SnoopyPower! Remote Power Attacks on Cache and Coherence Paths

Eliott Quéré*, Maria Méndez Real†, Alessandro Palumbo*, Thomas Rokicki*, Lilian Bossuet‡, Rubén Salvador*
*CentraleSupélec, Univ Rennes, Inria, CNRS, IRISA – {eliott.quere, alessandro.palumbo, thomas.rokicki, ruben.salvador}@irisa.fr
†Univ Bretagne-Sud, Lab-STICC, UMR 6285 – maria.mendez-real@univ-ubs.fr
‡Univ Lyon, UJM-Saint-Étienne, CNRS, Lab. Hubert Curien, UMR 5516 – lilian.bossuet@univ-st-etienne.fr

***Abstract*—When timing reaches its limits and fails to discriminate events, the microarchitecture falls silent. Yet power does not. Power leakage from the shared distribution network becomes key to remotely exploiting new side-channel sources on modern heterogeneous SoC architectures, directly challenging the literature built around timing-centric mitigations or presumed timing-level opacity of certain modern processor microarchitecture features, such as Arm's snoop control unit. In this work, we embed a time-to-digital converter in the programmable logic of a Zynq-7000 SoC-FPGA and show that a short, software-triggered power distribution network signature of a probed memory load from the Arm Cortex-A9 is sufficient to characterise the entire memory hierarchy (i.e., L1, L2, and DRAM) service, as well as coherence states from the snoop control unit. This reveals a new class of hybrid software-hardware vulnerabilities in which unprivileged user code, coordinated with an embedded power sensor, can record high-resolution traces from the shared power distribution network and enable fully remote, fine-grained microarchitectural observation using power leakage on heterogeneous architectures. Building on this capability, we introduce two new attack primitives: *Flush+Power*, a line-level resolution, same-core attack in shared-memory settings, and *SnoopyPower*, which reveals coherence-active cache lines across cores. Together, these results demonstrate how programmable logic can become a powerful physical threat vector to CPU microarchitectures in heterogeneous systems, revealing power consumption-based side channels that persist even when timing observation fails.**

***Index Terms*—Hardware security, side-channel, Power Distribution Network (PDN), power sensing, time-to-digital converter (TDC), cache attacks, coherence, Arm Cortex-A9, SoC-FPGA.**

I. Introduction

Microarchitectural side channels have traditionally relied on timing as the primary channel to expose processor behavior. Modern CPUs exhibit substantial latency differences across the memory hierarchy, and cache-based attacks show memory-access timing is often enough to infer microarchitectural state or recover secrets in shared environments. This reliance on timing has motivated a broad spectrum of countermeasures [1], [2]. Constant-time programming, timer perturbation, counter restrictions, and architectural latency equalization all aim to suppress or obfuscate timing variability. On many platforms, high-resolution timers are unavailable to unprivileged software [3], and in some heterogeneous processors, elements of the memory hierarchy are intentionally engineered to produce nearly identical timing profiles. As a result, constructing reliable and portable timer-based attacks has become increasingly challenging [4]–[6]. When timing no longer differentiates, microarchitectural behavior becomes inaccessible through the timing channel. This limitation, however, applies only to timing; other physical channels remain available.

As the adoption of heterogeneous architectures increases, so does the interaction of general-purpose cores and accelerators through shared on-die resources. These platforms merge previously separate compute domains into unified coherence regions, integrating last-level caches, coherent interconnects, clocking structures, and an on-chip power delivery network (PDN). This unification is driven by performance and power efficiency. However, consolidating heterogeneous components into a single coherent domain broadens the attack surface. Activity in one component can perturb or reveal the state of another through shared physical, architectural, and microarchitectural structures [7]–[9]. On Intel multicore processors, timing differences between coherence protocol states have already been characterized and exploited [10]. However, prior work observes that coherence-driven leakage in heterogeneous cache-coherent systems remains largely unexplored [9].

Power side channels are among the most established techniques in hardware security. Classical attacks correlate CMOS switching activity with analog power traces captured using oscilloscopes [11]–[13]. More recent work shows that coarse software-visible signals, such as energy counters [14] or DVFS behavior [15], can serve as power proxies leaking information about software execution and microarchitectural behavior. SoC-FPGA platforms extend this capability as the programmable logic (PL) can host custom hardware close to hard CPU cores while logically and physically separated. High-resolution power sensors, including time-to-digital converters (TDCs), can be embedded in the PL to observe fine-grained PDN fluctuations induced by CPU or accelerator activity [16], [17]. While such embedded sensors have been used for fingerprinting or for attacks on cryptographic circuits in the fabric, their use as remote probes of the microarchitectural behavior of hard CPU cores remains largely unexplored.

In this context, we ask the following research question:

Can remote power activity, captured through embedded PL sensors, resolve CPU microarchitectural states in heterogeneous SoCs, including those deliberately equalized in timing?

In this work, we focus on the Snoop Control Unit (SCU) in Arm SoCs and show that power, rather than timing, becomes

a decisive channel for observing microarchitectural states in heterogeneous SoCs. Peer-L1 transfers and self-loads converge into nearly identical timing signatures [18], intentionally making these coherence paths indistinguishable for unprivileged timers [3]. Nevertheless, underlying coherence operations and their associated traffic, which still exercise different hardware, may leave measurable power footprints on the shared PDN.

We embed a compact TDC in the PL of an AMD Zynq-7000 and construct a hybrid software-hardware attack model, in which we issue unprivileged and carefully fenced memory loads from software, while the embedded sensor captures short, software-triggered PDN traces. These traces reveal highly distinguishable signatures for each memory-service level (L1, L2, DRAM) and for coherence resolutions performed by the SCU. Even when timing is architecturally engineered to be indistinguishable, power measured remotely from the PL remains strongly differentiable. This observation transforms the PL into a built-in probe for characterizing the interaction between the PDN and the effects of microarchitectural execution using power leakage. Overall, our findings reveal new vulnerabilities and offer an opportunity for a more general and automated methodology to evaluate CPU security against power side channels in heterogeneous SoCs. Our contributions are summarized as follows:

- **Memory-level PDN characterization.** We demonstrate that brief PDN captures from an embedded TDC reliably separate L1, L2, and DRAM service on a heterogeneous SoC, and we quantify this separability using effect size and classification metrics.
- **Flush+Power: Same-core cache attack primitive.** Local flush followed by a PDN probe capture yields a timer-free cache-line resolution in shared memory.
- **SnoopyPower: Coherence-channel attack primitive.** Demonstration of power-based coherence-path leakage on Arm through the PDN in SoC-FPGAs, exposing SCU decisions and enabling cross-core address discovery.

Building on these contributions, **we demonstrate two end-to-end attacks**. Flush+Power recovers per-byte first-round AES T-table accesses in a same-core shared-page setting from a single PDN trace. SnoopyPower enables cross-core address discovery by discriminating the coherence path taken. Together, these results show how embedded power sensors introduce a new class of hybrid software/hardware attacks against CPU microarchitectures, which remain effective where timing-based approaches fail. To support open research, we release our code and data: https://zenodo.org/records/19003752.

The remainder of this paper is organized as follows. Section II introduces background and related work on SoC/cloud-FPGA platforms, power sensing and side-channel analysis, and cache/coherence architectures. Section III details the TDC and analysis pipeline and presents the results of the memory-level PDN characterization. Section IV introduces Flush+Power and evaluates its ability to recover cache-line accesses from a single PDN trace. Section V develops SnoopyPower and the associated coherence-path address-discovery experiments.

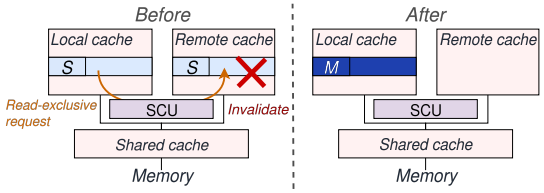


Fig. 1: MESI Shared-to-Modified transition: the requesting core issues a read-exclusive request, invalidates peer copies, and upgrades to the Modified state.

Section VI discusses portability, mitigations, and limitations.

II. Background and Related Work

A. Memory and coherence (Cortex-A9 / Zynq-7000)

Modern processors rely on hierarchical caches to reduce DRAM latency, providing low-latency access to recently used data. They are built as set-associative structures where each address maps to a set and the replacement policy determines which way is evicted [19]. In multicore systems, caches must also maintain coherence, which ensures that all cores observe a consistent view of shared data. Hardware coherence protocols implement this guarantee by tracking the state of each cache line. The widely used *MESI* protocol (*Modified, Exclusive, Shared, Invalid*) [20] defines state transitions as cores read, write, or evict lines. A *Shared* to *Modified* coherence upgrade is illustrated in Fig. 1: when a core attempts to write a line currently in the *Shared* state, it must first issue a read-exclusive request and invalidate peer copies before gaining ownership.

In Arm's Cortex-A9 MPCore, coherence is enforced for private L1 data caches through a snooping mechanism coordinated by the *SCU* [21]. Positioned between the per-core L1D caches and the shared memory (Fig. 2), the SCU maintains duplicate tag arrays that serve as a directory and snoop filter [21]. On an L1D miss, the SCU consults these tags to determine whether another core holds the requested line. If so, it issues a targeted snoop so the requester can obtain the data directly, including dirty lines in the Modified state. Otherwise, the request proceeds to the L2 cache or to DRAM. Coherent traffic eventually reaches the CoreLink Level-2 Cache Controller (PL310) [22], the architectural point of coherency for the Cortex-A9 cluster.

