a given instruction and path delays vary from chip to chip to generate the output response.

To better illustrate how these PUFs work, let us assume a synchronous processor consisting of a set of N combinatorial paths $P = \{P_1, P_2, \ldots, P_N\}$, which are characterized by their delays $D(P_i)$[2] for $i = 1, 2, \ldots, N$. As in any synchronous design, the conventional **static timing analysis (STA)** evaluates the longest timing path across all S pipeline stages and determines the the clock period ($ClkP$) at design time, such as

$$ClkP \geqslant \max_{s=1\ldots S} \left\{ \max_{p \in P^s} \{D(p)\} \right\} = \max_{p \in P} \{D(p)\}, \tag{1}$$

where P^s is the set of paths of pipeline stage s ($s = 1, 2, \ldots, S$). During circuit operation, the executed instruction activates a path P_i that has a positive timing slack, $slack_{PoF} = ClkP - D(P_i)$, until the **point of failure (PoF)**. Under any clock period reduction, also known as overclocking, more than $slack_{PoF}$, the activated path P_i will fail, since $D(P_i) > ClkP$, leading to a setup timing error [13]. Figure 1 provides an example where due to variations the same path P_i has variable delay ($D(P_i)$) and PoF ($slack_{PoF}$) in different chips. Based on this figure, a processor PUF takes an instruction I_i as input challenge ch_i, which sensitizes P_i, and generates an output response res_i exploiting variations in $D(P_i)$ and $slack_{PoF}$.

2.3.1 Non-software Processor-based PUFs. The majority of the existing processor PUFs extract entropy from path delay variations by introducing new circuitry to the processor for the purposes of response generation and extraction. In some cases, this is simply the addition of arbiters to measure existing delay paths [44], in others custom cores [8], and in others signal generation circuitry [5, 78, 81]. Such approaches require design changes to already complex circuits and incur a cost in terms of hardware resources, power budget, or both. These drawbacks limit their adoption in industry, especially in low-power platforms. A full discussion of the various processor-derived designs proposed to date can be found in Section 6.

2.3.2 Software Processor-based PUFs. In an effort to trim down the overheads induced by non-software PUFs, a software processor PUF [48] has been proposed. To the best knowledge of the

[2] $D(P_i)$ also considers the clock-to-output delay and the setup time of a register [13].

authors this is the only fully software-based, processor-derived PUF design proposed to date. In this design, the response bits are generated by exploiting the fact that a given instruction fails under different frequency points across different chips. As such, it requires no extra circuitry or design changes to extract responses so long as the clock frequency can be controlled to induce the necessary instruction failures. Although effective, such an approach requires precise overclocking to a wide range of values and a high number of repeated instruction calls to generate just 2 bits of a response. Further, the overall CRP space is limited by the resolution of the clock. Each distinct clock period can generate only 2 bits per unique instruction, requiring a considerable number of instructions executed at different overclocking levels to generate an adequate output response (e.g., 128-bit response). While subsequent studies have tried to address these issues via added custom instruction logic [8], we demonstrate in this work that it is possible to generate PUF responses quickly and with much lower power consumption, while retaining a fully software-based design. This is achieved by targeting pipeline cores and through the careful selection of instruction sequences as input challenges.

2.4 Dynamic Timing Behaviour and Instruction Execution History

The existing software processor PUF [48] exploits the data-dependent path activation [30] to generate unique responses. However, it neglects the impact of instruction execution history (i.e., type and order of instructions within a pipeline at any instant) on the dynamic timing behaviour of computational paths. Instructions that are executed concurrently (i.e., they share the same hardware circuitry in a time-sharing fashion) may affect the possibility of timing errors, because these instructions share control signals and execution stages, affecting the state of the forwarding logic, and thereby place great demand on circuit timing deadlines [69, 70]. In a previous study [37], it was shown that **instruction sequences (ISQs)** have a significant impact on timing error rates, but they have not indicated how many instructions within a sequence affect this dynamic timing behaviour. Intuitively, all those instructions that precede an instruction in the pipeline may have an effect on the timing error behaviour of this instruction. To investigate this, we extract real floating-point instructions executing on an ARM A7 board. To achieve this, we extend an open-source profiling tool [52] and extract a trace of 1M floating-point ISQs from the *bt* program of NAS benchmark suite [11]. Then, we run post-layout gate-level simulation (see Section 4.2) based on this trace of an FPU, the details of which will be discussed later (see Section 5.1). We simulate this trace in windows of increasing number of concurrently executed instructions under 15% clock reduction. Specifically, we start simulation with a window size of 1 instruction and increase it up to the pipeline depth, which is six stages in the FPU under test. Each experiment records the timing **error rate (ER)**, defined as

$$ER = \frac{Faulty\ ISQs}{Total\ ISQs}. \tag{2}$$

Figure 2 indicates that sequences consisting of six instructions have exactly the same ER when running the full trace (full history) through simulation. By contrast, a window size of one instruction leads to ∼46.6% lower ER when compared to the six-instruction window. From these findings, we conclude that all the pipelined instructions preceding the currently executed instruction may trigger a timing error in that instruction.

Taking into consideration the above, we propose DTA-PUF: a processor PUF design that overcomes the limited response bit generation and minimises processing/power overheads by fully exploiting all the factors that affect error rate in pipelined units. The design is described in detail in the following section.

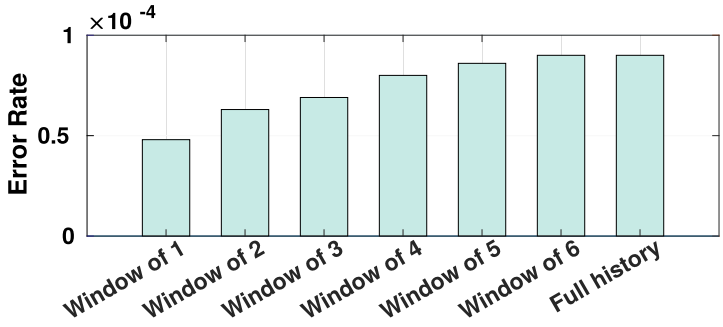

Fig. 2. Impact of instruction execution history on errors.

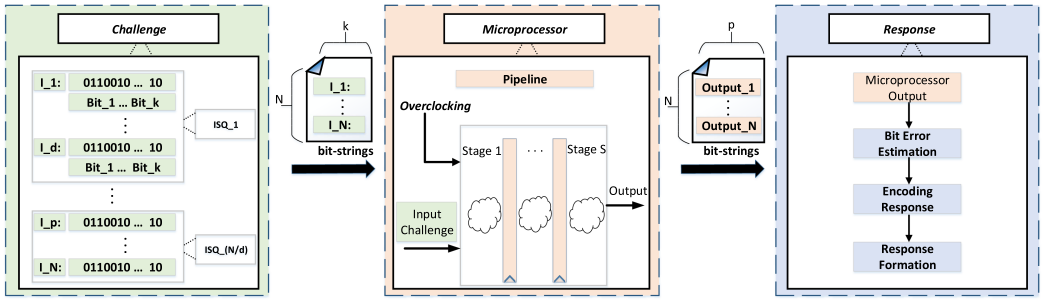

Fig. 3. Challenge response mechanism of DTA-PUF.

3 PROPOSED SOFTWARE PUF

DTA-PUF is a novel software PUF that extracts entropy with minimal resource usage by fully exploiting the dynamic timing behaviour of microprocessor circuits. DTA-PUF aims to jointly consider the inherent complexity of timing error manifestation with the intrinsic chip variations, proposing a new approach that inseparably intertwines PUF performance with dynamic error behaviour of pipelined cores. Our PUF implementation and evaluation rely on detailed post-layout gate-level simulation, which is one of the most accurate, pre-fabrication steps of the standard ASIC flow used in industry [58]. Figure 3 shows the proposed PUF design for generating the challenge-response mechanism.

3.1 Challenge Procedure

In this work, the input challenge consists of a set of N instructions $I = \{I\_1, \ldots, I\_N\}$. An instruction is composed of a string of k bits. It is important to note that these instructions are carefully selected to activate timing critical paths that maximise timing errors under a potential clock reduction. To this end, we use microarchitecture-aware timing information extracted at the design cycle (see Design Phase of Figure 4). As already explained, instruction execution history plays an important role in timing error manifestation. To create an execution history-aware challenge, while considering the data-dependent error occurrence, we split I into sequences ISQ_n consisting of d instructions such that $\cup_{n=1}^{N/d} ISQ\_n = I$.

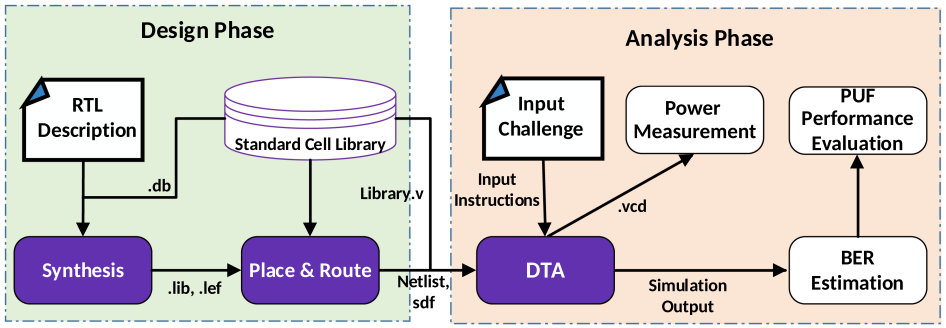

Fig. 4. Design and analysis phases of DTA-PUF.

