Power and Frequency Intrinsic Channels on gem5

Lilian Bossuet, *Senior Member, IEEE*, and Carlos Andres Lara-Nino

***Abstract*— Recent works have highlighted the vulnerability of System-on-a-Chip (SoC) platforms against intrinsic channels attacks. In this threat model, an adversary can leverage vulnerabilities in the SoC's firmware, the operating system, or the design tools to gain access to shared resources in the platform and transfer data covertly. Given the diversity of attack avenues and the constant evolution of heterogeneous SoCs, it is not practical to study these attacks using conventional approaches. To address this issue, we propose to employ *gem5* in the study of power and frequency intrinsic channels. Our work studies heterogeneous SoCs which feature a processor system and an FPGA. We employ the full system simulation of *gem5* to emulate a reference physical device. We then describe the emulation of different intrinsic channels which leverage the clock tree and power distribution network of the SoC to transfer data covertly. Our findings demonstrate that *gem5* can accurately replicate the logical behavior of power and frequency intrinsic channels.**

***Index Terms*— Frequency intrinsic channels, gem5, heterogeneous SoC, power intrinsic channels, Zynq UltraScale+.**

I. Introduction

A SYSTEM-ON-A-CHIP is a heterogeneous platform, constituted by the integration of general processors and hardware accelerators in the same die. Their main components include a processor system with some memory elements, a shared memory block, acceleration engines (ASICs, FPGAs, or ASIPs like DSPs and GPUs), and some interconnect logic. In this work, we focus on the cases where the accelerator is constituted by reconfigurable logic. Heterogeneous SoCs have gained popularity given the need to improve the performance of processors through hardware acceleration. The desire for new computing architecture has been pushed, in part, by the loss of Dennard's scaling and the deceleration of Moore's Law [1], [2]. But also, by the interesting features offered by reconfigurable hardware accelerators [3], [4]. Along with these factors, the monetary cost per logic-cell of FPGA fabric has dramatically decreased, which makes them an attractive choice for bulk acceleration [5].

Understandably, modern trends in the design of heterogeneous SoCs have been driven by the interest of drawing greater performance for emerging applications benchmarks [6]. However, recent studies have shed light on the vulnerabilities of these systems [7]. A platform with a greater diversity of hardware components will experience greater security challenges. We must consider that each part of the system can be targeted by attackers. This is a concern for using heterogeneous SoCs in critical applications. The design of secure SoC architectures which maintain a competitive performance profile has become a priority in such domains.

Covert data transmission is one of the attacks proposed against heterogeneous SoCs [8]. Under this threat model, an adversary leverages the shared resources between the different components of the platform to secretly transfer information. These shared resources become intrinsic channels. The goal of these attacks is to allow different components within the platform to exchange information. These covert communications could bypass security policies designed to isolate the platform's components. Detecting and preventing these attacks is an active area of research.

Studying intrinsic channels with conventional strategies presents multiple challenges. First, the diversity of shared resources which can be leveraged by an adversary is quite large. Even when the scope is limited to clock, and power trees (as in this work) the number of potential attacks is unlimited. Truly, creativity dictates the boundary. Next, there are many SoC architectures being used for different applications. The main selling points of SoCs is that they can be tailored for a particular use case. Unfortunately, each of these designs ought to have particularities which make it susceptible to specific attacks. Furthermore, studying intrinsic channels in a physical device results costly. In most scenarios, disconnecting an intrinsic channel is simply not possible. So, if intrinsic channel vulnerabilities are discovered once the SoC has been deployed, these could be difficult to mitigate.

The use of pre-silicon tools, like *gem5* can contribute to solving some of the challenges associated with the analysis of intrinsic channels. By offering multiple architectural and processor models it allows the designer to test many system configurations. The fact that the device does not need to be manufactured reduces testing time. In fact, *gem5* does not require hardware descriptions nor precise physical models of components. Instead, it relies on the architectural description

Manuscript received 6 March 2024; revised 10 July 2024; accepted 28 July 2024. This work was supported in part by French Government through the Agence Nationale de la Recherche in the framework of Project "micro-ARCHItectural SECurity" (ARCHISEC) under Grant ANR-19-CE39-0008. The work of Carlos Andres Lara-Nino was supported in part by the Spanish Instituto Nacional de Ciberseguridad (INCIBE) through the Project HERMES, in part by the Spanish Government through the Project ACITHEC under Grant PID2021-124928NB-I00, and in part by the Catalonian Agència de Gestiód'Ajuts Universitaris i de Recerca (AGAUR) under Grant 2021 SGR 00115. This article was recommended by Associate Editor Q. Liu. *(Corresponding author: Carlos Andres Lara-Nino.)*

Lilian Bossuet is with CNRS, Institut d'Optique Graduate School, Laboratoire Hubert Curien UMR 5516, Université Jean Monnet Saint-Étienne, 42000 Saint-Étienne, France (e-mail: lilian.bossuet@univ-st-etienne.fr).