B. Cache attacks on Arm

Cache-based side channels on Arm have evolved from early timer-driven demonstrations to mature, cross-core attacks on contemporary SoCs. Initial studies on Cortex-A8/A9 showed that even low-resolution timers suffice to recover key-dependent access patterns from table-driven cryptography [23], [24], and eviction-based primitives were soon adapted to Arm's cache hierarchy [25]. As multicore Arm systems became mainstream, these techniques proved viable under realistic conditions such as smartphones [3], [26] or Trustzone [27].

Coherence itself constitutes another source of microarchitectural leakage. On Intel systems, coherence-state transitions have been shown to introduce observable timing variations [10]: an LLC hit completes in under 60 cycles, while a remote

private-cache hit via coherence takes around 90 cycles [28]. In contrast, a largely unexplored dimension is Arm's coherence service path. Arm SoCs SCU collapses latency differences between self-loads and peer-L1 snoops, making them effectively indistinguishable from software [18]. As a consequence, prior work on Arm resorts to indirect indicators to infer coherence activity [29]. Recent surveys [9] highlight that while such attacks are well studied on traditional multicore CPUs, they remain underexplored in heterogeneous cache-coherent systems and coherent accelerators. In particular, Arm's accelerator coherency port (ACP) extends coherence to the PL in SoC-FPGAs, exposing new interference channels between CPU caches and FPGA accelerators in Zynq-class SoCs [30], [31].

C. SoC/Cloud-FPGA

With the slowdown of Moore's Law, computing has entered an era of specialization where general-purpose CPUs coexist with domain-specific accelerators [32], [33]. In the cloud, this trend materializes through FPGA acceleration offered as *FPGA-as-a-Service* or *Acceleration-as-a-Service*, exposing reconfigurable logic through APIs or tenant-dedicated regions [7]. A common template underlies these systems: a static *shell* handles communication, DMA, and memory, while a dynamic *role* hosts user logic, enabling GPU-like virtualization and scheduling [7]. Early deployments relied on PCIe-attached cards [34] and SmartNICs [35], whereas *system-on-chip FPGAs* (SoC-FPGAs) integrate CPUs and programmable logic on a single die through coherent, low-latency fabrics.

Although SoC-FPGAs are still rare in public clouds, they dominate edge and embedded platforms for radio, vision, or other signal processing [36]. Their shared resources, such as memory and power domains served by a common PDN, make them particularly relevant for security research [8]. They expose coherence, memory, and PDN interactions at the same physical scale, where one subsystem's activity can influence the state of another. As designs evolve toward fully coherent on-package accelerators [7], understanding these intra-die couplings becomes central to assessing the security of future heterogeneous system deployments in the cloud [9], [37].

D. Remote power analysis

Power analysis exploits the data dependence of CMOS power consumption: each logic transition charges or discharges capacitance, and switching activity dominates instantaneous current [12], [38]. Classical side-channel attacks such as SPA and DPA/CPA [11]–[13] are non-profiled, as they do not require a prior device characterization. In contrast, profiled attacks build explicit, accurate statistical leakage models, or templates, using a copy of the target device during a dedicated training phase [39]; new forms of profiled attacks include supervised classifier training, e.g., deep neural networks [40].

Modern platforms increasingly expose on-chip physical measurements through software-accessible interfaces [41]. Coarse-grained energy counters,e.g., Intel RAPL, have enabled practical key-recovery attacks [14] and memory-hierarchy-induced leakage [42]. Frequency-based channels further translate power variations into timing leakage [15], [43], [44]. Other on-chip and platform interfaces, e.g., DDR delay lines or battery telemetry, have been exploited to recover secrets or infer activity [45]–[48].

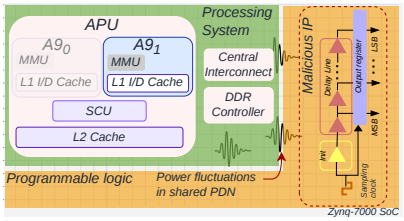


Fig. 2: Zynq-7000 SoC with on-chip malicious IP embedding a TDC and capturing PDN fluctuations induced by CPU activity.

In cloud-FPGA infrastructures, multi-tenant access introduces a new attack surface [7], [49], [50]. Adversaries can instantiate custom IP blocks as power sensors that monitor the shared PDN and capture voltage fluctuations from co-resident workloads. Early demonstrations used ring oscillators [16] for remote cross-tenant leakage [17]; later works introduced TDCs for finer PDN sampling; subsequent works focused on stealthier implementations using LUTs [51], [52], DSPs [53], MUXs [54], or routing delays [55]. These sensors have recovered secrets from cryptographic IPs [17], [56], [57], extracted properties of DNN or binarized neural network (BNN) accelerators [58]–[61], and observed CPU workloads like AES and RSA through the shared PDN [16], [17]. Yet passive, high-resolution observation of hard CPU cores from the PL remains scarce; the clearest instruction-level demonstrations still target soft-core CPUs instantiated in the PL [62].

Our contribution with this work is an alternative observable that defeats timing-indistinguishable behaviors for microarchitectural attacks in Arm SoC-FPGAs where CPUs and accelerators operate within shared coherence and power domains. We show how using power consumption through the shared use of the PDN we can distinguish memory and coherence paths. Table I summarizes representative profiled attacks on memory, cache, and coherence on Arm, and illustrates how our new attack primitives (*Flush+Power* and *SnoopyPower*) extend existing timing and power/EM approaches.

III. Characterizing memory access levels

This Section characterizes the PDN behaviour resulting from executing a privileged and an unprivileged microbenchmark targeting different memory hierarchy levels. Using a TDC in the PL and a simple Linear Discriminant Analysis (LDA) classifier, we demonstrate effective discrimination among memory levels (L1, L2, DRAM).

A. Hardware platform and measurement setup

We target an AMD Zynq-7000 (XC7Z020) SoC on a Digilent Zybo Z7-20 board, featuring 2 Arm Cortex-A9 cores (private L1s), an SCU, and a shared L2 cache in the processing system (PS), schematized in Fig. 2. PS and PL share the same PDN. All experiments run on a single core in a *bare-metal* configuration, with the second core disabled to elimi-

TABLE I: Comparison of cache side-channel attacks on Arm processors.

Attack	Source	Medium	Arm Cortex cores	Target Scope	Flush/ES	SHM	Remote	Policy	Scope
SnoopyPower†	Power	TDC	A9	Coherence state	-	✓	✓	**C**	●
Flush+Power†	Power	TDC	A9	L1-D hit/miss	✓/×	✓	✓	**A**	○
Modify+Recall [63]	Logic	Exec. Flow	A53, A76	Stale instruction recall	-	×	✓	**WC**	●
Store+Ret [64]	Logic	Exec. Flow	many cores	Incoherence I/D cache	-	×	✓	**A**	●
Lx+Sx [64]	Logic	Registers	many cores	Cache-line state	-	×	✓	**A**	●
SF-Prime+Probe [29]	Timing	SW Timer	A53, A73	SF directory state	×/✓	×	✓	**NI**	●
EVICT+EM [65]	EM	DRAM Bus	A9	Cache miss	×/✓	×	×	**A**	●
PRIME+EM [65]	EM	DRAM Bus	A9	Cache set activity	-	×	×	**A**	●
COLLISION+EM [65]	EM	DRAM Bus	A9, A7	Intra-table collision	-	×	×	**A**	●
Return-Oriented F+R [66]	Timing	SW timer	A53, A57	L1-I / LLC	✓/×	✓	✓	**I**	●
Evict+Reload [3]	Timing	SW Timer	A53	LLC hit/miss	×/✓	✓	✓	**I**	●
Prime+Probe [3]	Timing	SW Timer	A53	LLC set activity	×/✓	×	✓	**I**	●
Flush+Reload [3]	Timing	SW Timer	A57	LLC hit/miss	✓/×	✓	✓	**I**	●
Flush+Flush [3]	Timing	SW Timer	A57	LLC hit/miss	✓/×	✓	✓	**I**	●

†This work. **Policy: I**: Inclusive, **NI**: Non-Inclusive/Directory, **C**: Coherence, **WC**: Weak Coherence, **A**: Any/Agnostic.
Scope: ○: Same-Core only, ●: Cross-Core/Cluster. **Flush/ES:** Format is Flush/Eviction Set. **Shared Memory (SHM):** Attacker & victim share phys. page.