3.2 Microprocessor Operation

The generated challenge is then given as an input to the target microprocessor. This PUF accepts I as a challenge and produces the count of the bit errors in the output of the microprocessor as the response. Since I consists of N instructions (or N/d ISQs), the microprocessor outputs a string consisting of $N \times p$ bits, where p depends on the architecture of the design under test. Timing errors are captured by operating the microprocessor pipeline at a reduced clock period, known as overclocking. We refer to the magnitude of this clock reduction as ΔT. Note that ΔT should not exceed a certain clock period beyond which the microprocessor fails completely (we refer to as T_{fail}). In particular, ΔT should be between PoF and T_{fail} such that $T_{fail} > \Delta T \geqslant PoF$. It is important to mention that such an overclocking technique is applied to target microprocessor when it operates as a PUF; during the normal operation, the nominal clock period is used (see Section 3.4).

3.3 PUF Response

The response is extracted from the output of the ISQs by counting the bit errors of the final instruction in each sequence. While in some cases multiple erroneous instructions occur within a sequence, we observed that the greatest entropy was achieved only in the final instruction of a sequence of d instructions. Additionally, we observed that some bits had near zero entropy, and thus were not useful for constructing the response. This is explained in further detail in Section 5. The number of errors is encoded as a 6-bit binary value (the amount of bits needed to represent the possible error values of the 64-bit output minus the excluded bits). These values are concatenated to create a response of the desired length. For example, assume an n =18-bit response is required. Three ISQs are used that produce 1-, 9-, and 7-bit errors, respectively. The corresponding binary responses would be 000001, 001001, and 000111. The resultant 18-bit response is the concatenation of these bit strings, i.e., 000001001001000111.

3.4 Microarchitectural Support

The target microprocessor supports two modes: (i) the normal, fully accurate mode where a set of instructions executed in the pipeline at the nominal clock period, and (ii) the PUF mode where the microprocessor operates at a reduced clock period. To this end, we extend the instruction set of microprocessor by two special instructions `PUFstart` and `PUFstop`. `PUFstart` indicates that the microprocessor will explicitly operate as a PUF (PUF mode). Once the PUF response has been generated, the `PUFstop` instruction forwards the PUF response and switches the microprocessor to the normal operation mode. Since the PUF mode disrupts the microprocessor normal operation,

there is an execution time overhead on programs executed in normal mode. However, as we will explain in Section 5.6, DTA-PUF's high bit generation speed limits such overhead.

4 IMPLEMENTATION WORKFLOW

The implementation workflow of DTA-PUF is depicted in Figure 4. The design phase is being executed only once, while the analysis phase runs for each challenge.

4.1 Design Phase

The first step of this phase is the Synthesis, which is followed by the Place and Route steps. These steps are performed utilizing optimizations that aim to achieve maximum performance. The design phase outputs the following files:

(i) A library verilog file, which specifies the logic and the rise and fall times of the standard cells.
(ii) A gate-level netlist, which is stored in a verilog format (.v). This file consists of a list of the electronic components in the circuit and a list of nodes they are connected to.
(iii) A **standard delay format (SDF)** file, which describes the cell and interconnect delay. This file is obtained at the Place and Route step.

Design phase also generates a timing report that includes delays of the timing paths. This helps to create an input challenge set (see Section 3.1) that activates timing critical paths, leading to an increased number of timing errors at a reduced clock period.

4.2 Analysis Phase

Dynamic Timing Analysis (DTA). To enable characterization of the data-dependent path activation, we perform DTA using post-layout gate-level simulation. DTA identifies the actual timing margins of the target core at runtime by including path activation information (instruction type, operand values, pipeline sequence) that is unavailable during static timing analysis. The DTA tool uses as input the generated challenge (see Section 3), the outputs of the design phase and the set clock period. Providing that every set of instructions under nominal conditions and clock period produces an error-free output, we define this simulation output as O_{gold}. We estimate the number of timing failures by comparing O_{gold} with the simulation output under the reduced clock period. Timing errors in the final instruction of each ISQ are then used to form an n-bit PUF response. Based on the output response, we utilize three quality metrics to evaluate the DTA-PUF performance: uniqueness, min-entropy and reliability.

Such an analysis phase also helps to extract a **value change dump (VCD)** file essential for power analysis. This file contains information about the switching activity and value changes occurred during the simulation.

5 EVALUATION RESULTS

In this section, we first present our experimental set-up. Second, we evaluate the efficacy of DTA-PUF for generating unique and reliable responses. Third, we estimate any possible power gains by exploiting the high bit generation speed of DTA-PUF compared to an existing processor-based software PUF design. Then, we present several use cases of DTA-PUF and last, we discuss our PUF vulnerability to potential attacks.

5.1 Experimental Set-up

For the case study of using DTA-PUF, we focus on arithmetic, floating-point operations, since those are more prone to timing errors, as also reported by existing studies [15, 40, 48, 72].

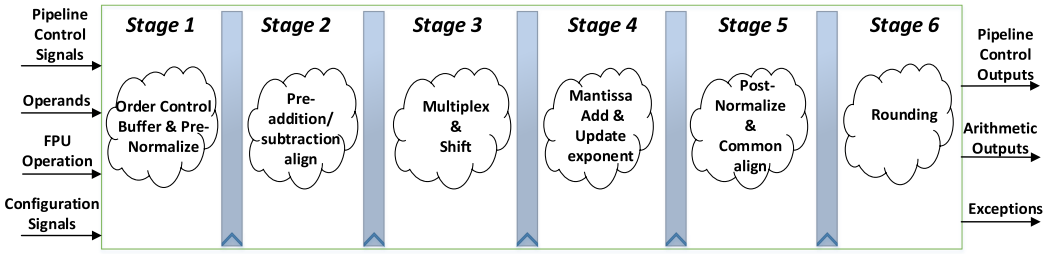

Fig. 5. Microarchitecture of the floating-point additions/subtraction related stages.

We apply our approach to a six-stage pipelined, out-of-order, IEEE-754 compatible [1] FPU that supports double precision operations. According to the IEEE-754 Standard, a floating-point number follows the representation $-1^S \times M \times 2^E$, where S: sign, E: exponent, and M: mantissa. In a double precision FP number the **most significant bit (MSB)** indicates the sign, the next 11 bits represent the exponent and the mantissa consists of the last 52 bits. This unit is part of the mor1kx MAROCCHINO pipeline, which is a single-core processor based on the OpenRISC-1000 instruction set architecture [4]. The FPU supports the following floating-point instructions: multiplication, division, addition and subtraction, integer-to-float and float-to-integer conversions. Figure 5 illustrates the microarchitecture of the targeted FPU, highlighting the floating-point addition/subtraction. At Stage 1, an Order Control Buffer and a Pre-Normalize block are implemented, which permits data dependencies detection and adjustment of the exponent and mantissa, respectively. Stage 2 is responsible for the pre-addition/subtraction alignment, while Stage 3 performs the necessary multiplexing and shifting of the operands. Mantissa addition and exponent update are performed at Stage 4; rounding occurs in the last two stages.

Based on OpenRISC 1000 instruction set architecture and IEEE-754 Standard, the input challenge of the PUF consists of N instructions from $k = 140$ bits each. We simulate these instruction in sequences consisting of $d = 6$ instructions, where d corresponds to the pipeline-depth of the FPU under test and thus to the maximum number of concurrently executed instructions in the pipeline. While simulation output of the FPU is composed of $p \times N$ bits, where $p = 64$. N strongly depends on the number of the bits in PUF response (see Section 5.5).

This FPU design is implemented using the typical corner of the CCS NanGate 45 nm library (@1.1 V) [2]. For hardware Synthesis and Place and Route, we use the Design Compiler (version: N-2017.09-SP3) from Synopsys and Innovus (version: v16.13-s0451) from Cadence, respectively. For the power measurements, we invoke Voltus (version 16.2) from Cadence. DTA is performed using detailed post-layout gate-level simulation supported by ModelSim (version 10.7c) from Mentor Graphics. The maximum clock frequency achieved is 425 MHz, i.e., $ClkP = 2.35$ ns. For these experiments, we capture timing error by reducing $ClkP$ by $\Delta T = 30\%$. We refer to the reduced clock period as $ClkP_{red} = 1.65$ ns.