Carlos Andres Lara-Nino is with the Departament d'Enginyeria, Informàtica i Mathemàtiques, Universitat Rovira i Virgili, 43007 Tarragona, Spain (e-mail: carlos.lara@fundacio.urv.cat).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2024.3435841.

Digital Object Identifier 10.1109/TCSI.2024.3435841

of the platform to emulate its behavior. Given these features, the interest for using *gem5* in this research field is clear. In this paper, we demonstrate that *gem5* can emulate frequency and power intrinsic channels accurately.

Our main contributions include:

1) We present multiple approaches for transferring data between different elements within a SoC.
2) We demonstrate the feasibility of these attacks by implementing the intrinsic channels in a physical device.
3) We show how to emulate these attacks in *gem5* with a full-system ARM simulation and outline the challenges that we have overcome in the process.
4) We provide, to the best of our knowledge, a complete literature review on the emulation on intrinsic channels.

A conference version of this paper [9] has shown that it is possible to use *gem5* in the emulation of frequency intrinsic channels. Particularly, that *gem5* can emulate the behavior of a specific device whose characteristics are known. Earlier work [10] has shown that *gem5* can also simulate power intrinsic channels. However, that work does not employ a reference device to show the accuracy of the approach. In this paper, our goal is to bridge the gap between earlier works by presenting power and frequency intrinsic channels which are first implemented in a physical device and then emulated with *gem5*. As in [9], we provide an open repository with the source files which allow to reproduce our findings:

`https://github.com/CarlosAndresLARA/int-gem5`

The rest of the paper is structured as follows. In Section II we review the state of the art on intrinsic channels and the use of *gem5* for their emulation. Section III describes the threat model of covert communication attacks considered in this work and their limitations. In Section IV we detail the use of frequency and power intrinsic channels to mount covert communications attacks in a physical heterogeneous SoC. Subsequently, Section V describes the process to carry a successful emulation of these attacks in *gem5*. Then, in Section VI we discuss the results of our work and outline different perspectives. Lastly, Section VII presents our final remarks and concludes the paper.

II. Related Works

As mentioned, the covert transmission of data through intrinsic channels is one of multiple threats leveraged against complex heterogeneous SoCs. These internal links may allow an attacker to transfer data between processes that would not be allowed to communicate by the system's security policies. Typically, a covert transmission uses a spy application or circuit that infiltrates the system and transfers the sensitive data to a receiver which decodes it and uses it for illegitimate purposes.

The literature describes several methods for the utilization of intrinsic channels. Some of them rely on shared hardware resources such as memory elements. In [11], Lipp et al. used a common library (shared memory) and cache memory attacks to exchange sensitive data between two unprivileged processes. The covert means of communication between the two processes were based on the use of either the *Flush* + *Reload* [12] or the *Flush* + *Flush* [13] cache attacks. Others are not so evident to detect. In [14] the authors described how to exploit the mutual exclusion and synchronization mechanism of modern operating systems to transfer data covertly. These works highlight that the system architecture itself is ripe with opportunities for the instantiation of intrinsic channels.

In [15], Masti et al. evaluated the feasibility of thermal intrinsic channels. They used the thermal sensor included in a processor core to communicate two processes running on two different cores of the same processor. To send a logical one, their spy process would stress the core which would cause the processor to overheat. To send a logical zero, the spy process would decrease the workload of the core. To decode the data, their receiving process simply took core temperature readings belonging to the same processor. A related approach was used by Tian and Szefer [16] to mount temporal-thermal intrinsic channels on Cloud-based FPGAs. In their work, the authors showed that heat generated by one user of the FPGA could be observed by another user of the same FPGA in a subsequent session. Their attack was carried out on Microsoft Catapult's servers where FPGAs are available as remote acceleration platforms. Despite the inherent limitations of the approach, the work demonstrated that even with up to a few minutes of idle period it was possible to transmit data.