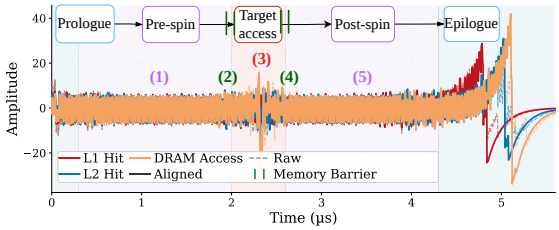


Fig. 3: Aligned mean power traces for the same code snippet across L1, L2, and DRAM memory accesses

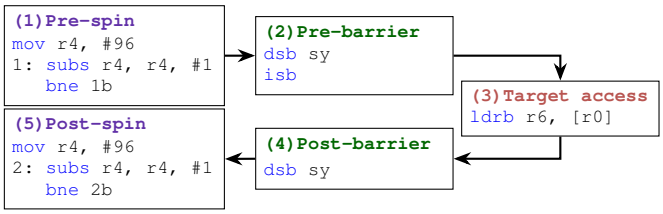


Fig. 4: Privileged code with strong memory barriers.

nate interference. Both the evaluation code and measurement logic execute on the same core. The acquisition, storage and analysis pipeline features an online and offline phase. **For the online phase**, trace acquisition is based on the SCABox framework [17] with default drivers. The TDC is clocked at 250 MHz (250 MSa/s), yielding a sampling period of 4 ns. The CPU runs at $f_{\mathrm{CPU}} = 112.5\,\mathrm{MHz}$ The TDC sensor bank writes 32-bit trace samples into a synchronous BRAM FIFO (8192 words) at the PL clock rate on the attacker side. The TDC capture is triggered when the microbenchmark in Fig. 3 is executed, so the measurement window deterministically spans the target load. The assembly is ARMv7, and identical across classes. The PS reads the stored samples through an AXI4-Lite register interface after acquisition completes, ensuring decoupling between capture and readout. Samples, i.e., raw traces without any pre-processing or alignment, are transmitted to the host via UART for the offline analysis phase, performed on the host machine. **The offline phase** includes pre-processing (high-pass filtering at 1 MHz, cropping), trace alignment (intra-class cross-correlation), PoI selection (KS-based with Bonferroni correction), classifier training (LDA/QDA with cross-validation), and attack evaluation.

B. Discriminating L1/L2/DRAM with privileged instructions

Our first objective is to show that a *single, fixed* microbenchmark yields distinct power signatures, as captured by the on-chip TDC, for three controlled service paths: private-L1 hit, shared-L2 hit, and cache miss.

1) Microbenchmark: To have a precise measurement window, we isolate a single fenced byte load. Fig. 4 shows the full routine. The execution begins with a short delay spin **(1)**. This pre-access delay stabilizes the pipeline and aligns the PDN transient with the TDC capture window, even when the CPU frequency changes. Immediately following this preamble, the *essential* fence-load-fence sequence is executed (**2**-**3**-**4**). This sequence is identical across L1, L2, and DRAM cases, and aligns the TDC trigger to `[r0]` memory service.

The first data-synchronization barrier (DSB) ensures that all preparatory cache operations (Section III-B2) complete before the load. The instruction-synchronization barrier (ISB) then flushes the pipeline, discarding any prefetched or speculative instructions so that the subsequent load byte instruction **(3)** executes with the correct architectural state. A final DSB guarantees that the access has been completed architecturally before any following instruction may retire. This fence-load-fence pattern is identical across runs; only the *pre-call cache state* differs (L1-resident, L2-resident, or evicted to DRAM).

The probe concludes with a second delay spin **(5)**, which stabilizes the front end and provides guard time around the memory access. With a 4 ns aperture at 250 MSa/s, this margin centers the memory-access transient within the capture window and reduces sensitivity to surrounding noise.

2) Cache-State Preparation: We configure the residency of the target cache line using Arm CP15 and PL310 maintenance operations via AMD bare-metal APIs [67]. Each cache preparation stage is terminated with `DSB; ISB` to prevent any bleed-through into the measurement window.

(i) *L1 hit:* Load the target address once to bring it into *CPU0*'s private L1 D cache, without further maintenance. After fencing, the `ldrb` resolves from L1.

(ii) *L2 hit:* Fetch the line once so it is present in the hierarchy, then flush CPU0's L1 D-cache line. The line remains clean and valid in the shared L2. After fencing, the `ldrb` misses in L1 and hits in L2.

(iii) *DRAM access:* Flush the entire data cache hierarchy (L1+L2) so that the line is absent from both levels. After fencing, the `ldrb` traverses the SCU/L2/L1 pipeline and is serviced from DRAM.

3) Trace signatures, Point-of-Interest (POI) selection, and separability: We analyze transient PDN signatures generated by controlled memory-access events $\mathcal{E} = \{e_{\mathrm{L1}}, e_{\mathrm{L2}}, e_{\mathrm{DRAM}}\}$, corresponding to an L1 hit, an L1 miss/L2 hit and a DRAM access respectively. All traces are high-pass filtered at 1 MHz to remove regulator drift while preserving short-lived transients.

Let $\mathbf{X} \in \mathbb{R}^{N \times T}$ be the aligned trace matrix, with class labels $Y_i \in \mathcal{E}$ and class-conditional CDF $F_{c,t}(x) = \Pr(X_t \leq x \mid Y = c)$.

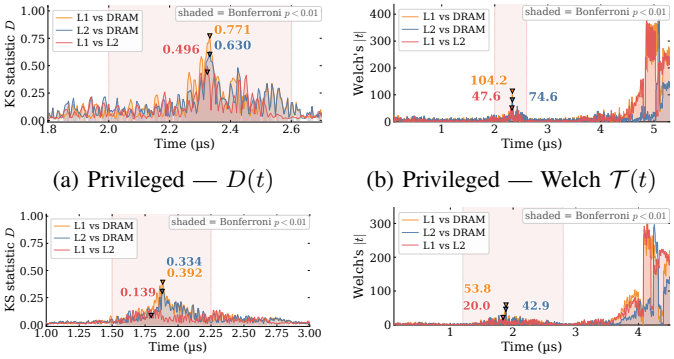
Shapiro–Wilk tests [68] with Bonferroni correction ($M = 3 \cdot T_w$, $\alpha = 0.01$) reject normality at every time sample for all three classes ($W \leq 0.989$, $p < 10^{-17}$). We complement these results with Pearson χ^2 goodness-of-fit tests, which confirm non-Gaussian structure at all sampled time slices ($p \leq 0.05$). We consequently adopt a non-parametric inferential framework, retaining Welch's t-statistic solely as a descriptive leakage map.

For each class pair (a, b) and time sample t within an *a priori* execution window, we apply the two-sample Kolmogorov-Smirnov test [69]:

$$D_{a,b}(t) = \sup_x \left| \widehat{F}_{a,t}(x) - \widehat{F}_{b,t}(x) \right|, \tag{1}$$

testing $H_0^{(a,b,t)} : F_{a,t} = F_{b,t}$ against the omnibus alternative of any distributional difference. At each time sample t within the analysis window of T_w samples, the global null hypothesis is $H_0^{(t)}$: $F_{e_{\mathrm{L1}},t} = F_{e_{\mathrm{L2}},t} = F_{e_{\mathrm{DRAM}},t}$, i.e. all three class distributions are identical. We test this via three pairwise KS tests $H_0^{(a,b,t)}$: $F_{a,t} = F_{b,t}$; rejection of any pair at time t constitutes evidence of class-dependent leakage. To control the familywise error rate $\Pr[\text{at least one false rejection}] \leq \alpha$ across the $M = 3 \cdot T_w$ simultaneous hypotheses, we apply Bonferroni correction [70], rejecting $H_0^{(a,b,t)}$ iff $p_{a,b}(t) < \alpha/M$ at $\alpha = 0.01$. As shown in Fig. 5a and 5b, both metrics isolate a narrow, high-contrast interval around the fenced `LDRB` and the end of trace marked by the timing difference. PoI selection proceeds by aggregating the pairwise KS statistic $D_{a,b}(t)$ across all three class pairs, retaining only samples that are Bonferroni-significant at α=0.01 across M=3T_w joint hypotheses, with a minimum inter-PoI spacing of 3 samples, then sweeping $n_{\mathrm{PoI}} \in [8, \ldots, 256]$.

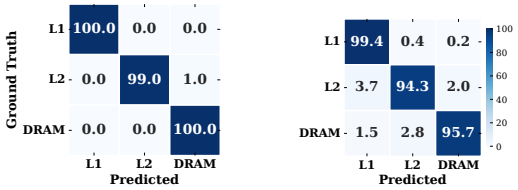
For each count, we train an LDA classifier (`solver=eigen`, Ledoit-Wolf automatic shrinkage [71]) under 5-fold stratified cross-validation on 5,000 traces per memory level, with features standardised (zero mean, unit variance) independently per fold. Traces are intra-class aligned via two-pass cross-correlation prior to the fold split; since alignment is computed per class over the full dataset using only within-class statistics, it cannot introduce cross-class distributional bias into any test partition. The LDA confusion matrix (Fig. 6a) with n_{PoI}=128 shows 99.64 % accuracy,



(a) Privileged — $D(t)$ (b) Privileged — Welch $\mathcal{T}(t)$

(c) Unprivileged — $D(t)$ (d) Unprivileged — Welch $\mathcal{T}(t)$

Fig. 5: Time-resolved separability for privileged and unprivileged traces. Left: KS statistic $D_{a,b}(t)$; markers denote Bonferroni-significant samples. Right: Welch $\mathcal{T}(t)$.



(a) Privileged instr. (b) Unprivileged instr.

Fig. 6: LDA confusion matrix on POI-restricted features; 5,000 traces per class; mean over 5 stratified folds.

confirming that memory-hierarchy-dependent patterns are directly distinguishable through PDN signatures.