5.2 Representing Delay Variability of Integrated Circuits

Due to increased static and dynamic variations of nanometer circuits, the delay of chips after fabrication will be different (see Section 2). In fact, such a delay uncertainty is manifested in the form of core-to-core and chip-to-chip frequency variations. As explained in Equation (1), the clock frequency/period can be modeled as the maximum delay of all the timing paths in the chip under test. For example, in the Intel's 65 nm 80-core chip (@1.2 V) [34], the maximum core frequency is achieved at 7.3 GHz, whereas the minimum one is 5.7 GHz. Such a frequency fluctuation has been also observed between different chips [14, 21, 34, 41]. The most common technique to model

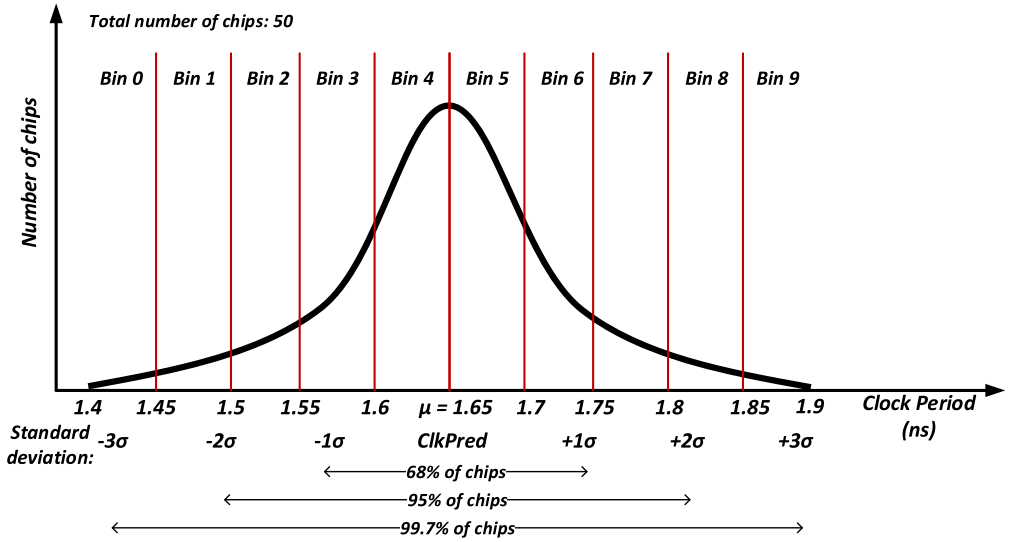

Fig. 6. Speed binning. Random/Gaussian distribution of clock period across the considered chips.

this, is the so-called speed or frequency binning [19, 62], where different chips (that implement the same functionality) fall into separate speed/frequency bins.

Therefore, we sort the considered chips within 10 clock frequency bins depicted in Figure 6. Following [16, 44, 77], we also assume that clock period variations in different chips follows a random/Gaussian distribution $N(\mu, \sigma^2)$ with $\sigma = 0.08$ ns and $\mu = ClkP_{red} = 1.65$ ns. We vary the clock period between -3σ to $+3\sigma$ from $ClkP_{red}$ and the clock period of each chip is estimated as follows:

$$Clk\_C_m = ClkP_{red} \pm Tvar_m,$$

where m ($m \in [1, 50]$) denotes the total number of considered chips and $Tvar_m$ ($Tvar_m \in [-3\sigma, +3\sigma]$) the deviation of Clk_C_m from $ClkP_{red}$. In other words, this clock stretching technique represents as a single factor the overall influence of potential worst-case delay variations between chips. Additionally, the levels of variation-induced delay fluctuations (i.e., σ and $Tvar_m$) used in these experiments are consistent with what have been reported in literature [14, 34, 43, 63]. It is important to note that such a technique does not provide a perfectly accurate representation of real hardware; however, it sufficiently reflects the actual timing behaviour of circuits after fabrication as recently indicated [40, 73]. Nonetheless, such a delay representation is based on detailed post-layout gate-level simulation, which is among the final steps of the typical ASIC design flow used in industry [58]. In general, in the IoT regime where million of devices are connected to the Internet, there is a need to address hardware security challenges and validate the effectiveness of security protocols at design cycle, i.e., before the actual manufacturing of the chips.

Overall, we perform 50 simulations, representing $m = 50$ different chips. Based on Gaussian distribution properties, we randomly select 34 chips (68% of total chips) from bin 3 to bin 6 and 16 chips from the rest of the bins. The plots in Figures 7, 8, and 9 reflect this setting.

5.3 Uniqueness

In this subsection, we evaluate the effectiveness of our PUF in generating unique response to a given challenge in any two randomly selected chips in the overall population.

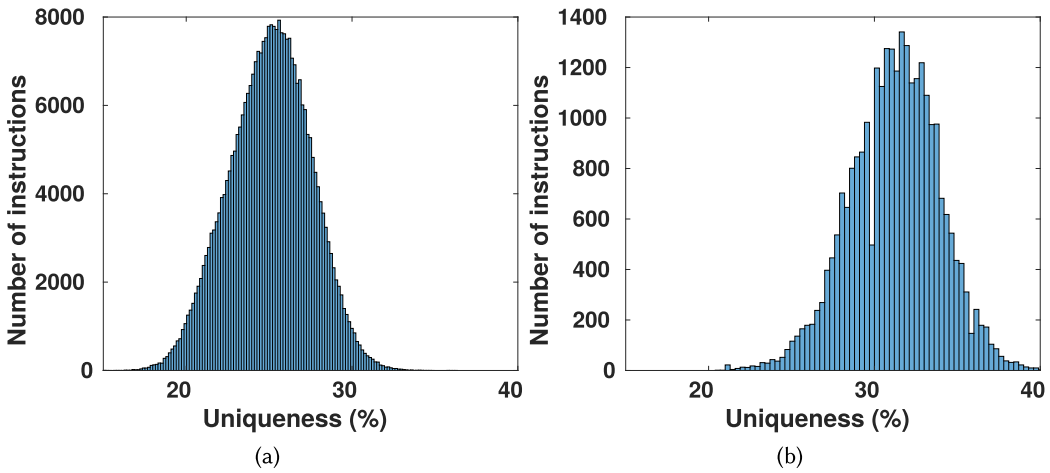

Fig. 7. Uniqueness distribution of the analysed CRPs using (a) 9M instructions with no post-processing and (b) best-case selection of 90K instructions with minimal processing effort.

Uniqueness is calculated here by comparing the response of each chip to every other chip in the population as shown below, where m is the number of PUFs in the population, Rn is the response of a given PUF instance, and n is the number of response bits:

$$U = \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \frac{HD(R_i, R_j)}{n} \times 100\%. \tag{3}$$

Ideally, the uniqueness of all the possible CRPs will be a normal distribution centred around 50%. As can be seen in Figure 7, the median uniqueness of this design is in the order of 26%. Note that we use $|I| = 9$M or 1.5M ISQs as input challenge across $m = 50$ chips/simulations to extract the uniqueness distribution depicted in Figure 7(a). Some ISQs show higher uniqueness than others across the entire population of devices and as such it is possible to be more selective when choosing which ISQs will be used as CRPs to improve the uniqueness value at the cost of reducing the CRP space. When such a technique is applied to select the best 90K rather than 9M instructions as in the full experiments, the average uniqueness increases to 27.9%. Additionally, if certain bits are excluded from the raw response and the resultant error rate encoded as a 6-bit response along with the previous technique, then the average uniqueness further improves to 31.4%; and it is likely that with further development this could be improved at the cost of increased complexity. The derivation of this 6-bit response is thoroughly explained in Section 5.4 below.

5.4 Min-entropy

Min-entropy (H_{min}) measures the worst-case entropy of the design, i.e., the minimum difference that will be seen between any two instances in the worst-case. The ideal value of 100% will be measured in a design in which any given bit being equal to 0 or 1 is equal probable. Min-entropy is calculated as follows for an n-bit response measured across m devices, where Pb_{max} denotes the maximum bit probability:

$$Pb_{max} = \begin{cases} \frac{HW_b}{m} & HW_b > \frac{m}{2} \\ 1 - \frac{HW_b}{m} & HW_b \leq \frac{m}{2} \end{cases}, \tag{4}$$

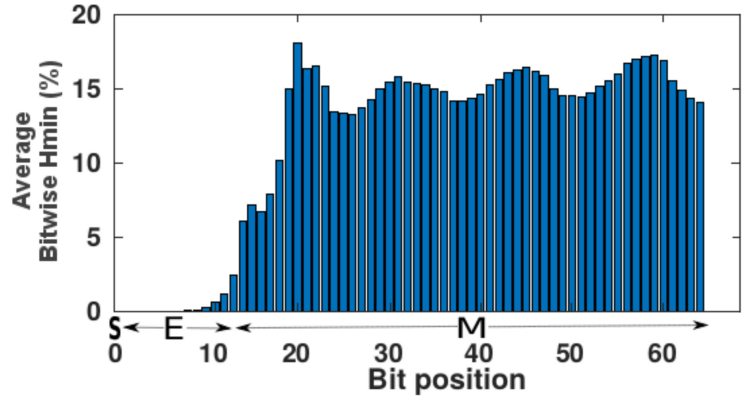

Fig. 8. Distribution of Min-entropy across output bits. X axis depicts the sign (S) bit, the exponent (E) bits, and the mantissa (M) bits.

$$H_{min}b = -log_2(Pb_{max}), \tag{5}$$

$$H_{min} = \frac{1}{n}\sum_{b=1}^{n} H_{min}b. \tag{6}$$

An initial evaluation of H_{min} was performed using the raw 64-bit output of every instruction in the instruction set for all devices, i.e., for $n = 64$ and $m = 50$, to evaluate if the raw output itself was usable as a PUF response. The encoding of bits into PUF responses in this case was kept as simple as possible, with the each bit of the response being a 0 if the bit was the expected output and a 1 if the bit was erroneous. The average H_{min} measured across this set of instructions was 11.68%, which is very low. However, it was observed that there were distinct trends, when the bitwise Min-entropy ($H_{min}b$) was averaged across the instruction set.