In [17], it was demonstrated that it is possible to exploit the crosstalk phenomenon of long wires [18] in the Arria 10 SoCs. In their work, the authors showed that the crosstalk can be observed for different long wires within these platforms. The basis for this type of attack is that the value transmitted in a long wire across the fabric has a noticeable effect on the delay of adjacent long wires. These wires exist to improve the routing and the performance of the reconfigurable designs. Crosstalk between long wires causes information leakage from one wire to another if they are close enough. When the source is used as part of a protected region of the SoC and the receiver is a wire connected to a malicious IP, this can lead to information leakage. If the transmitter is malicious and seeks to send information to a different IP, the long wires can be used to create intrinsic channels. A related approach was used by [19] to mount an attack on an AES core on a Cyclone IV FPGA. A potential countermeasure for these attacks was later presented by [20], who proposed routing strategies to mitigate the risks of crosstalk attacks by isolating sensitive nets from other components. Later, [8] studied the characteristics of frequency-based intrinsic channels in modern SoC-FPGA platforms. That work demonstrated that frequency-based intrinsic channels can be stealthy but also high performing.

The authors of [21] presented a practical use case for intrinsic channels. That work employed the clock tree of the device to transfer a single bit of information from a malicious application in the processor system to a malicious IP in the FPGA. This bit was used to synchronize the acquisition of power traces with an internal sensor. With that work, the authors showed that intrinsic channels do not require the transfer of large volumes of data or fast data rates to be of practical use for adversaries.

In [10] the authors have used *gem5* to study power analysis and intrinsic channel attacks on heterogeneous SoCs. Their work employs a high-demand payload in the kernel to induce a noticeable increase in the activity of the emulated system, which is supposed to affect the power dissipation of the device. The authors employ the simulation statistics produced by *gem5* to study the behavior of the simulation. However, that work does not corroborate the accuracy of the simulation by replicating the experiments in a physical device. Most recently, [9] showed that it is possible to employ *gem5* to emulate frequency-based covert communications. But that work does not address the potential of *gem5* for emulating the physical behavior of the circuit.

A. Exploiting the Frequency or Voltage Modulation

Works like [22] have used the power distribution network (PDN) of FPGAs to transfer data covertly to an external receiver. The technique described in that paper employs a power pattern generator inside the core as a transmitter. The receiver can be anything capable of monitoring the power trace of the board; in their case an oscilloscope was used. In their work, the authors did not envision the creation of intrinsic channels, but rather intended for this communication strategy to be used in monitoring, debugging, or watermarking [23], [24]. However, the evidence suggests that this approach could be coupled with in-board receivers to complement a covert remote-monitoring scheme. In this line of research, the work in [25] proposed to use the PDN to mount actual covert communications within the FPGA. The authors used non-combinatorial ring oscillator as transmitters and TDC-based sensors as receivers. This class of attacks underscores the significant challenges for isolation-based protection approaches since the PDN is a common resource throughout most SoCs. Indeed, the isolation challenges persist even when the logic is implemented in different dies. This was demonstrated by [26] using FPGAs with 2.5D integration of multiple dies, in concrete the Virtex Ultrascale+ series. In their work, the authors managed to create intrinsic channels across the different dies of the FPGA just by exploiting the perturbations induced on the PDN. Furthermore, in [27] the authors demonstrated that an FPGA could be used to analyze the power traces of a different FPGA within the same board. Later, in [28] the authors proposed to use the current management techniques found in modern processors to implement covert communications within different processes.

The potential of using frequency modulation to mount intrinsic channel attacks on multi core platforms was first studied by Alagappan et al. [29]. In that work, the authors demonstrated the feasibility of an intrinsic channel using frequency modulation. Their work employed dynamic frequency adjustment to transfer sensitive data between the spy process and the receiving process. To send a logical one, their spy process would overload the CPU as in [15], which would cause the system to change its frequency to meet the workload being demanded by the spy process. The frequency chosen would depend on the frequency governor mode used by the system (*performance*, *powersave*, *userspace*, *ondemand*, or *conservative*). To send a logical zero, their spy process would decrease the workload applied to the processor. The receiving process performed a simple frequency reading to decode the message. Independently, [30] presented the CLKSCREW attack which exploited vulnerabilities in the DVFS mechanisms to bypass the protections of the system. That work showed that a malicious driver could extract secret cryptographic keys from TrustZone, and escalate its privileges by loading self-signed code into application space. As countermeasures the authors proposed the introduction of hardware limit regulators and division of power domains across security boundaries, as well as potentially redesigning the chip with additional logic and timing redundancy to mitigate the effects of faults. Recently, the authors in [31] have shown that novel frequency management mechanisms in Intel CPUs can be leveraged to mount covert communication attacks.