C. Probing with unprivileged instruction-only

To evaluate practicality, we design a second probe that uses only unprivileged instructions. It mirrors the privileged probe almost exactly: the memory-barrier sequence is replaced by a chain of register-based dependencies, and the surrounding delay spins **(Steps 1 and 5)** are made dependent on the same registers to improve the isolation. This dependency-based structure follows prior designs for EM probing [65]. The TDC trigger and POI window remain identical to Section III-B3. Fig. 7 shows the complete unprivileged code sequence:

- **Step 2**: introduces a true address dependency using `eor r7, sp, sp` and `add r5, r0, r7`. The result is data-dependent on `sp`, enforcing in-order execution of `ldrb` and preventing speculative address generation.
- **Step 3**: performs the targeted memory access. The single `ldrb r6, [r5]` triggers the targeted memory operation (L1 hit, L2 hit, or DRAM access).
- **Step 4**: runs a short register-only loop composed of `EOR/ORR/ROR/MUL` operations, all driven by `r6` and `r5`. This keeps the integer pipeline active and extends the power transient without performing any additional memory accesses.

Applying the same analysis pipeline, including POI localization (Fig. 5c-5d), separability metrics, and LDA, produces nearly identical class distinctions (Fig. 6b). As shown in

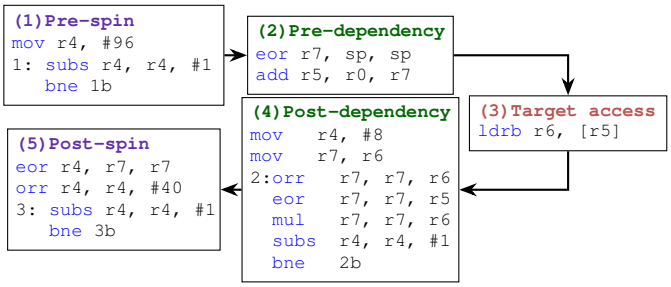


Fig. 7: Unprivileged code with dependency-based isolation.

TABLE II: LDA comparison of privileged vs. unprivileged probes (5,000 traces/class, 5-fold CV).

Probe	Mean accuracy	Mutual information
Privileged assembly (baseline)	**99.64%**	1.55 bits
Unprivileged instruction-only	96.48%	1.34 bits

Table II, with n_{PoI}=512, the unprivileged probe achieves 96.5% accuracy demonstrating that memory-service levels can be inferred using only user-mode code.

IV. Flush+Power: Same-Core Cache Attack

This Section describes *Flush+Power*, a profiled, timerless cache-probing primitive that exploits PDN power transients generated by a fenced memory load. The attacker performs a local flush, lets the victim access data, and issues a probing load that triggers a short measurement window. A lightweight classifier identifies the load service source (L1, L2, DRAM), providing **line-level resolution** in shared caches and enabling **single-trace recovery** of a victim's cache line usage.

A. Threat model

The attacker is a low-privilege co-tenant executing untrusted code on the same SoC as the victim. As in Flush+Reload [72], attacker and victim share the cache hierarchy, and the victim's code or data is mapped into a shared, file-backed page that the attacker can legally access. The attacker has no timers, PMU access, or privileged interfaces, and can only issue cache-maintenance instructions that flush the local data cache. Victim activity may populate shared cache levels, and the attacker can subsequently probe candidate addresses while collecting short physical measurements. The side channel consists solely of the observed PDN transient, making the distinguishing of memory levels from this signal the core primitive contribution.

B. Attack primitive

Flush+Power consists of three short, deterministic steps that together expose if a chosen memory address is serviced from L1 or from deeper levels of the hierarchy. Each trace follows three atomic steps as shown in Fig. 8 and Algorithm 1:

Algorithm 1 Description of Flush+Power

Require: Set of monitored lines ℓ
1: FLUSH
2: $C \leftarrow$ VICTIM_ACTIVITY
3: POWER_PROBE_LINE(ℓ)

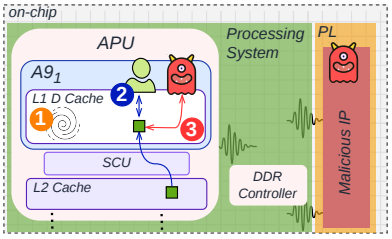


Fig. 8: Overview of the *Flush+Power* primitive: ① **Flush**, ② **Victim activity**, ③ **Power probe**.

1) **Flush.** The attacker starts from a clean slate by flushing its local data cache. This guarantees that any subsequent hit on a monitored line must originate from the victim's activity, not from stale attacker state.
2) **Victim activity.** The victim executes normally and may access any subset of the shared memory hierarchy. These accesses can bring specific lines into L1 or L2, setting the microarchitectural state the attacker aims to recover.
3) **Power Probe.** The attacker performs a single, fenced load to the monitored address. This load generates a short PDN transient whose shape reflects the servicing level: L1, L2, or DRAM. A sensor triggered by the load records this transient, and a pre-built classifier (Section III) determines the memory level for the probed address.

By repeating the power-probe step on a set of candidate addresses, which can be obtained using methods from previous works [3], [65], the attacker reconstructs exactly which lines were touched by the victim. As in the classical Flush+Reload attack, the attack infers victim-dependent memory usage; however, the discriminant here is the physical power signature of the load, rather than its timing.

C. Application to AES T-table attack

We apply *Flush+Power* to a custom, table-driven implementation of AES first round running on the same core as the attacker. In this setting, a single L1 flush, one encryption, and one fenced probe are enough to recover the cache line accessed by the first-round T-table lookup. Because each line stores eight entries, identifying the accessed line directly leaks the five most significant bits of the lookup index from a single PDN trace using a pretrained memory-level discriminant.

Experimental setup: We use the bare-metal configuration of Section III. We assume a chosen-plaintext attack model in which the adversary can repeatedly trigger encryptions of controlled plaintexts under a fixed, secret key. In each run, the AES T-table is brought into the shared L2 as a deliberate remediation against prefetches and EM-based exploitation of off-chip DRAM accesses [65]. On this platform, the T_0 table contains 256 words arranged in 32 cache lines of 8 words each. Distinguishing L1 from L2 hits reveals the line index $\ell = \lfloor i/8 \rfloor$, although the intra-line byte offset remains unresolved. We therefore apply the *Flush+Power* primitive exactly as defined in Section IV-B;limited to L1 and L2 hits.

To evaluate our primitive, we introduce Algorithm 2. First, the attacker issues a user-level L1 flush. Then, the victim

encrypts a chosen plaintext under an unknown key, which hits exactly one line of the T_0 table during the first round. Finally, the attacker iterates over all 256 indices with a stride S issuing the fence-load-fence pattern. Every load automatically triggers TDC capture. We perform the validation using two probing strategies, corresponding with two Algorithm 2 instantiations: a *byte-level sweep* for $S = 1$, and a *line-level sweep* for $S = 8$. We account for noise by introducing another parameter, R, that specifies the number of times we repeat the probing process for the same address. The lower the value of R, the stronger the threat model, with $R = 1$ the strongest attacker requiring only a single activation of victim's activity.

Algorithm 2 Index sweep for Flush+Power.

Require: Repetition count R;

Step size $S \leftarrow \begin{cases} 1 & \text{(byte-level sweep)} \\ 8 & \text{(line-level sweep)} \end{cases}$

1: for $r = 1$ to R do
2: FLUSH
3: Victim AES encryption
4: for $i = 0$ to 255 step S do
5: $p \leftarrow \&T_0[i]$ ▷ candidate address; $\ell = i/8$ for line-granularity
6: POWER_PROBE_LINE(p)

The profiling dataset and preprocessing pipeline are identical to those of Section III with $10,000$ traces per class (limited to L1 and L2), collected with the unprivileged patterns on random cache locations. For the attack phase, we apply the created power profile on T-table memory accesses.

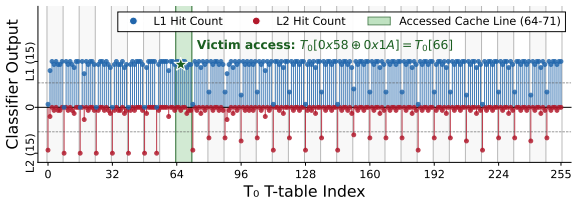
To evaluate Flush+Power attack accuracy, we report *balanced accuracy* (BA). The LDA classifier labels each of the N cache lines as *L1-hit* (positive) or *L2-hit* (negative). BA equally weights the per-class recall and is therefore immune to the class imbalance inherent in our setting:

$$\text{BA} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right) = \frac{1}{2}(\text{TPR} + \text{TNR}),$$

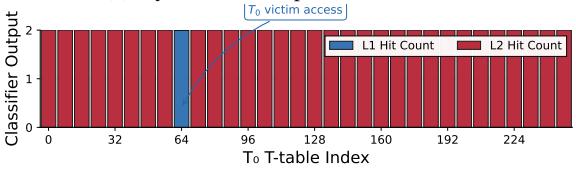
where TPR is the fraction of sweeps in which the true L1-hit line is correctly identified and TNR the fraction of the $N-1$ L2-hit lines correctly classified as such. Because each AES access after the flush produces exactly one L1 hit among N lines, a trivial always-L2 classifier attains raw accuracy $(N-1)/N \approx 1$ yet BA $= 0.5$; BA is thus the appropriate metric for this task.