First, the sign and exponent bits had near zero min entropy. This matches with what would be expected as the uppermost bits of a pipelined core are in general the least prone to error, meaning that these bits across almost all instructions were the expected value and hence produced a 0 in the PUF response. Further, even within the mantissa bits the first 10 bits were much lower in terms of average $H_{min}b$; and while after this point the values remain relatively high, there are still distinct peaks and troughs. This can be clearly seen in Figure 8. From this, it can be implied that it is beneficial to exclude specific bits from the output when encoding the response as these bits will very rarely generate any errors, and hence will produce near identical response bits even across a relatively large number of challenges such as the 9M instructions evaluated here. Consequently, we select to filter out the low entropy bits and compare only the output of the bits that are likely to generate errors when forming the response. Specifically, the worst 33 bits in terms of $H_{min}b$ were excluded and the error rate of the remaining 31-bit output was encoded using 6 bits. Even then, the average $H_{min}b$ is still only in the order of 15%, however, this can be further improved by encoding the bit error rate as the response rather than the raw output, in which case the overall average H_{min} increases to 25.78%. This combined with more selective choice of ISQs when forming challenges can also increase the uniqueness of the design to 31.4%, as can be seen in Figure 7(b).

Each of these improvements in H_{min} and uniqueness come at the cost of lowering the amount of bits generated per ISQ. With only the simplest post processing, 64 bits are generated per ISQ, while in the final scheme this is reduced to 6. However, even with this reduction in throughput, the

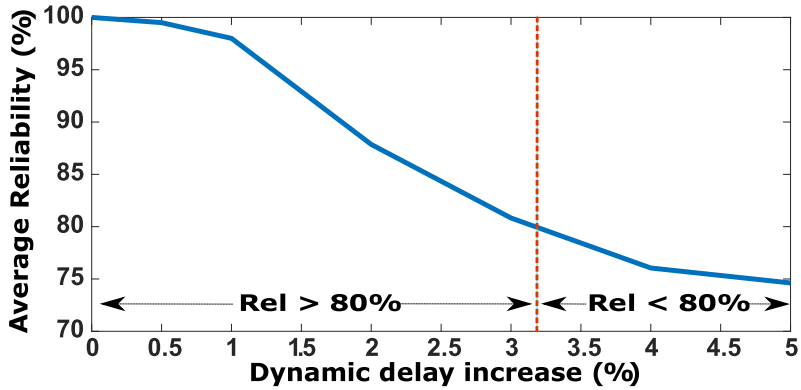

Fig. 9. Average reliability of response generation across different levels of environmental induced delay increase.

number of instructions required in comparison to existing fully software-based, processor-derived PUFs is still greatly reduced with a corresponding reduction in the power consumption. This is explained in Section 5.6.

5.5 Reliability

As they exploit low level variations in the circuit fabrication, PUFs are often vulnerable to influence from environmental factors such as transistor aging or supply voltage fluctuations [7]. Reliability measures the amount by which the response of a given chip will vary under non-nominal environmental conditions. Reliability has been calculated by comparing a set of responses measured under varying conditions to a reference response, where m is the number of measurements taken, $Rref$ is the reference response, Ri is the response under varying environmental conditions, and n is the number of response bits:

$$Rel = \frac{1}{m}\sum_{i=1}^{m}\frac{HD(R_{ref}, R_i)}{n} \times 100\%. \tag{7}$$

As our evaluation results are based on an accurate, post-layout gate-level simulation (see Figure 4), we have added an additional delay increase on top of the static delay variance, reflecting dynamic/over-time variations of the same chip. To provide a robust analysis of the likely PUF performance, we have measured the average reliability under increasingly high, environmentally induced variation up to 5% [43, 63].

As depicted in Figure 9, even under relatively large amounts of environmentally induced variance, the average reliability does not drop below 74%. Further, so long as this variance can be controlled such that it does not exceed 3.2%, the average reliability will remain above 80%. This is insufficient for use as input to a cryptographic algorithm, but it suffices for the purpose of chip ID or IoT node ID. The three primary sources of such dynamic variance in this instance are temperature, supply voltage, and transistor-aging. In regards to temperature and supply voltage, we can mitigate the influence of these factors by enrolling the PUF initially at a variety of temperature-voltage combinations [56]. The current supply voltage and temperature can then be passed as supplementary data with the response, such that the PUF response can be compared with the expected response for the temperature and voltage that is closest to the current conditions. To account for transistor aging, it is possible to completely or partially re-enrol the PUF after a set period or to provide low cost, aging-aware mechanisms [7]. Re-enrolling our PUF induces an

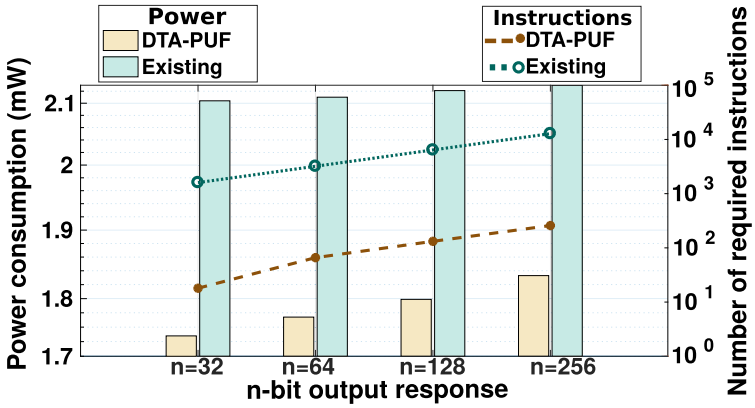

Fig. 10. Power consumption and required number of instructions across DTA-PUF and the existing software processor PUF [48] under different response bits.

execution time overhead on the executed application, but as we explain in Section 5.6, the high-speed response generation keeps this overhead small. The circuit level, aging-aware technique presented in Reference [7] explores approximate computing principles [51, 80] in the context of aging. Thus, it comes with an output quality loss that may be accepted by many applications [24, 33]. Specific methods to improve the reliability of this design are planned for future work, but are out of scope for this article.

5.6 Power Consumption, Execution Time, and Output Response

Using the simple concept explained in Section 3.3, we generate 6 response bits per each ISQ (i.e., 6 bits per 6 instructions) in the input challenge. Therefore, the number of the instructions needed for an n-bit response is: $\lceil n/6 \rceil \cdot 6$. For example, we will need an input challenge consisting of 132 instructions (or 22 ISQs) to generate an 128-bit response. Conversely, the existing fully software-based processor PUF [48] (we refer to as soft-PUF) uses 12.8k instructions as challenge to generate the 128-bit response. Using the analysis phase of Figure 4 and the extracted VCD files (see Section 4), we estimate the dynamic power consumption under different number of response bits. Figure 10 compares the power consumption incurred by our PUF with the one obtained by soft-PUF across increasing number of response bits. The number of the instructions that DTA-PUF and soft-PUF require for generating responses is also depicted in the right y axis of the same figure. Note that for the sake of this comparison, we simulate the target FPU using different number of instructions required for DTA-PUF and soft-PUF to generate the n-bit response. As shown in Figure 10, DTA-PUF requires up to 50× less instructions than soft-PUF to generate output responses. Such a property, allows DTA-PUF to save up to 26% power when compared with soft-PUF.

As we discussed, DTA-PUF interrupts normal microprocessor operation and uses the target design only as a PUF. This imposes an execution time penalty on the executed applications, which strongly depends on the number of instructions required to generate an n-bit PUF response. In fact, this penalty will be equal to the execution time of the PUF query procedure. To better understand the involved overhead, we define the execution time (ExT) of any PUF running at $ClkP_{red}$ as

$$ExT = \#cycles \times ClkP_{red} = (\#instructions + pipeline\_depth) \times ClkP_{red}. \quad (8)$$

As explained, DTA-PUF requires 132 instructions to generate an 128-bit response leading to $ExT_{DTA-PUF} = 138 \times ClkP_{red}$. When compared to the execution time overhead incurred by

soft-PUF for generating the 128-bit response ($ExT_{soft-PUF} = 12806 \times ClkP_{red}$), DTA-PUF reduces the execution time penalty by ∼92.8×.

5.7 Potential Use Cases

As shown, DTA-PUF achieves a low-power security primitive without needing any hardware addition. Such a PUF can be used in any (micro)processor to provide a secure challenge-response mechanism. Additional use cases of DTA-PUF are discussed next.