In [32], the authors demonstrated for the first time a malicious use of the frequency modulation against a TrustZone-enabled SoC. The work described four proofs of concept to transfer sensitive data from a secure entity in the SoC to a non-secure one. Their study used the Zybo board which is equipped with a Zynq-7000 ARM/FPGA SoC (XC7Z010-1CLG400C). The main limitation of this work is that their approach is restricted to the context of a specific device. As new architectures and security systems have been proposed, these results are now outdated.

B. DVFS in gem5

gem5 is a modular platform for computer-system architecture research, encompassing the system-level architecture as well as processor micro-architectures [33]. This Open-Source simulator was created after merging the M5 and GEMS simulators, preserving the processing-emulation capabilities of the former and the memory-emulation components of the latter. It allows to run cycle-accurate simulations of multiple processor architectures, among them ARM. By creating a conglomerate of objects (*SimObjects*), *gem5* allows to emulate the interaction between the different components of the processing system and study their synergy, rather than simply trying to predict the outcome of some computation.

The use of *gem5* to emulate the DVFS-management was first introduced in [34] where the authors extended the simulator to support full-system DVFS modeling. Their goal was to enable energy-efficiency experiments to be performed in *gem5* and to highlight such studies. That work provided, for the first time, clock and voltage domain declaration, online power-estimation, a DVFS controller, and kernel drivers for full-DVFS support. Their proposal would become the basis for the DVFS handler included in the official *gem5* releases. Subsequently, works like [35] have proposed high-level improvements to enhance the performance of the DVFS handler in gem5. In that work, the authors introduced a non-intrusive application-controlled DVFS management implementation for the system-call emulation mode. The general goals of these works are to improve the data modeling quality or the handler's efficiency. This is an interesting use of gem5: it makes it possible to explore multiple design strategies without the need to create physical prototypes which result costly in terms of time and money.

In [36], the authors introduced an ongoing study aiming at analyzing the attacks relying on the hardware vulnerabilities of the micro-architectures of CPUs and SoCs using gem5. The main objectives of their work are to create a virtual and open platform that emulates the behavior of micro-architectural features and their interactions with the peripherals, like accelerators and memories in emerging technologies. The authors describe diverse attacks which can be mounted on the *gem5* simulator, among them the possibility of creating DVFS covert-channels as described in [32].

C. Other Emulation Tools

Emulating the comportment of analogue and digital systems is a challenge which has interested researchers for a long time. The usual goals of these analyses are to verify the correctness of the system and to discard unintended defaults. As such, many commercial and academic tools have been proposed to reproduce the electrical behavior of different platforms. A reduced set of products has been designed with the intended purpose of aiding in the security auditing of computing architectures. These are denominated leakage verification or detection tools. However, to the best of our knowledge, only [10] has dealt with the emulation of intrinsic channels.

Being an architectural attack, the first problem with emulating covert communications is that we must study the full system. There are few platforms which conduct full-system simulations like *gem5*. Another problem is that frequency intrinsic channels fall under the umbrella of logical behaviors, whereas power intrinsic channels fall under the class of analogue behaviors. There are few simulators which can produce results for both cases as well.

Leakage analysis tooling could be adopted in the study of intrinsic channels with some adjustments. However, the state-of-the-art in this field can be divided into two large groups: industrial and academic tooling which are not openly available. Open-source tools for leakage verification include MAPS [37] and COCO [38]. The former is exclusive for ARM Cortex M3 systems but is not cycle accurate. The latter can analyze any circuit but requires a gate-level description of the platform. Another tool which relies on an open-source initiative is SLEAK [39]. This system employs *gem5* to perform the emulation of ARM Cortex A8 processors. However, SLEAK itself is not readily available. In our work, we intend to use *gem5* "as is" to boost the reproducibility of our findings. Anybody who wishes to replicate our work can refer to the freely available distribution of the simulator.

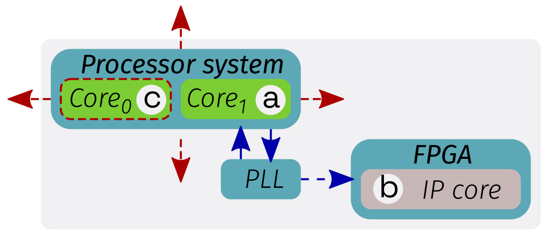


Fig. 1. Attack scenarios considered in this work. a) A malicious application transfers information to a second application through a frequency intrinsic channel. b) The receiver can also be a hardware accelerator with a malicious payload. c) A malicious application without root privileges can encode information by stressing the core, leveraging a power intrinsic channel.