D. Results

Our LDA classifier achieves BA $= 0.861$. The gap relative to the profiling accuracy in Table II stems from a distribution shift [73] between the controlled profiling phase and the attack phase. The first strategy ($S = 1$) probes *every* byte of the AES T_0 table, producing a sequential probing of each T_0 table index. Fig. 9a shows the output of the classifier for L1 (blue) and L2 (red) hits, for $R = 15$. As expected, each probe falling at the beginning of a line produces an L1 miss, while the subsequent 7 produce a hit, completing a full line (8 bytes). On the contrary, when the victim accesses the table on its side (green), the probe produces an L1 hit because the full line was already brought in by the victim's activity.



(a) Byte-level sweep: $S = 1$, $R = 15$



(b) Line-level sweep: $S = 8$, $R = 2$

Fig. 9: T_0 table index probing classification using Algorithm 2.

This demonstrates that *a byte-level sweep still achieves a cache-line-level granularity*. We can observe noise (hit counts different from 15 or 0). We hypothesize that, as consecutive indices map to the same cache line, effects such as prefetching or replacement may introduce additional variability in the traces, thereby contributing to the observed noise.

The second strategy ($S = 8$) probes only the *first entry* of each of the 32 cache lines, avoiding consecutive accesses to same-set values and reducing prefetch or replacement effects. This enabled us to evaluate the reduction of repetitions down to $R = 1$, *achieving a single-trace attack setting* with improved classifier accuracy. Fig. 9b illustrates the results: each probe in a line results in an L2 hit (red), except for line 64, which is served as expected from L1 (blue) after victim activity. The impact of noise in this method is greatly diminished compared to Fig. 9a, where some predicted L2 hits resulted in L1 hits. This method yields a much cleaner separability, with the correct line ranked top-1 among all 32 candidates after a single flush-encrypt-probe sequence from Algorithm 1.

Overall, this lightweight experiment requires only one user-level L1 flush, one victim encryption, and one fenced probe per tested line, while the TDC records a short PDN window of 951 samples. In our platform configuration (32 cache lines of 8 words each), each probe directly reveals the accessed line $\ell = \lfloor i/8 \rfloor$, that is, the five most significant bits of the lookup index. A single trace per index achieves about 86% accuracy; a redundant second trace eliminates residual errors. Two complete sweeps complete in about 7s.

V. SNOOPYPOWER: SCU COHERENCE LEAKAGE

This section introduces *SnoopyPower*, a profiled and timerless primitive that extends the microbenchmarking approach of *Flush+Power* (Section IV) to the discrimination of coherence states in multicore environments. When a core loads a shared cache line, the request is either served locally or, if a peer holds the line, via an SCU-initiated snoop. *SnoopyPower* separates these access paths with their characteristic PDN signatures using a lightweight classifier, enabling **coherence-state resolution** without relying on timers, PMUs, or privileged code.

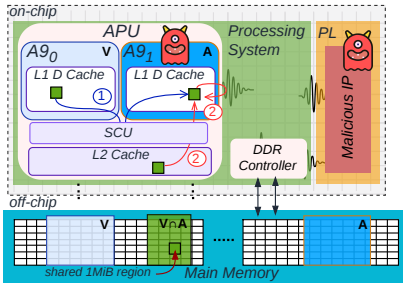


Fig. 10: *SnoopyPower* threat model. Two tenants run on separate cores with private DRAM regions and a shared page (**V**∩**A**). During a probe, the SCU serves the access via either ① **peer-L1 snoop** or ② **self-path load**. The TDC records the resulting PDN signatures.

A. Threat model

We consider two isolated tenants, a victim (V) on core 0 and an attacker (A) on core 1 (Fig. 10) each with a private L1 cache and a shared L2 connected through a centralized SCU. Both tenants map a common page and private DRAM regions. The shared page is configured as *Normal, Shareable, write-back, write-allocate*, ensuring that all accesses participate in hardware coherence. Symmetric Multiprocessing (SMP) is enabled to activate cache coherency among L1 data caches [21], and firmware broadcasts are disabled; therefore, the SCU mediates all cross-core interventions. The attacker is fully unprivileged: can read from the shared region and trigger the user-accessible TDC sensor, but has no PMU counters or external probes. Their capabilities match those in *Flush+Power* (Section IV), except they cannot issue cache-maintenance instructions, accounting for a stronger threat model.

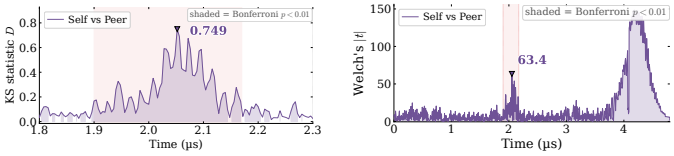
Leakage arises because an attacker's load to a shared line is resolved either through the *self path* (local L1 or shared L2) or through a *peer-L1 snoop* when the victim holds a modified copy. While these coherence resolutions differ by less than one cycle in latency and are indistinguishable from software on Arm [18], they actually induce distinct PDN signatures. Classifying each captured waveform as one of two coherence classes $\mathcal{C}_{\text{self}}$ or $\mathcal{C}_{\text{peer}}$, reveals if the victim has recently written the probed line, enabling coherence-level activity monitoring and address discovery without privileged access.

B. Attack primitive

SnoopyPower operates at the level of coherence rather than the cache hierarchy. It leverages the power difference stemming from whether a probed line is served through a **peer-L1 snoop** ①, or via the **self path** ②, as illustrated in Fig. 10. *SnoopyPower* is composed of two simple steps:

(i) Victim activity. The victim proceeds normally. If they wrote the target line, it now holds the only modified copy in the system; if not, the line resides cleanly in either the attacker's local L1 or the shared L2.

(ii) Coherence probe. The attacker performs a fenced load (Fig. 7) to the monitored address. The SCU resolves this access through the self path or a peer-L1 snoop. The TDC, triggered with the fenced load execution, records the PDN signatures. The attacker uses a lightweight classifier on the recorded trace to determine the coherence decision for the probed address. By repeating this probe across multiple candidate lines, the attacker recovers which line the victim has recently modified.



(a) Self/Peer — $D(t)$ (b) Self/Peer — Welch $\mathcal{T}(t)$

Fig. 11: Time-resolved KS and Welch separability between $\mathcal{C}_{\text{self}}$ (self-load) and $\mathcal{C}_{\text{peer}}$ (peer-L1) coherence paths.

C. Experimental setup

All SnoopyPower experiments run *bare metal* on the same Zynq-7000 platform used for Flush+Power. Each Cortex-A9 runs an independent SMP binary, with attacker pinned to Core 0 and victim to Core 1. We disable DVFS and mask interrupts to maintain a stable baseline. We characterize coherence behavior and evaluate SnoopyPower on unseen addresses as follows: the victim either stores to a chosen line-aligned address or remains idle, controlling if the attacker load will trigger a peer-L1 snoop or the self path; the attacker continuously issues fenced `LDRB` instructions to this same address. Each load triggers a TDC capture of a 2-3 μs.

D. Coherence-path characterization

We define two coherence classes: $\mathcal{C}_{\text{peer}}$ (peer-L1 snoop) and $\mathcal{C}_{\text{self}}$ (self-path load). In the online phase, we collect $T_c = 5,000$ traces per class, labeling each waveform via ground truth obtained from the SCU's `snoop-linefill` PMU. The offline phase is similar to Section III. As seen in Fig. 11, both KS and Welch exhibit a burst of discriminative activity within a sub-microsecond window, with the KS profile peaking at $D_{\max}$=0.749 during the coherence transaction. We sweep with $n_{\text{PoI}} \in [8, \ldots, 125]$. Each trace is then projected to $x \in \mathbb{R}^{n_{\text{PoI}}}$ and class-conditional Gaussian templates are estimated:

$$\hat{\mu}_c = \frac{1}{T_c}\sum_{j=1}^{T_c} x_j^{(c)}, \qquad \hat{\Sigma}_c = \frac{1}{T_c - 1}\sum_{j=1}^{T_c}(x_j^{(c)} - \hat{\mu}_c)(x_j^{(c)} - \hat{\mu}_c)^\top,$$

with Ledoit–Wolf shrinkage on $\hat{\Sigma}_c$ [71]. Given equal priors, QDA assigns each trace the log-likelihood ratio

$$s(x) = \log p(x \mid \mathcal{C}_{\text{peer}}) - \log p(x \mid \mathcal{C}_{\text{self}}).$$

The best configuration is QDA with n_{PoI}=128, yielding a 5-fold accuracy of 0.99.

To validate the classifier, we first verify that the profiled Gaussian templates reliably distinguish the two coherence paths. These templates are built once and used unchanged during the inference phase. Table IIIa shows the classifier results, which achieve accuracy = 0.986, F_1 = 0.99, and AUC = 0.999, confirming that the templates capture stable

Algorithm 3 Address discovery on unseen lines

Require: candidate set C, Address $\alpha \in C$

Victim core	**Attacker core**
1: while true do	1: while true do
2: **write**(α)	2: for a in C do
	3: for $x = 0$ to r do
	4: **measure**(**a**)

PDN differences between the two coherence resolutions despite their nearly identical microarchitectural latency, and that both mean and covariance differ systematically across classes.