5.7.1 IoT Device Identification and Authentication. Lightweight and low cost authentication and identification are very important security aspects for IoT devices, because they often cannot afford resource-demanding cryptographic protocols. DTA-PUF is a fully software-derived (i.e., no need for extra circuitry or design changes), low-power solution to secure IoT. Further, DTA-PUF proposes a simple and fast challenge-response mechanism that generates relatively large output responses, while limiting the processing effort and size of input challenges. Therefore, DTA-PUF could be extremely beneficial for low-power device ID and authentication in IoT nodes or in any other system that lacks computational resources [29].

5.7.2 DTA-PUF and Approximate Computing. DTA-PUF leverages the inherent complex manifestation of dynamic timing errors to provide a lightweight secure protocol. In our design, timing errors are induced by overclocking the design under test. Increasing the clock frequency (i.e., reducing the clock period) [18, 66] or decreasing the supply voltage [42, 71] are common techniques that the approximate computing paradigm [51, 80] exploits to significantly reduce the energy consumption of a system. Frequency and voltage scaling may increase the energy savings, but as we showcase in this article, it comes with the cost of timing errors. The quality loss incurred by these errors [41, 71, 73] may be tolerated by inherent resilient algorithms such as signal/image processing, machine learning and scientific computation [24, 33]. In this space, there are design methodologies that opportunistically ignore timing errors in memory units [26] and CPUs [32, 66], trading off energy, output quality and error resiliency.

Hence, DTA-PUF could be jointly considered with approximate computing methods [46] to offer resource and power/energy savings, while sacrificing quality in applications that are amenable to approximations and can tolerate inaccurate results. In such a system, the PUF responses could be extracted from the error prone least significant bits of approximate computations. As shown in Section 5.4, the bits in which error can be tolerated in approximate calculations are also the bits from which the best PUF response can be extracted. A scheme making proper use of this principle could in effect generate PUF responses while incurring near zero power and execution time costs relative to normal operation.

Finally, modern platforms allow operations beyond the nominal voltage/frequency values for improving energy efficiency [9, 53, 54, 68]. This makes it possible to integrate DTA-PUF in these platforms without requiring any extra mechanism (e.g., clock generator, voltage regulator) to dynamically adjust the voltage or frequency margins.

5.8 Discussion on DTA-PUF Vulnerability to Attacks

The proposed PUF leverages the data-dependent dynamic timing behaviour of pipelined cores to generate unique and reliable responses. At the most fundamental level, the entropy source is the variance in delays within the core. It has been previously demonstrated that many delay-based PUFs are vulnerable to attacks based on ML modelling [59]. Further, obfuscation of the PUF response and other established methods of impeding ML modelling attacks are not possible without the addition of hardware, which invalidates the fully software-based nature of the design. As such, while a full analysis of ML attacks against the proposed design is beyond the scope of this work

(though it may form the basis of future work), it is useful to discuss the possible vulnerability to this kind of attack.

Conventional delay-based PUFs are in most cases fairly simplistic in their underlying architecture. In the case of conventional arbiter and RO PUFs this can be as simple as a fixed size chain of delay elements [31, 67]. Derivation of a mathematical model able to represent the variables that contribute to entropy within them is fairly trivial. Conversely, the scale and complexity of even a moderately complex pipelined core is an order of magnitude higher in regard to the number of variables and their interaction to form the final PUF response. As such, it is not clear that deriving a model will be trivial in comparison to conventional delay-based PUFs, even with a sufficiently large set of training data.

In regards to the gathering of training CRPs, there are also non-trivial challenges. Unlike in conventional delay-based PUFs, the possible inputs and the challenge set significantly vary. In fact, depending on the exact implementation, the challenge set may be as small as 15K ISQs out of the total possible $\left(6 \times 2^{64} \times 2^{64}\right)^6 \approx 7.24 \times 10^{235}$ (based on six instruction ISQs with two 64-bit operands as discussed in Section 5.1). The vast majority of the ISQs outside of this set will produce minimal or zero entropy (since they do not produce timing errors) and hence will not be useful for training a model. It has been estimated [76] that roughly 99% of timing critical paths are triggered by less than 10% of all ISQs; and the probability to obtain the worst-case input conditions, which result in timing errors, is extremely small [22].

Which ISQs are useful for triggering critical paths can be determined quite readily with full access to the original design files and using the methods outlined above. Nevertheless, for an attacker without such access or prior knowledge of the ISQs, to target the model must be trained using arbitrarily chosen ISQs. As the vast majority of these will be non-entropic, this can be compared to the well known poisoning attack [39].

Hence, even if an adversary has full access to the PUF for a relatively long period the odds of generating a valid training dataset without prior knowledge of the challenge set, are very low. In fact, an adversary need to know precise details of the chip as well as all the parameters that affect the dynamic timing behaviour of errors (i.e., clock reduction, type of instructions, input operands, instruction execution history). In effect, the necessity of acquiring or deriving the challenge set itself acts as a layer of obfuscation. However, the security of DTA-PUF would be compromised if an attacker had access to the challenge set and could overwrite it. To protect challenges from being overwritten, secure access mechanisms and data authentication protocols (e.g., digital signature and cryptographic hash) have been proposed [63, 65]. Note that we focus on the scenario that the input challenge set is secured and cannot be used for malicious activities.

While neither of these factors in themselves render the proposed PUF definitively immune to modelling attacks [27], they are non-trivial barriers. This raises the bar of entry and forces an adversary to expend greater time and resources than it would seem at first glance.

6 RELATED WORK

In this section, we present an overview of work in the field to date regarding processor-based PUFs, considering designs that are non-software-based, requiring the addition of substantial additional hardware; lightweight non-software-based, where the additional hardware required is relatively small; and fully software-based processor PUFs where the response is entirely derived using software. The comparison of all the PUFs explained below is shown in Table 1.

6.1 Fully Software Processor-based PUFs

There are relatively few designs for processor-derived PUFs that are fully software-based (i.e., which require no additional hardware whatsoever to perform the PUF query procedure).

Table 1. Processor-derived PUF Designs

PUF	Type	Hardware Additions	Practical Considerations	Average Uniqueness	Suitable for Resource-constrained Devices
HELP [5]	Non-Software	Large test circuit	Long query time, Limited response size	49.9%	✗
PASC [8]	Non-Software	Instruction logic	Must implement a custom instruction	38.2/49.2%	✗
Maiti et al. [48]	Fully Software	None	Difficult to characterise, Programmable PLL/clock generator required, High power consumption	37.5%	✗
PUFAtt [44]	Lightweight Non-Software	Arbiters after redundant ALU components	A means to synchronise ALU inputs is required	35.9%	✗
DScan PUF [81]	Lightweight Non-Software	Scan chains, Additional control circuitry	Response generation method is highly complex, Programmable PLL/clock generator required	49.9%	✗
Scan PUF [78]	Lightweight Non-Software	Scan chains, Modifications to signal generating circuits	Requires generation of controlled input signals	47%	✗
DTA-PUF	Fully Software	None	Single raised clock domain is required	31.4%	✓

Maiti et al. [48] propose a fully software processor-based PUF. The principle of operation is that by altering the clock frequency of a processor by a specific amount and running a sequence of instructions some large number of times, the failure rate is unique to a given processor at a given frequency. This is because for a given instruction there is a curve of increasing failure rate approaching 100% as frequency increases with the start and end points of this curve determined by low level variances unique to a given processor. So, at a clock reduction of, for example, 26% for the same instruction set no failures may occur on one chip, 20% failure rate on another, 76% failure rate on a third, and so on, depending on the position and shape of the failure curves in regards to increasing frequency for each chip. The failure rate is encoded such that minimal failures constitute a binary 00, the lower half of the curve 01, the upper half 10, and near or complete failure 11.

This has the advantage of being a fully software-derived design, assuming that a clock **phase-locked loop (PLL)** or other programmable clock control mechanism is available. However, it has several drawbacks. The design as proposed is highly, though not ideally, reliable but the performance in terms of inter-chip uniqueness is far from ideal. A given instruction and operand can begin to fail anywhere within a quite wide range of frequencies, meaning that multiple frequency values must be used to ensure uniqueness. Due to this, the total available response bits is also somewhat limited by the granularity of control over the frequency. Characterising this PUF is challenging as every instruction must be characterised at every frequency within the given range to gather full CRP data. Further, the generation of every two response bits requires the execution of some large number of instructions in the order of hundreds or thousands. Last, the source of entropy is the number of failed instructions at a given point, but this ignores the fact that two

failed instructions with non-correctable outputs may in fact have differing levels of bitwise error and different positioning of errors within the output of nominally "failed" instructions.

6.2 Lightweight Non-software Processor-based PUFs

Several processor-derived PUFs have been proposed that attempt to minimise the hardware overhead if additional circuitry is needed or that are nominally software-based but make use of test structures that may not be present in the right configuration in practice.