III. Threat Model

In our work we consider two scenarios which can lead to the exploitation of intrinsic channels in a platform. The victim is always a heterogeneous SoC with a cluster of ARM Cortex-A53 processors running a generic Linux, a cluster of ARM Cortex-R5F processors running third-party applications on bare-metal, and a reconfigurable nucleus which implements third-party accelerators. In the first case (a) we consider a malicious application running on the user space as a transmitter which wishes to establish covert communications with a second application also on the user space. The second case (b) is that of a malicious application running on the user space as a transmitter which wishes to establish covert communications with a hardware accelerator in the FPGA. The hardware accelerator includes a malicious payload composed of ROS. In a third case (c), the transmitter is a malicious application running on the user space which wishes to establish covert communications with any other element in the SoC. These three cases are illustrated in Fig. 1.

For cases a) and b), we consider that the transmitter can modify the dividers of the different PLLs in the SoC to encode a message through different frequency symbols. For this, the transmitter requires root privileges which can be obtained through privilege escalation attacks or exploiting day-zero vulnerabilities. The receivers must be capable of monitoring the frequency of the SoC. This can be achieved by accessing the kernel APIs or by implementing internal sensors within the platform. The second assumption is stronger as it provides the adversary with the capabilities of modifying the bitstream of the system. However, for case c), we do not grant privileges to the malicious application and thus it cannot modify the device's clock tree. Instead, it can perform a sequence of complex operations to stress the device and produce power drops. The receiver can be inside or outside the SoC and is assumed to possess a certain sensing capability to monitor the power activity of the device. Internally, this could be achieved through accessing the kernel drivers, sensing digital delays, or monitoring the status of the on-board power regulator. Externally, an adversary might employ an oscilloscope, spectrum analyzer, or even a thermometer.

Covert communication attacks are less known than other physical attacks. Their reach and limitations have not been completely studied. On one hand, they are powerful tools which enable the attacker to bypass logical isolation policies.

On the other hand, the assumed adversarial capabilities may restrict the exploitability of intrinsic channels. For example, in case studies a) and b) from Fig. 1 it is assumed that the malicious application has root privileges, but this can be thwarted with the virtualization of untrusted modules [40], [41], [42]. Additionally, in a practical context, the sender might have to compete with other processes for the access to the clocking resources. For case study b) it is assumed that the adversary can integrate a malicious payload in the SoC, but this can be defeated by bitstream checking techniques [43], [44]. Power waster applications like the one employed in case study c) do not require privileges and could not trigger any compiler warnings. However, their operation could be disrupted by the activity of other processes in the system or through the implementation of active leakage-disruption countermeasures [45], [46]. Most recently, in [47] the authors have proposed a deep learning approach for the detection of covert communications. Overall, the study of the effectiveness of such countermeasures falls outside the scope of this work. We are concerned with the emulation of intrinsic channels and covert communication attacks. However, reproducing the behavior of the countermeasures is a natural future step on this line of research.

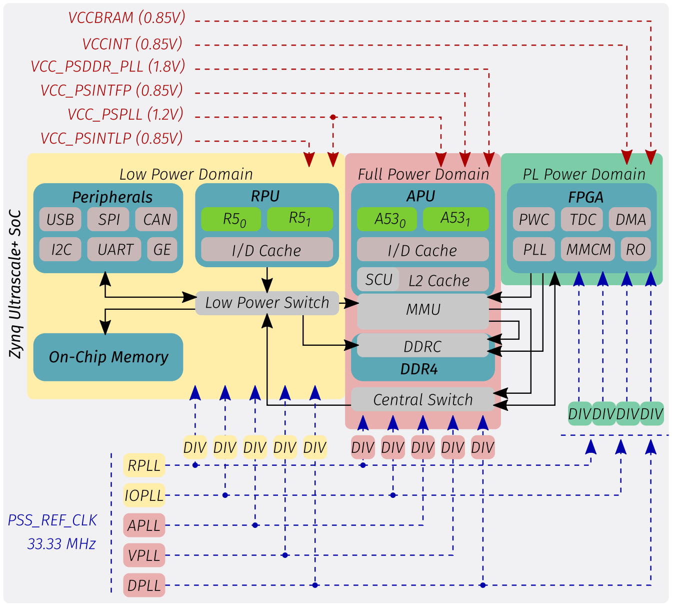


Fig. 2. The architecture of the AMD-Xilinx xczu2cg. The clock tree and power domains of these platforms are also illustrated. This image is adapted from [9].