TABLE III: Coherence path QDA performance classification.

(a) Characterization

Metric	Result
Accuracy	0.99
F_1 macro	0.99
AUC	0.999

(b) 100-line scan

Metric	Result
Peer L1 found	index 31
Mean confidence	0.999
Wall clock	<2 min

E. SnoopyPower evaluation on unseen addresses

To assess the address-agnosticity of our model, we evaluate it on a completely new set of cache-line addresses. In this experiment, the attacker must guess which line is written by the victim. For this attack, BA is computed per individual cache load: the QDA classifier labels each load as *snoop* (positive) or *self* (negative). Our classifier achieves BA $= 0.795$. The reduction in profiling accuracy (Table IIIa) reflects a distribution shift since the 100 unseen addresses differ from the fixed training pair. We run Algorithm 3 with C a set of 100 candidate addresses $\mathcal{A} = \{a_1, \ldots, a_N\}$, different from the training set. The victim uses one address $\alpha \in C$, and the attacker tries to guess it. On one core, the victim repeatedly writes to α. On another core, the attacker repeatedly iterates through the candidate lines C: for each candidate $a \in C$, the attacker acquires r traces and aggregates their QDA scores as:

$$\bar{s}(a) = \frac{1}{r} \sum_{k=1}^{r} s(x_{a,k}).$$

Classification uses a threshold τ, with $\bar{s}(a) > \tau$ indicating $\mathcal{C}_{\text{peer}}$. Scanning $N = 100$ lines with $r = 5$ probes per line identifies the correct victim-touched line in under 2 minutes.

As shown in Fig. 12, a single address exceeds the peer-L1 threshold despite a few single-trace misclassifications (up to two for non-victim lines and 4/5 correct votes for the true line). The attack remains reliable with $r = 4$; however, reducing to $r = 3$ leaves too much variance and causes miss detection. Larger values offer no measurable benefit. Aggregation therefore suppresses the distribution shift observed on the unseen addresses, and $r = 5$ provides a practical and robust setting.

VI. Discussion

Our results show that a TDC can transform the shared PDN of a SoC into an observability channel that discriminates timing-indistinguishable cache and coherence events, without relying on performance counters or external physical probes. We now discuss how these findings extend to realistic deployments and what they imply for SoC and cloud-FPGA security.

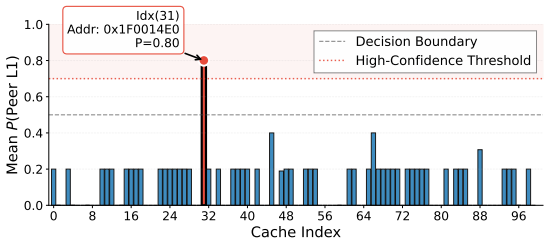


Fig. 12: Aggregated scores $\bar{s}(a)$ for an unseen 100-line set with $r = 5$ probes per address. One line (index 31) crosses the peer-L1 threshold.

A. On the Flush+Power primitive

While relying on a different signal source, Flush+Power achieves an efficiency comparable to Flush+Reload in shared-memory and same-core settings. In AES experiments, a single trace reveals the five MSBs of each lookup index, with the remaining bits recoverable using a few chosen plaintexts. Although our implementation uses an unprivileged data-cache flush instruction, the primitive extends naturally to an *Evict+Power* variant using eviction sets [3], [74].

B. On the SnoopyPower primitive

SnoopyPower provides the coherence-level counterpart to the line-level cache resolution of Flush+Power. Although these coherence resolutions differ by less than one cycle in latency, they produce distinct PDN signatures, enabling discrimination in a cross-core setting. Building on the SnoopyPower primitive, the attacker can operate in two modes. A *passive* mode repeatedly probes the same line, producing a timerless peer/self sequence that reflects the victim's access cadence. An *active* mode arises when the mapping is writable: attacker and victim alternately modify the line, creating a controlled coherence *ping-pong* pattern. Even for read-only mappings, the victim's writes are enough to generate peer-L1 signatures.

C. Portability

To evaluate portability, models are trained on a single Zybo Z7-20 board and evaluated, without modification, on two distinct boards of the same model. For Flush+Power, cross-board transfer degrades balanced accuracy to 73.2%. Despite this drop, the classifier still reliably identifies the L1-hit line in the end-to-end AES T-table attack. For SnoopyPower, the same cross-board evaluation degrades per-load balanced accuracy from 79.5% to 76.6%. Score aggregation over r probes fully absorbs this shift, preserving address-level recovery. Both attacks thus generalise across hardware instances.

As previous work has shown, the TDC's position in the PL can have a non-negligible impact on captured traces, thereby affecting attack performance [75], [76], which, in our case, translates into an impact on the model's accuracy. Consequently, we evaluate the accuracy of our LDA by varying its location on the same board. Using the same setup from Section III, we evaluate 13 TDC placements uniformly distributed across the PL with 3,000 traces per class per position. From these experiments, we found that position

X26Y16, adjacent to the PS boundary, ranks highest with 98% accuracy. Placements closer to the PS outperform edge positions ($\mathrm{X} = 108$), consistent with shorter PDN return paths closer to the cache subsystem yielding stronger coupling. Notably, even the weakest placement (X108Y48) achieves 95.6% classification accuracy, demonstrating the attack can be effectively mounted from any location within the PL.

D. Impact of OS and noise

To quantify resilience to OS noise, we provide a PoC implementation running under PynqLinux 3.0 (custom PetaLinux kernel by Xilinx/AMD), based on Ubuntu 22.04, as distributed in the official AMD PYNQ images used in this work, version 3.0. The attacker does not have control over the CPU frequency. The pipeline, remains unchanged except for minor engineering adjustments; only the training data must match the OS noise. When trained on $30,000$ traces, our Section III LDA achieves a 98.7% accuracy in the three-class characterization (L1/L2/DRAM). Section V QDA yields 94.3% accuracy between self loads and snoops after retraining on $10,000$ traces. These results show that the attack remains effective in an OS environment with retraining only.

We have evaluated the Section III characterization classifier in our OS environment while introducing artificial noise on a separate core to emulate realistic interference. Under OS-level scheduling noise with concurrent `stress-ng` [77] workloads (L2 sweeps, ALU, memcpy), the original QDA classifier accuracy drops to 6.4%). However, if we retrain our classifier using $30,000$ traces obtained on this noisy environment, we regain an accuracy of 96.4% on the three-class characterization (L1/L2/DRAM). It is not possible to assess SnoopyPower under these conditions because the evaluation platform features only two cores and does no hyperthreading, so introducing noise would require removing either the victim or the attacker.

E. Threat model

The multi-tenant threat model is motivated by recent literature already identifying FPGA-based cloud deployments at board [78] and chip-integration levels [79], including new AMD Versal devices [80], [81]. Other motivations include third-party IP cores in bitstreams as indirect sharing, and malicious PL IP blocks acting as attackers [82]. We used a SoC-FPGA as a PoC of the MPSoC model, representing the *strongest PL-CPU coupling* via the shared PDN. Prior work shows remote power leakage across chips on the same board [83] and even across boards sharing a PSU [84]. We conclude that our results provide a higher bound on cross-boundary leakage. Prior work demonstrates how leakage captured by TDC sensors remains exploitable across FPGA generations, including Ultrascale+ [85]. Additional works evaluate active fence countermeasures on platforms like ZCU104 [86], while others show how to identify co-tenant activity on cloud-grade devices like XCVU9P in AWS EC2 F1 instances to recover secret keys from different IP cores [87]. Moreover, successful attacks are reported across Super Logic Regions on Ultrascale+ devices in AWS/Huawei clouds, despite logical and physical isolation between attacker and victim [88].

F. Beyond timing: timerless microarchitectural observability

Our results demonstrate that PDN sensing offers a new dimension for microarchitectural observability. Even under timing-indistinguishable microarchitectural events, their power activity remains different. This enables timerless observation of microarchitectural behavior, bypassing timing-based defenses. More broadly, the HW/SW model of SoC-FPGAs becomes a real instrument for exploring leakages that are otherwise inaccessible. Any other hardware structure with distinguishable power signatures may be observable. As heterogeneous processors integrate more complex memory fabrics, the impact of power-based observability is likely to increase.

G. Mitigation directions

Limiting PDN leakage requires coordinated measures across hardware, software, and fabric management. In hardware, stronger CPU/PL power-domain separation, local regulation, or reduced cross-core coherence activity may lower microarchitectural events visibility in the PDN. At the system level, avoiding shared pages across trust boundaries, isolating sensitive workloads, and moderating PL usage during critical phases can impact PDN leakage. Malign PL usage is challenging, as power-sensing circuits are difficult to distinguish from benign monitoring IP [51], [54]. Literature explores bitstream validation, sensor bandwidth limits, or spatial separation between untrusted PL and CPU. As CPU-FPGA integration tightens, the PL should be considered an active element in side-channel threat models, rather than a passive accelerator.