PUFatt, proposed by Kong et al. [44], is comparatively lightweight. It inserts delay arbiters after redundant (i.e., identical) ALU circuits within a processor design. This minimises the hardware resources required as only the arbiter components themselves are new. The same instruction is passed to each ALU and the response bits generated based on the relative speed of execution. The execution time is a product of the path delay variances within each ALU. This is very similar conceptually to well studied delay-based PUFs, such as the arbiter PUF [31], in that the entropy source is a product of cumulative path delay variances. The resultant PUF response is not ideal but is within acceptable bounds with 35.9% uniqueness on average and average reliability of 88.7%. An obfuscation scheme is proposed that increases the measured uniqueness to 44.6% at the cost of requiring additional post-processing.

While this design minimises the extra hardware resource usage, it is still not fully software-based. Performance in regard to security metrics is passable but not ideal and much lower in certain metrics than other processor PUF designs such as that proposed by Maiti et al. [48]. In addition, the reliability of the design is measured only for the raw output and not for the final output of the proposed obfuscation scheme, which, by the admission of the authors, can produce distinctly different outputs if even a small number of input bits (from the raw PUF response) are incorrect. As this design is conceptually similar to many well studied delay PUFs and does not implement any kind of CRP obfuscation, it may be vulnerable to common ML modelling attacks [27].

Kong and Koushanfar [43] take a similar approach, implementing arbiters after redundant ALU components. This work is more developed and applies some of the obfuscation techniques now common to delay PUF architectures, which at least partially mitigates the risk of ML modelling attacks. As with PUFatt [44], this design is not fully software-based. It performs fairly well with a uniqueness in the order of 38% and the authors propose a targeted aging algorithm, which can reportedly increase this to 45%. However, it is pointed out by the authors that this algorithm could also be used maliciously to reduce the uniqueness of the PUF, and thus rendering it more vulnerable to modelling. Preventing such an attack would require additional aging detection circuitry in addition to the arbiters needed for PUF operation.

Scan PUF, proposed by Zheng et al. [78], uses the delay variances in scan chains, a common design-for-test structure, as the entropy source. Such structures are not universally present in commodity devices but are very common and as such this design is largely software-based. However, additional circuitry is required to extract responses from the scan chain structures. This design performs well in key security metrics with a uniqueness of 47% and reliability above 95% at room temperature in practical tests.

Whilst the scan chains are already present on the device, it is necessary to generate precise input signals to generate a PUF response from them. In Scan PUF, this is done by adding a small amount of signal generation circuitry. This prevents the design from being truly software-based but has significantly lower hardware resource overheads than many of the other proposed designs. It is also noted by the authors that the implementation of Scan PUF produces small increases in power consumption in test modes. No analysis is given of the power overheads of switching between test and functional mode to access the PUF response.

Zheng et al. [81] later proposed DScan PUF. This design operates on similar principles to the earlier Scan PUF [78] but employs a novel control circuit on top of the already present scan chains. This control circuit facilitates the design of a PUF with more robust characteristics than Scan PUF. The resultant PUF is ideal in terms of uniqueness and can generate highly reliable responses. However, as with the Scan PUF, while it is relatively low overhead in terms of hardware resources, it is not truly software-based as design changes and some hardware resource usage are required to implement it.

6.3 Non-software Processor-based PUFs

Aarestad et al. proposed **Hardware Embedded Delay PUF (HELP)** [5], a design that makes use of an added on-chip test structure called "REBEL" to measure the path delays directly. Several extraction methodologies are proposed to convert pairs of path delays into a binary PUF response. The rate of generation of response bits is limited, but the response that can be generated is both highly unique and reliable under the proposed conditions.

The auxiliary test circuit used in HELP requires a large amount of hardware resource—more than 100% of the size of the circuit being measured in the experiments reported. Notably, query time is very long in comparison to what is typical in PUF designs with a rate of less than 3 bits per second on the test hardware. Furthermore, the methods of key extraction are computationally intensive and limit the maximum number of bits that can be generated to a practical upper bound of 256 for the most robust generation method. This design excellently performs in regards to security metrics, but the hardware costs in addition to the computational cost and corresponding power cost renders it unsuitable for resource-constrained systems.

Aysu and Schaumont proposed PASC [8] to address some of the weaknesses in the work of Maiti et al. [48]. Specifically they note the requirement for the ability to control the clock frequency over quite a large range to target the failure points of certain instructions. They propose that it is not practical to use the native instructions for this reason. To this end, PASC [8] employs a custom instruction designed to fail at around the same frequency in all devices. The PUF responses are differentiated by the slope of the failure curve, rather than both slope and positioning on the frequency spectrum.

This has several advantages. Only a single raised clock domain is required to implement the PUF and the output is both reasonably unique and highly reliable. Nevertheless, outside of the domain of FPGAs the implementation of the required custom instruction requires architectural changes to the processor and is thus non-software. Further, the problem of needing multiple clock domains only applies if the failure rate is used as the entropy source.

Overall, in this article, we aim at implementing a fully software, processor-based PUF design that combines the positive aspects of the PUFs discussed above and addresses their limitations. As has been demonstrated, DTA-PUF extracts entropy from instructions at a single raised clock domain by proper selection of instruction sequences and by looking at the error rate within the failed instructions. This allow us to provide a simple and quick challenge-response mechanism that generates output bits with a considerably small number of input instructions. Such properties render DTA-PUF extremely suitable for low-power, resource-scarce devices though the average uniqueness value of our PUF deviates from the ideal value of 50%.

7 CONCLUSIONS

In this article, we presented DTA-PUF, a fully software-derived PUF design that is based on the existing timing variability of pipelined cores. Our novel method closely intertwines the PUF with the underlying circuit, exploiting the inherent complexity of timing error occurrence. Our evaluation results suggest that the proposed PUF shows a good amount of uniqueness and reliability in terms

of the PUF responses and, at the same time, incurs limited or no processing costs and minimal power overheads. Moreover, the high bit generation speed of our PUF results in significant power savings when compared with existing fully software-based processor PUFs. DTA-PUF can be used as a lightweight mechanism to address emerging challenges in hardware security of devices that have power and computational resource concerns, such as IoT nodes, especially in regard to device identification and authentication.

REFERENCES

[1] IEEE Standard for Floating-Point Arithmetic, in *IEEE Std 754-2008*, vol., no., pp.1-70, 29 Aug. 2008, DOI: 10.1109/IEEESTD.2008.4610935

[2] NanGate. 2011. FreePDK45 Open Cell Library. Retrieved from http://nangate.com.

[3] International Data Corporation. 2021. Retrieved from https://www.idc.com/.

[4] OpenRISC Community. OpenRISC 1000 architecture manual. Retrieved from https://openrisc.io/or1k.html.

[5] J. Aarestad, P. Ortiz, D. Acharyya, and J. Plusquellic. 2013. HELP: A hardware-embedded delay PUF. *IEEE Design Test* 30, 2 (2013), 17–25.

[6] M. N. Alraja, M. M. J. Farooque, and B. Khashab. 2019. The effect of security, privacy, familiarity, and trust on users' attitudes toward the use of the IoT-based healthcare: The mediation role of risk perception. *IEEE Access* 7 (2019), 111341–111354. DOI: 10.1109/ACCESS.2019.2904006

[7] H. Amrouch, B. Khaleghi, A. Gerstlauer, and J. Henkel. 2017. Towards aging-induced approximations. In *Proceedings of the Design Automation Conference (DAC'17)*. 1–6.

[8] A. Aysu and P. Schaumont. 2013. PASC: Physically authenticated stable-clocked soc platform on low-cost FPGAs. In *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig'13)*. 1–6.

[9] Anys Bacha and Radu Teodorescu. 2014. Using ECC feedback to guide voltage speculation in low-voltage processors. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'14)*. IEEE Computer Society, 306–318. https://doi.org/10.1109/MICRO.2014.54

[10] B. M. S. Bahar Talukder, B. Ray, D. Forte, and M. T. Rahman. 2019. PreLatPUF: Exploiting DRAM latency variations for generating robust device signatures. *IEEE Access* 7 (2019), 81106–81120.

[11] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. 1991. The NAS parallel benchmarks summary and preliminary results. In *Proceedings of the Supercomputing Conference*. 158–165.

[12] G. Bakirtzis, B. J. Simon, A. G. Collins, C. H. Fleming, and C. R. Elks. 2019. Data-driven vulnerability exploration for design phase system analysis. *IEEE Syst. J.* 14, 4 (2020), 4864–4873. DOI: 10.1109/JSYST.2019.2940145

[13] J. Bhasker and R. Chadha. 2009. *Static Timing Analysis for Nanometer Designs: A Practical Approach*. Springer, New York, NY.

[14] K. A. Bowman, J. W. Tschanz, S. L. Lu, P. A. Aseron, M. M. Khellah, A. Raychowdhury, B. M. Geuskens, C. Tokunaga, C. B. Wilkerson, T. Karnik, and V. K. De. 2011. A 45 nm resilient microprocessor core for dynamic variation tolerance. *IEEE J. Solid-State Circ.* 46, 1 (2011), 194–208.

[15] Chun-Kai Chang, Wenqi Yin, and Mattan Erez. 2019. Assessing the impact of timing errors on HPC applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'19)*, Michela Taufer, Pavan Balaji, and Antonio J. Peña (Eds.). ACM, 70:1–70:19. https://doi.org/10.1145/3295500.3356184

[16] B. Cline, K. Chopra, D. Blaauw, and Y. Cao. 2006. Analysis and modeling of CD variation for statistical static timing. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*. 60–66.