IV. Intrinsic Channels on the Physical Device

A. The Zynq Ultrascale+ Heterogeneous SoCs

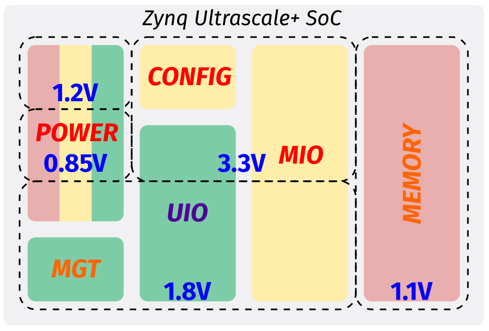
The AMD-Xilinx Zynq Ultrascale+ is an interesting case study for modern heterogeneous SoCs. We illustrate this architecture in Fig. 2. These chips feature an *application* processing unit (APU), powered by an array of ARM Cortex-A53 cores ($A53_0$ and $A53_1$ in Fig. 2), and a *real-time* processing unit (RPU), which includes an array of ARM Cortex-R5F cores ($R5_0$ and $R5_1$ in Fig. 2). Each one of these processing units has independent instruction and data caches, and up to L2 cache in the case of the APU. The main memory of the SoC is an external DDR unit, driven by an on-chip memory controller. There is also a smaller on-chip memory which can be shared by the different cores, and a memory management unit which performs the necessary assignments. These boards also feature a nucleus of programmable logic: an array of reconfigurable elements and silicon accelerators. The interconnection between processors and accelerators follows the AMBA-AXI specification through two main switches. The reconfigurable fabric of the SoC can implement a wide range of customized accelerators. For our work this means that we can use this module to create internal sensors based on reconfigurable logic.

In Fig. 2, in blue, we illustrate part of the clock tree in the Zynq Ultrascale+ SoC. A main reference clock (PSS_REF_CLK) is used to source the five main PLLs of the architecture (RPLL, IOPLL, APLL, VPLL, DPLL). To generate the PLL output, the reference clocks are multiplied by a constant. The resulting oscillators are then divided by one or two six-bit constants to produce specific clock domains for the distinct parts of the architecture.

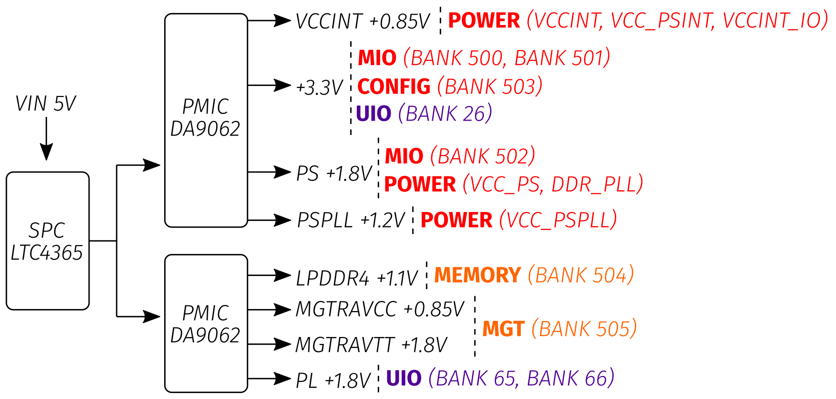
From Fig. 2 it can also be seen how there are three main power domains in these Ultrascale+ SoCs. The *Low Power Domain* will source the RPU, the peripherals, the on-chip memory, and one of the interconnect switches. The *Full Power Domain* will supply the APU, the memory management unit, the memory controller, and the central interconnect switch. The *PL Power Domain* will supply the reconfigurable fabric. The goal for this separation of power domains is to improve the system's energy efficiency by allowing it to shut down complete areas of the SoC when not used. For the low and full power domains, the five main PLLs can be used to generate clocks. For the FPGA, only three of the PLLs (RPLL, IOPLL, DPLL) can be used to generate the four clocks available to the fabric (from the processing system, since it is also possible to use external clocks.)

Ultrascale+ SoCs allow the use of the RPU and the APU independently. The cores in the RPU would normally run a real-time operating system [48] or standalone applications. The cores in the APU, on the other hand, are more complex and their full potential can best be drawn by a kernel, like Linux. In this work, we presume that both clusters can be operated independently. We implement bare metal applications in the RPU and Linux-based applications in the APU. These chips also feature a power management unit (PMU) which oversees the monitoring and configuration of the PDN. The PMU features anti-tampering characteristics which increase the difficulty of modulating the power supply of the chip. In this regard, as illustrated in Fig. 2, the SoC under analysis uses multiple voltage levels to source its different components.