VII. Conclusion

This work showed that a compact TDC placed in the PL can turn the shared PDN of an Arm SoC-FPGA into a precise observation window for microarchitectural behavior. Short, auto-triggered captures around a fenced load reliably differentiate L1, L2, and DRAM servicing paths, as well as separate peer-L1 snoops from self-path coherence events, distinctions that remain invisible to software timing. Using standard side-channel analysis metrics, we quantified the distinguishability of these PDN signatures and demonstrated that lightweight statistical models can effectively exploit them. Building on this characterization, we introduced two new attack primitives, *Flush+Power* and *SnoopyPower*, which resolve cache and coherence paths without timers and rely only on logic that could appear as ordinary PL IP. More broadly, our results show that CPU-PL power coupling enables observability channels that persist even when software measurement interfaces are restricted, highlighting PDN effects as a major concern in the security evaluation of future SoC-FPGA platforms.

Acknowledgment

Work partially funded by the ANR within the framework of the PIA EUR CyberSchool project (ANR-18-EURE-0004).

REFERENCES

[1] H. Fang, S. Dayapule, F. Yao, M. Doroslovacki, and G. Venkataramani, "Prefetch-guard: Leveraging hardware prefetchers to defend against cache timing channels (short paper)," in *Proceedings of IEEE Symposium on Hardware Oriented Security and Trust, IEEE*, 2018.

[2] H. Fang, S. S. Dayapule, F. Yao, M. Doroslovački, and G. Venkataramani, "Prodact: Prefetch-obfuscator to defend against cache timing channels," *International Journal of Parallel Programming*, vol. 47, no. 4, pp. 571–594, 2019.

[3] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "{ARMageddon}: Cache attacks on mobile devices," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 549–564.

[4] M. Schwarz, C. Maurice, D. Gruss, and S. Mangard, "Fantastic timers and where to find them: High-resolution microarchitectural attacks in javascript," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 247–267.

[5] D. Katzman, W. Kosasih, C. Chuengsatiansup, E. Ronen, and Y. Yarom, "The gates of time: Improving cache attacks with transient execution," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 1955–1972.

[6] A. Purnal, M. Bognar, F. Piessens, and I. Verbauwhede, "Showtime: Amplifying arbitrary cpu timing side channels," in *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, 2023, pp. 205–217.

[7] T. Chen, Y.-a. Tan, W. Li, Z. Ci, and N. Shi, "Toward secure program execution in multi-tenant cloud fpga environments: T. chen et al." *The Journal of Supercomputing*, vol. 81, no. 8, p. 871, 2025.

[8] D. G. Mahmoud, V. Lenders, and M. Stojilović, "Electrical-level attacks on cpus, fpgas, and gpus: Survey and implications in the heterogeneous era," *ACM Computing Surveys (CSUR)*, vol. 55, no. 3, pp. 1–40, 2022.

[9] H. Naghibijouybari, E. M. Koruyeh, and N. Abu-Ghazaleh, "Microarchitectural attacks in heterogeneous systems: A survey," *ACM Comput. Surv.*, vol. 55, no. 7, Dec. 2022.

[10] F. Yao, M. Doroslovacki, and G. Venkataramani, "Are coherence protocol states vulnerable to information leakage?" in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 168–179.

[11] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual international cryptology conference*. Springer, 1999, pp. 388–397.

[12] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer, 2007.

[13] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2004, pp. 16–29.

[14] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "Platypus: Software-based power side-channel attacks on x86," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 355–371.

[15] Y. Wang, R. Paccagnella, E. T. He, H. Shacham, C. W. Fletcher, and D. Kohlbrenner, "Hertzbleed: Turning power {Side-Channel} attacks into remote timing attacks on x86," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 679–697.

[16] M. Zhao and G. E. Suh, "FPGA-Based Remote Power Side-Channel Attacks," in *2018 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA: IEEE, May 2018, pp. 229–244.

[17] J. Gravellier, J.-M. Dutertre, Y. Teglia, P. L. Moundi, and F. Olivier, "Remote side-channel attacks on heterogeneous soc," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2019, pp. 109–125.

[18] S. Deng, N. Matyunin, W. Xiong, S. Katzenbeisser, and J. Szefer, "Evaluation of cache attacks on arm processors and secure caches," *IEEE Transactions on Computers*, vol. 71, no. 9, pp. 2248–2262, 2021.

[19] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," *ACM SIGARCH Computer Architecture News*, vol. 18, no. 2SI, pp. 364–373, 1990.

[20] M. Dubois, M. Annavaram, and P. Stenström, *Parallel computer organization and design*. cambridge university press, 2012.

[21] *ARM Cortex-A9 Technical Reference Manual*, ARM Ltd., 2011, revision r4p1, available from ARM Developer documentation. [Online]. Available: https://developer.arm.com/documentation/100511/latest/

[22] *ARM L2C-310 Level 2 Cache Controller Technical Reference Manual*, Arm Ltd., 2011, document ID: DDI 0246 (various revisions). [Online]. Available: https://developer.arm.com/documentation/

[23] M. Weiß, B. Heinz, and F. Stumpf, "A cache timing attack on aes in virtualization environments," in *International Conference on Financial Cryptography and Data Security*. Springer, 2012, pp. 314–328.

[24] R. Spreitzer and B. Gérard, "Towards more practical time-driven cache attacks," in *IFIP International Workshop on Information Security Theory and Practice*. Springer, 2014, pp. 24–39.

[25] R. Spreitzer and T. Plos, "On the applicability of time-driven cache attacks on mobile devices," in *International Conference on Network and System Security*. Springer, 2013, pp. 656–662.

[26] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ flush: A fast and stealthy cache attack," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 279–299.

[27] B. Lapid and A. Wool, "Navigating the samsung trustzone and cache-attacks on the keymaster trustlet," in *Computer Security*, J. Lopez, J. Zhou, and M. Soriano, Eds. Cham: Springer International Publishing, 2018.

[28] Y. Guo, A. Zigerelli, Y. Zhang, and J. Yang, "Adversarial prefetch: New cross-core cache side channel attacks," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1458–1473.

[29] Z. Kou, S. Sinha, W. He, and W. Zhang, "Attack directories on arm big. little processors," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[30] L. Bossuet and E. M. Benhani, "Performing cache timing attacks from the reconfigurable part of a heterogeneous soc—an experimental study," *Applied Sciences*, vol. 11, no. 14, p. 6662, 2021.

[31] H. Lee, S. Jang, H.-Y. Kim, and T. Suh, "Hardware-based flush+ reload attack on armv8 system via acp," in *2021 International Conference on Information Networking (ICOIN)*. IEEE, 2021, pp. 32–35.

[32] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Communications of the ACM*, vol. 62, no. 2, pp. 48–60, 2019.

[33] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th annual international symposium on Computer architecture*, 2011, pp. 365–376.

[34] A. S. Corporation, "Achronix vectorpath accelerator card achieves pcie 5.0 ×16 compliance at 32 gt/s," Press Release, May 2023. [Online]. Available: https://www.achronix.com/press-releases/achronix-vectorpath-accelerator-card-achieves-pcie-50-16-compliance-32-gts

[35] E. F. Kfoury, S. Choueiri, A. Mazloum, A. AlSabeh, J. Gomez, and J. Crichigno, "A comprehensive survey on smartnics: Architectures, development models, applications, and research directions," *IEEE Access*, 2024.

[36] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq book: embedded processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 all programmable SoC*. Strathclyde Academic Media, 2014.

[37] S. Pereira, T. Gomes, J. Cabral, and S. Pinto, "Beneath the fabric: A survey on threats and opportunities in reconfigurable systems," *Available at SSRN 5669594*, 2025.

[38] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital integrated circuits*. Prentice hall Englewood Cliffs, 2002, vol. 2.

[39] F.-X. Standaert, T. G. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2009, pp. 443–461.

[40] S. Picek, G. Perin, L. Mariot, L. Wu, and L. Batina, "SoK: Deep Learning-based Physical Side-channel Analysis," *ACM Computing Surveys*, vol. 55, no. 11, pp. 227:1–227:35, Feb. 2023.

[41] G. Le Gonidec, G. Bouffard, J.-C. Prevotet, and M. Méndez Real, "Do not trust power management: A survey on internal energy-based attacks circumventing trusted execution environments security properties," *ACM Transactions on Embedded Computing Systems*, 2025.

[42] A. Kogler, J. Juffinger, L. Giner, L. Gerlach, M. Schwarzl, M. Schwarz, D. Gruss, and S. Mangard, "{Collide+ Power}: Leaking inaccessible data with software-based power side channels," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 7285–7302.

[43] C. Liu, A. Chakraborty, N. Chawla, and N. Roggel, "Frequency throttling side-channel attack," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1977–1991.

[44] D. R. Dipta and B. Gulmezoglu, "Df-sca: Dynamic frequency side channel attacks are practical," in *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022, pp. 841–853.

[45] J. Gravellier, J.-M. Dutertre, Y. Teglia, and P. L. Moundi, "Sideline: How delay-lines (may) leak secrets from your soc," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2021, pp. 3–30.

[46] Y. Michalevsky, A. Schulman, G. A. Veerapandian, D. Boneh, and G. Nakibly, "{PowerSpy}: Location tracking using mobile device power analysis," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 785–800.

[47] K. Taneja *et al.*, "Hot pixels: Frequency, power, and temperature sensors as side channels," in *32nd USENIX Security Symposium*, 2023.