[17] Jeremy Constantin, Andreas Peter Burg, Zheng Wang, Anupam Chattopadhyay, and Georgios Karakonstantis. 2016. Statistical fault injection for impact-evaluation of timing errors on application performance. In *Proceedings of the Design Automation Conference (DAC'16)*. 13:1–13:6. https://doi.org/10.1145/2897937.2898095

[18] Jeremy Constantin, Lai Wang, Georgios Karakonstantis, Anupam Chattopadhyay, and Andreas Burg. 2015. Exploiting dynamic timing margins in microprocessors for frequency-over-scaling with instruction-based clock adjustment. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'15)*, Wolfgang Nebel and David Atienza (Eds.). ACM, 381–386. https://doi.org/10.7873/DATE.2015.0303

[19] Abhishek Das, Serkan Ozdemir, Gokhan Memik, Joseph Zambreno, and Alok Choudhary. 2007. Mitigating the effects of process variations: Architectural approaches for improving batch performance. In *Proceedings of the Workshop on Architectural Support for Gigascale Integration (ASGI)*, San Diego, CA.

[20] J. Delvaux and I. Verbauwhede. 2013. Side channel modeling attacks on 65nm arbiter PUFs exploiting CMOS device noise. In *Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST'13)*. 137–142.

[21] S. Dighe, S. R. Vangal, P. Aseron, S. Kumar, T. Jacob, K. A. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, V. K. De, and S. Borkar. 2011. Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core TeraFLOPS processor. *IEEE J. Solid-State Circ.* 46, 1 (2011), 184–193.

[22] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge. 2010. Near-threshold computing: Reclaiming Moore's law through energy efficient integrated circuits. *Proc. IEEE* 98, 2 (2010), 253–266.

[23] Anh Nguyen Duc, Ronald Jabangwe, Pangkaj Paul, and Pekka Abrahamsson. 2017. Security challenges in IoT development: A software engineering perspective. In *Proceedings of the XP Scientific Workshops (XP'17)*. Association for Computing Machinery, New York, NY, Article 11, 5 pages. https://doi.org/10.1145/3120459.3120471

[24] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Architecture support for disciplined approximate programming. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'12)*, Tim Harris and Michael L. Scott (Eds.). ACM, 301–312. https://doi.org/10.1145/2150976.2151008

[25] W. Yan F. Tehranipoor, N. Karimian and J. A. Chandy. 2017. DRAM-based intrinsic physically unclonable functions for system-level security and authentication. *IEEE TVLSI* 25, 3 (2017), 1085–1097.

[26] S. Ganapathy, A. Teman, R. Giterman, A. Burg, and G. Karakonstantis. 2015. Approximate computing with unreliable dynamic memories. In *Proceedings of the IEEE 13th International New Circuits and Systems Conference (NEWCAS'15)*. 1–4. https://doi.org/10.1109/NEWCAS.2015.7182027

[27] Fatemeh Ganji, Shahin Tajik, Fabian Fäßler, and Jean-Pierre Seifert. 2016. Strong machine learning attack against PUFs with no mathematical model. In *Proceedings of the Cryptographic Hardware and Embedded Systems (CHES'16) (Lecture Notes in Computer Science)*, Vol. 9813. Springer, 391–411.

[28] Fatemeh Ganji, Shahin Tajik, and Jean-Pierre Seifert. 2016. PAC learning of arbiter PUFs. *J. Cryptogr. Eng.* 6, 3 (2016), 249–258. https://doi.org/10.1007/s13389-016-0119-4

[29] M. Gao, Q. Wang, M. T. Arafin, Y. Lyu, and G. Qu. 2017. Approximate computing for low power and security in the Internet of Things. *Computer* 50, 6 (2017), 27–34.

[30] D. Garyfallou, I. Tsiokanos, N. Evmorfopoulos, G. Stamoulis, and G. Karakonstantis. 2020. Accurate estimation of dynamic timing slacks using event-driven simulation. In *Proceedings of the 21st International Symposium on Quality Electronic Design (ISQED'20)*. 225–230. https://doi.org/10.1109/ISQED48828.2020.9137017

[31] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. 2002. Silicon physical random functions. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'02)*. 148.

[32] Georgios Karakonstantis, Nilanjan Banerjee, Kaushik Roy, and Chaitali Chakrabarti. 2007. Design methodology to trade off power, output quality and error resiliency: Application to color interpolation filtering. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 199–204. https://doi.org/10.1109/ICCAD.2007.4397266

[33] Beayna Grigorian, Nazanin Farahpour, and Glenn Reinman. 2015. BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing. In *Proceedings of the 21st IEEE International Symposium on High Performance Computer Architecture (HPCA'15)*. IEEE Computer Society, 615–626. https://doi.org/10.1109/HPCA.2015.7056067

[34] Puneet Gupta, Yuvraj Agarwal, Lara Dolecek, Nikil D. Dutt, Rajesh K. Gupta, Rakesh Kumar, Subhasish Mitra, Alexandru Nicolau, Tajana Simunic Rosing, Mani B. Srivastava, Steven Swanson, and Dennis Sylvester. 2013. Underdesigned and opportunistic computing in presence of hardware variability. *Trans. Comput.-Aided Des. Integr. Circuits Syst.* 32, 1 (2013), 8–23. DOI: 10.1109/TCAD.2012.2223467

[35] B. Halak, M. Zwolinski, and M. S. Mispan. 2016. Overview of PUF-based hardware security solutions for the internet of things. In *Proceedings of the IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS'16)*. 1–4.

[36] T. Hiramoto, K. Takeuchi, T. Mizutani, A. Ueda, T. Saraya, M. Kobayashi, Y. Yamamoto, H. Makiyama, T. Yamashita, H. Oda, S. Kamohara, N. Sugii, and Y. Yamaguchi. 2016. Ultra-low-power and ultra-low-voltage devices and circuits for IoT applications. In *Proceedings of the IEEE Silicon Nanoelectronics Workshop (SNW'16)*. 146–147.

[37] Giang Hoang, Robby Bruce Findler, and Russ Joseph. 2011. Exploring circuit timing-aware language and compilation. In *Proceedings of the Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11)*. ACM, 345–356.

[38] D. E. Holcomb, W. P. Burleson, and K. Fu. 2009. Power-Up SRAM state as an identifying fingerprint and source of true random numbers. *IEEE Trans. Comput.* (Sep. 2009). https://doi.org/10.1109/TC.2008.212

[39] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li. 2018. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'18)*. 19–35. DOI: 10.1109/SP.2018.00057

[40] X. Jiao, A. Rahimi, Y. Jiang, J. Wang, H. Fatemi, J. P. de Gyvez, and R. K. Gupta. 2018. CLIM: A cross-level workload-aware timing error prediction model for functional units. *IEEE Trans. Comput.* 67, 6 (2018), 771–783. DOI: 10.1109/TC.2017.2783333

[41] Georgios Karakonstantis, Abhijit Chatterjee, and Kaushik Roy. 2011. Containing the nanometer "pandora-box": Cross-layer design techniques for variation aware low-power systems. *IEEE J. Emerg. Sel. Topics Circ. Syst.* 1, 1 (2011), 19–29. https://doi.org/10.1109/JETCAS.2011.2135590

[42] G. Karakonstantis and K. Roy. 2011. Voltage over-scaling: A cross-layer design perspective for energy efficient systems. In *Proceedings of the 20th European Conference on Circuit Theory and Design (ECCTD'11)*. 548–551. https://doi.org/10.1109/ECCTD.2011.6043592

[43] J. Kong and F. Koushanfar. 2014. Processor-based strong physical unclonable functions with aging-based response tuning. *IEEE Trans. Emerg. Topics Comput.* 2, 1 (Mar. 2014), 16–29. DOI: 10.1109/TETC.2013.2289385

[44] J. Kong, F. Koushanfar, P. K. Pendyala, A. Sadeghi, and C. Wachsmann. 2014. PUFatt: Embedded platform attestation based on novel processor-based PUFs. In *Proceedings of the Design Automation Conference (DAC'14)*. 1–6. https://doi.org/10.1145/2593069.2593192

[45] N. Li, D. Liu, and S. Nepal. 2017. Lightweight mutual authentication for IoT and its applications. *IEEE Trans. Sustain. Comput.* 2, 4 (2017), 359–370. DOI: 10.1109/TSUSC.2017.2716953

[46] W. Liu, C. Gu, M. O'Neill, G. Qu, P. Montuschi, and F. Lombardi. 2020. Security in approximate computing and approximate computing for security: Challenges and opportunities. *Proc. IEEE* 108, 12 (2020), 2214–2231. https://doi.org/10.1109/JPROC.2020.3030121

[47] A. Maiti and P. Schaumont. 2011. Improved ring oscillator PUF: An FPGA-friendly secure primitive. *J. Cryptol.* 24 (2011), 375–397.