In practice, however, the power domains employ different power supplies but some are shared (Fig. 3a). Moreover, depending on the prototyping board just one or two power management integrated circuits (PMIC) may be used to generate the different power levels required by the chip (Fig. 3b). This connection strategy found in several commercial boards



(a) voltage regions and power domains of the Zynq Ultrascale+



(b) two PMIC are used to generate all the voltage levels for the TE0802

Fig. 3. The power distribution network of the TE0802.

creates as a result a strong eclectic coupling between the different components of the SoC. Thus, facilitating the instantiation of intrinsic power channels.

B. Frequency Modulation in the Zynq Ultrascale+

The frequency of the different clocks can be modified by editing their multiplier or divider values. The multiplier register will affect the PLL output, and in turn modify the frequency of all the SoC components which rely on that given oscillator. In contrast, the divider registers are specific for a given clock and modifying them will only modify the frequency of a particular clock signal. There are clocks which use one and two dividers. These are stored as a six-bit section of a 32-bit register. To modify the frequency of an oscillator it is then necessary to edit the contents of these control registers.

At low level, like in bare-metal applications, the control registers of the SoC can be edited through direct access operations. For example, using the `xil_io` library. However, to edit one of these control registers it is necessary to edit multiple security and configuration registers so that the frequency change is enacted. Furthermore, the application performing the operation must have access rights.

In the presence of a kernel, the modulation of frequency can be simplified with the help of drivers which allow to request the modification of specific clocks. For example, the processor clocks (by using the `cpufreq` driver of Linux) or the FPGA clocks (by using the `fclk` drivers of Xilinx). This scenario is more favorable for attackers since the complexity of the kernel may allow them to hide malicious applications more easily.

C. Frequency Detection

A frequency detector can be a circuit built from logic elements which can measure the variation in the propagation delay of a digital signal [49]. These fluctuations are generated from changes in the power dissipation, electromagnetic coupling, and thermal fluctuations of the circuit. For this reason, such sensors have been employed to perform internal monitoring of the chip [50], [51], [52]. The main types of such sensors are based in time-to-digital converters and ring oscillators. The former are more accurate and provide greater resolution in the sampling but must be calibrated precisely and placed directly in the platform. In contrast, sensors based in ring oscillators (ROS) do not require any precise placement directives and provide sufficient information when enough samples are available.

Our work employs ring oscillator-based sensors due to their simplicity [51], [52]. In these architectures, the ring oscillator provides a consistent oscillatory wave whose period fluctuates according to the nominal operation of the circuit. This signal is then used to source a counter, which is subsequently sampled by an external clock to produce a measurement. The number of counts retrieved in a sampling period is thus correlated to the frequency of the ring oscillator, and in turn to the operation of the circuit. However, we are more interested in the sampling clock of the sensor. By modifying this signal, we can obtain an offset in the measurements due to the periodicity of the small counter.

The frequency fluctuation can be detected from the FPGA by observing the output of the sensor. Or from the processors, by reading the value of the divider registers. In this work we focus on the interaction between the processors and the programmable logic, so we prefer the latter method to monitor the frequency variation in the SoC. For this, we created a simulation model of the sensor which can produce a digital output as response to the frequency change.

D. Characterization of the System

To understand the limits of the proposed intrinsic channels we first studied the behavior of a PLL in the target platform. For all the proposed cases we used the IOPLL which can be used to source clocks in all the power domains of the SoC. Using a digital oscilloscope we sampled the time-response of these components when requesting a change in the output frequency. The oscilloscope was connected to a physical output of the prototyping board (PMOD), a TE0802 manufactured by Trenz electronic. The GPIOs are routed from the PS through the FPGA to the chip ports, then through the PCB to the

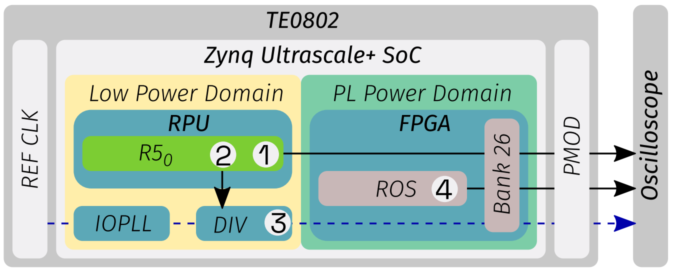


Fig. 4. The experimental setup used to characterize the frequency intrinsich channels in the Zynq Ultrascale+ SoC. 1) The ARM Cortex-R5F produces a digital trigger to signal the start of the operation. 2) It request a frequency change by updating the dvider of the IOPLL. 3) The IOPLL updates its output frequency as response to the processor's request. 4) The internal sensor detects the frequency change and produces an output offset.