[48] M. Oberhuber, M. Unterguggenberger, L. Maar, A. Kogler, and S. Mangard, "Power-related side-channel attacks using the android sensor framework," in *NDSS*, 2025.

[49] M. Kawser Ahmed, M. Panoff Kealoha, J. Mandebi Mbongue, S. K. Saha, E. Nghonda Tchinda, P. E. Mbua, and C. Bobda, "Multi-tenant cloud fpga: A survey on security, trust, and privacy," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 18, no. 2, pp. 1–44, 2025.

[50] J. Szefer and R. Tessier, *Security of FPGA-Accelerated Cloud Computing Environments*. Springer, 2024.

[51] D. Jayasinghe, B. Udugama, and S. Parameswaran, "1lutsensor: Detecting fpga voltage fluctuations using lookup tables," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 1, pp. 51–86, 2024.

[52] B. Udugama, D. Jayasinghe, H. Saadat, A. Ignjatovic, and S. Parameswaran, "Viti: A tiny self-calibrating sensor for power-variation measurement in fpgas," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 657–678, 2022.

[53] A. Fellah-Touta, L. Bossuet, and C. A. Lara-Nino, "A lightweight non-oscillatory delay-sensor for remote power analysis," in *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2024, pp. 343–348.

[54] X. Zhang, J. Zou, Z. Zhang, Q. Shen, Y. Gao, J. Cui, Y. Feng, Z. Wu, and D. Abbott, "Muxleak: Exploiting multiplexers as a power side channel against multi-tenant fpgas," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.

[55] D. Spielmann, O. Glamočanin, and M. Stojilović, "Rds: Fpga routing delay sensors for effective remote power analysis attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 543–567, 2023.

[56] Y. Zhu, J. Zhou, and P. Zhou, "Exploring remote power attacks targeting parallel data encryption on multi-tenant fpgas," in *Proceedings of the Great Lakes Symposium on VLSI 2023*, 2023, pp. 57–62.

[57] O. Glamocanin, L. Coulon, F. Regazzoni, and M. Stojilovic, "Are cloud fpgas really vulnerable to power analysis attacks?" in *Proceedings Of The 2020 Design, Automation & Test In Europe Conference & Exhibition (Date 2020)*. IEEE, 2020, pp. 1007–1010.

[58] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier, "Remote power side-channel attacks on bnn accelerators in fpgas," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1639–1644.

[59] S. Tian, S. Moini, D. Holcomb, R. Tessier, and J. Szefer, "A practical remote power attack on machine learning accelerators in cloud fpgas," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[60] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. A. Al Faruque, "Stealing neural network structure through remote fpga side-channel analysis," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4377–4388, 2021.

[61] G. Lomet, R. Salvador, B. Colombier, V. Grosso, O. Sentieys, and C. Killian, "Side-channel extraction of dataflow ai accelerator hardware parameters," in *2025 IEEE 31st International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2025, pp. 1–7.

[62] O. Glamočanin, S. Shrivastava, J. Yao, N. Ardo, M. Payer, and M. Stojilović, "Instruction-level power side-channel leakage evaluation of soft-core cpus on shared fpgas," *Journal of Hardware and Systems Security*, vol. 7, no. 2, pp. 72–99, 2023.

[63] Y. Jin, M. Sun, D. Wang, P. Qiu, Y. Zhang, and S. Deng, "Ghostcache: Timer- and counter-free cache attacks exploiting weak coherence on risc-v and arm chips," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '25. Association for Computing Machinery, 2025, p. 3795–3809.

[64] F. Thomas, M. Torres, D. Moghimi, and M. Schwarz, "Exfilstate: Automated discovery of timer-free cache side channels on arm cpus," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '25. Association for Computing Machinery, 2025, p. 2564–2578. [Online]. Available: https://doi.org/10.1145/3719027.3765061

[65] J. Maillard, T. Hiscock, M. Lecomte, and C. Clavier, "Cache side-channel attacks through electromagnetic emanations of dram accesses," *Cryptology ePrint Archive*, 2023.

[66] X. Zhang, Y. Xiao, and Y. Zhang, "Return-oriented flush-reload side channels on arm and their implications for android devices," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. Association for Computing Machinery, 2016, p. 858–870. [Online]. Available: https://doi.org/10.1145/2976749.2978360

[67] I. Xilinx, *Arm Cortex-A9 Processor Cache Functions*, Standalone Library API Reference, 2023, [Online; accessed 17-Nov-2025]. [Online]. Available: https://docs.amd.com/r/2023.1-English/oslib_rm/Arm-Cortex-A9-Processor-Cache-Functions

[68] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3-4, pp. 591–611, 1965.

[69] F. J. Massey Jr, "The kolmogorov-smirnov test for goodness of fit," *Journal of the American statistical Association*, vol. 46, no. 253, pp. 68–78, 1951.

[70] R. A. Armstrong, "When to use the bonferroni correction," *Ophthalmic and physiological optics*, vol. 34, no. 5, pp. 502–508, 2014.

[71] O. Ledoit and M. Wolf, "A well-conditioned estimator for large-dimensional covariance matrices," *Journal of multivariate analysis*, vol. 88, no. 2, pp. 365–411, 2004.

[72] Y. Yarom and K. Falkner, "{FLUSH+ RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack," in *23rd USENIX security symposium (USENIX security 14)*, 2014, pp. 719–732.

[73] O. Choudary and M. G. Kuhn, "Template attacks on different devices," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2014, pp. 179–198.

[74] M. Green, L. Rodrigues-Lima, A. Zankl, G. Irazoqui, J. Heyszl, and T. Eisenbarth, "{AutoLock}: Why cache attacks on {ARM} are harder than you think," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1075–1091.

[75] D. Das, M. Sabbagh, R. Elnaggar, G. Chen, S. Ray, and J. Fung, "Optimal placement of tdc sensor for enhanced power side-channel assessment on fpgas," in *2024 37th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID)*. IEEE, 2024, pp. 443–448.

[76] M. Probst, L. Tebelmann, M. Wettermann, and M. Pehl, "Remote side-channel analysis of the loop puf using a tdc-based voltage sensor," *Journal of Cryptographic Engineering*, vol. 15, no. 1, p. 1, 2025.

[77] C. I. King, "Stress-ng," *URL: http://kernel. ubuntu. com/git/cking/stressng. git/(visited on 28/03/2018)*, vol. 39, 2017.

[78] J. Szefer, "Architectures and security of FPGA-accelerated cloud computing," SIGARCH Blog, May 2021, accessed: 2025. [Online]. Available: https://www.sigarch.org/architectures-and-security-of-fpga-accelerated-cloud-computing/

[79] C. Bobda, J. M. Mbongue, P. Chow, M. Ewais, N. Tarafdar, J. C. Vega, K. Eguro, D. Koch, S. Handagala, M. Leeser, M. Herbordt, H. Shahzad, P. Hofste, B. Ringlein, J. Szefer, A. Sanaullah, and R. Tessier, "The Future of FPGA Acceleration in Datacenters and the Cloud," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 15, no. 3, pp. 34:1–34:42, Feb. 2022.

[80] AMD, "Versal-hbm-series-product-brief," Product Brief, 2023.

[81] ——, "Versal-premium-series-gen-2-product-brief," Product Brief, 2024.

[82] G. Provelengios, D. Holcomb, and R. Tessier, "Mitigating voltage attacks in multi-tenant fpgas," *ACM transactions on reconfigurable technology and systems (TRETS)*, vol. 14, no. 2, pp. 1–24, 2021.

[83] F. Schellenberg, D. R. E. Gnad, A. Moradi, and M. B. Tahoori, "Remote inter-chip power analysis side-channel attacks at board-level," in *Proceedings of the International Conference on Computer-Aided Design*. San Diego California: ACM, Nov. 2018, pp. 1–7.

[84] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "C3APSULe: Cross-FPGA Covert-Channel Attacks through Power Supply Unit Leakage," in *Proceedings of the 2020 IEEE Symposium on Security and Privacy (S&P'20)*. IEEE, 2020, pp. 1728–1741.

[85] S. Moini, X. Li, P. Stanwicks, G. Provelengios, W. Burleson, R. Tessier, and D. Holcomb, "Understanding and Comparing the Capabilities of On-Chip Voltage Sensors against Remote Power Attacks on FPGAs," in *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug. 2020, pp. 941–944.

[86] C. Diktopoulos, K. Georgopoulos, A. Brokalakis, G. Christou, G. Chrysos, I. Morianos, and S. Ioannidis, "Assessing the Effectiveness of Active Fences Against SCAs for Multi-Tenant FPGAs," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*, Aug. 2022, pp. 391–396.

[87] C. Drewes, T. Sheaves, O. Weng, K. Ryan, B. Hunter, C. McCarty, R. Kastner, and D. Richmond, "Turn on, Tune in, and Listen up: Maximizing Side-Channel Recovery in Cross-Platform Time-to-Digital Converters," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 17, no. 3, pp. 49:1–49:30, Sep. 2024.

[88] I. Giechaskiel, K. Rasmussen, and J. Szefer, "Reading Between the Dies: Cross-SLR Covert Channels on Multi-Tenant Cloud FPGAs," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*, Nov. 2019, pp. 1–10.