[48] A. Maiti and P. Schaumont. 2012. A novel microprocessor-intrinsic Physical unclonable function. In *Proceedings of the International Conference on Field Programmable Logic (FPL'12)*. 380–387. https://doi.org/10.1109/FPL.2012.6339208

[49] J. Miskelly and M. O'Neill. 2020. Fast DRAM PUFs on commodity devices. *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* 39, 11 (2020), 3566–3576. https://doi.org/10.1109/TCAD.2020.3012218

[50] M. S. Mispan, H. Su, M. Zwolinski, and B. Halak. 2018. Cost-efficient design for modeling attacks resistant PUFs. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'18)*. 467–472.

[51] Sparsh Mittal. 2016. A survey of techniques for approximate computing. *ACM Comput. Surv.* 48, 4 (2016), 62:1–62:33. https://doi.org/10.1145/2893356

[52] Lev Mukhanov, Dimitrios S. Nikolopoulos, and Bronis R. de Supinski. 2015. ALEA: Fine-grain energy profiling with basic block sampling. In *Proceedings of the International Conference on Parallel Architecture and Compilation (PACT'15)*. IEEE Computer Society, 87–98. https://doi.org/10.1109/PACT.2015.16

[53] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, P. Lawthers, and S. Das. 2017. Harnessing voltage margins for energy efficiency in multicore CPUs. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'17)*. 503–516.

[54] K. Parasyris, P. Koutsovasilis, V. Vassiliadis, C. D. Antonopoulos, N. Bellas, and S. Lalis. 2018. A framework for evaluating software on reduced margins hardware. In *Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'18)*. 330–337. https://doi.org/10.1109/DSN.2018.00043

[55] Taejoon Park and Kang G. Shin. 2004. LiSP: A lightweight security protocol for wireless sensor networks. *ACM Trans. Embed. Comput. Syst.* 3, 3 (Aug. 2004), 634–660. https://doi.org/10.1145/1015047.1015056

[56] Md. Tauhidur Rahman, Alison Hosey, Zimu Guo, Jackson Carroll, Domenic Forte, and Mark Tehranipoor. 2017. Systematic correlation and cell neighborhood analysis of SRAM PUF for robust and unique key generation. *J. Hardware Syst. Secur.* 1, 2 (2017), 137–155. https://doi.org/10.1007/s41635-017-0012-3

[57] A. Raychowdhury, S. Ghosh, and K. Roy. 2005. A novel on-chip delay measurement hardware for efficient speed-binning. In *Proceedings of the 11th IEEE International On-Line Testing Symposium*. 287–292.

[58] J. N. Rodrigues, M. Kamuf, H. Hedberg, and V. Owall. 2005. A manual on ASIC front to back end design flow. In *Proceedings of the IEEE International Conference on Microelectronic Systems Education (MSE'05)*. 75–76.

[59] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. 2010. Modeling attacks on physical unclonable functions. In *Proceedings of the ACM Computer and Communications Security Conference (CCS'10)*. ACM, 237–249. https://doi.org/10.1145/1866307.1866335

[60] Ulrich Rührmair, Xiaolin Xu, Jan Sölter, Ahmed Mahmoud, Mehrdad Majzoobi, Farinaz Koushanfar, and Wayne Burleson. 2014. Efficient power and timing side channels for physical unclonable functions. In *Proceedings of the 16th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'14)*. Springer-Verlag, Berlin, 476–492. https://doi.org/10.1007/978-3-662-44709-3_26

[61] S. Sakib, A. Milenković, M. T. Rahman, and B. Ray. 2020. An aging-resistant NAND flash memory physical unclonable function. *IEEE Trans. Electron Devices* 67, 3 (2020), 937–943.

[62] J. Sartori, A. Pant, R. Kumar, and P. Gupta. 2010. Variation-aware speed binning of multi-core processors. In *Proceedings of the 11th International Symposium on Quality Electronic Design (ISQED'10)*. 307–314.

[63] M. Sauer, P. Raiola, L. Feiten, B. Becker, U. RÃ¼hrmair, and I. Polian. 2017. Sensitized path PUF: A lightweight embedded physical unclonable function. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'17)*. 680–685. https://doi.org/10.23919/DATE.2017.7927076

[64] Sophiane Senni, Lionel Torres, Gilles Sassatelli, and Abdoulaye Gamatie. 2016. Non-volatile processor based on MRAM for ultra-low-power IoT devices. *J. Emerg. Technol. Comput. Syst.* 13, 2, Article 17 (Dec. 2016), 23 pages. https://doi.org/10.1145/3001936

[65] Dimitrios N. Serpanos and Artemios G. Voyiatzis. 2013. Security challenges in embedded systems. *ACM Trans. Embed. Comput. Syst.* 12, 1s, Article 66 (Mar. 2013), 10 pages. https://doi.org/10.1145/2435227.2435262

[66] K. Shi, D. Boland, E. Stott, S. Bayliss, and G. A. Constantinides. 2014. Datapath synthesis for overclocking: Online arithmetic for latency-accuracy trade-offs. In *Proceedings of the 51st ACM/EDAC/IEEE Design Automation Conference (DAC'14)*. 1–6. https://doi.org/10.1145/2593069.2593118

[67] G. E. Suh and S. Devadas. 2007. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference (DAC'07)*. 9–14.

[68] K. Tovletoglou, L. Mukhanov, G. Karakonstantis, A. Chatzidimitriou, G. Papadimitriou, M. Kaliorakis, D. Gizopoulos, Z. Hadjilambrou, Y. Sazeides, A. Lampropulos, S. Das, and P. Vo. 2018. Measuring and exploiting guardbands of server-grade ARMv8 CPU cores and DRAMs. In *Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W'18)*. 6–9. https://doi.org/10.1109/DSN-W.2018.00013

[69] I. Tsiokanos and G. Karakonstantis. 2021. ExHero: Execution history-aware error-rate estimation in pipelined designs. *IEEE Micro* 41, 1 (2021), 61–68. DOI: 10.1109/MM.2020.3012045

[70] Ioannis Tsiokanos, Lev Mukhanov, Giorgis Georgakoudis, Dimitrios S. Nikolopoulos, and Georgios Karakonstantis. 2020. DEFCON: Generating and detecting failure-prone instruction sequences via stochastic search. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE'20)*. IEEE, 1121–1126.

[71] I. Tsiokanos, L. Mukhanov, and G. Karakonstantis. 2019. Low-power variation-aware cores based on dynamic data-dependent bitwidth truncation. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE'19)*. 698–703. https://doi.org/10.23919/DATE.2019.8714942

[72] Ioannis Tsiokanos, Lev Mukhanov, Dimitrios S. Nikolopoulos, and Georgios Karakonstantis. 2018. Variation-aware pipelined cores through path shaping and dynamic cycle adjustment: Case study on a floating-point unit. In *Proceedings of the International Symposium on Low-power Electronics and Design (ISLPED'18)*. ACM, 52:1–52:6. https://doi.org/10.1145/3218603.3218617

[73] I. Tsiokanos, L. Mukhanov, D. S. Nikolopoulos, and G. Karakonstantis. 2019. Significance-driven data truncation for preventing timing failures. *IEEE TDMR* 19, 1 (Mar. 2019), 25–36. https://doi.org/10.1109/TDMR.2019.2898949

[74] E. I. Vatajelu, G. D. Natale, M. S. Mispan, and B. Halak. 2019. On the encryption of the challenge in physically unclonable functions. In *Proceedings of the IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS'19)*. 115–120.

[75] I. Verbauwhede, D. Mukhopadhyay, and S. S. Roy. 2016. Embedded security. In *Proceedings of the 29th International Conference on VLSI Design and 15th International Conference on Embedded Systems (VLSID'16)*. 23–23.

[76] J. Xin and R. Joseph. 2011. Identifying and predicting timing-critical instructions to boost timing speculation. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'11)*. 128–139.

[77] Yan Pan, Joonho Kong, S. Ozdemir, G. Memik, and Sung Woo Chung. 2009. Selective wordline voltage boosting for caches to manage yield under process variations. In *Proceedings of the 46th ACM/IEEE Design Automation Conference*. 57–62.

[78] Yu Zheng, A. R. Krishna, and S. Bhunia. 2013. ScanPUF: Robust ultralow-overhead PUF using scan chain. In *Proceedings of the 18th Asia and South Pacific Design Automation Conference (ASP-DAC'13)*. 626–631.

[79] Rafael Zalman and Albrecht Mayer. 2014. A secure but still safe and low cost automotive communication technique. In *Proceedings of the 51st Annual Design Automation Conference (DAC'14)*. Association for Computing Machinery, New York, NY, 1–5. https://doi.org/10.1145/2593069.2603850

[80] Hang Zhang, Mateja Putic, and John Lach. 2014. Low-power GPGPU computation with imprecise hardware. In *Proceedings of the 51st Annual Design Automation Conference (DAC'14)*. ACM, 99:1–99:6. https://doi.org/10.1145/2593069.2593156

[81] Y. Zheng, F. Zhang, and S. Bhunia. 2016. DScanPUF: A delay-based physical unclonable function built into scan chain. *IEEE Trans. Very Large Scale Integr. Syst.* 24, 3 (2016), 1059–1070.

Received June 2020; revised September 2020; accepted November 2020