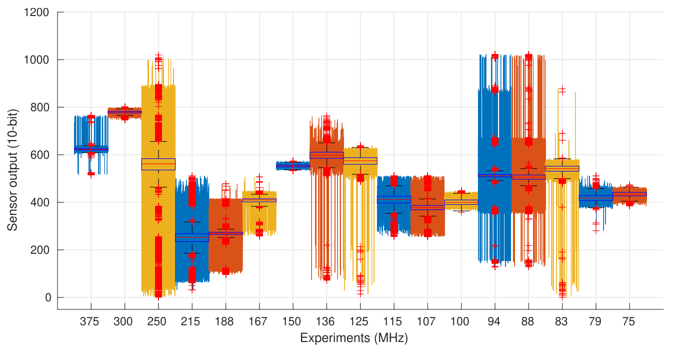


Fig. 5. Characterizing the output of the ROS as a function of the sampling frequency. The selected frequencies range from 375 MHz (dividing the output of the IOPLL at 1.5 GHz by four) to 75 MHz (dividing the same oscillator by 20). Each observation consists of 40,000 samples.

physical PMODs powered with a constant voltage of 3.3V. The FPGA itself allows for different voltage levels, but the physical outputs in the I/O Bank 26 used by the TE0802 only admit $Vcco = 1.8\ V$ (LVCMOS18). As reference, we generated a digital trigger through the processor's GPIOs. We then measured the width of these pulses. We also captured the activation of the MSB bit in the output of the delay sensor.

The experimental setup is illustrated in Fig. 4. We captured the behavior of the IOPLL output as a response to a change request from a bare-metal application which produces a digital trigger and the most-significant bit in the sensor output. Our intention was to measure the response time of the IOPLL and of the sensor to find the channel's minimal latency. Our findings suggest that the minimum response time for a frequency change is approximately $600ns$. That is the time elapsed from the moment one of the RPU cores modifies the register until the output of the sensor is updated. Therefore, assuming that we could transfer one bit per transition, the maximum bandwidth for the proposed channels would be 1.6 MBps. Note that this is the theoretical limit, without considering the necessary delay to achieve a consistent transmission (low-error rate).

With the configuration shown in Fig. 4 we are not able to test for additional GPIO or PMOD voltage levels on the TE0802. However, we suspect that the time response of the GPIOs would be different with other conditions (i.e., in other prototyping boards). In particular, the frequency response of the outputs might differ, which could impact the accuracy of the measurement of interest: the delay in the PLL response. Empirically, we verified that the PMODs of the TE0802 are accurate up to 100MHz. This gives us a resolution of 10ns. That is, 1/60th of the perceived response delay (600ns). Therefore, we believe that the impact of the prototyping board is not large enough to impede achieving an accurate characterization of the PLLs in the Zynq Ultrascale+ SoC.

We also characterized the ROS which would be used to detect the frequency change. For this, we implemented a matrix of 64 sensors and sampled it using different frequencies. Results for this experiment are provided in Fig. 5. It was possible to clearly differentiate between the multiple sample windows, with greater separation between 100, 150, and 300 MHz. This analysis was useful to document the sensor's behavior and create a model which could be used in the simulation.

E. Intrinsic Channels Within the APU

The first intrinsic channels we investigated are those that can be implemented within the APU of the platform. That is, we assume that a malicious application or driver being executed in one of the cores can transfer some information to a receiver in a different part of the APU. This kind of attack might be interesting for applications which delegate one or more of the cores to perform trusted computations.

A regular Linux kernel, if configured properly, will feature the `cpufreq` driver which allows to modify the frequency of the underlying system. This can be leveraged to implement an intrinsic channel between different cores controlled by the same operating system. We used this driver, available in the Linux distribution of AMD-Xilinx, to modify the processor clock of the APU. This oscillator is used by all the cores plus other components in this system. Therefore, through frequency modulation it is possible to transfer information between various parts of the APU. To demonstrate the feasibility of this attack, we created a *sender* program and a *receiver* program. They were cross-compiled and loaded in the file system using Petalinux. Subsequently each application was executed in different cores of the APU. We transmitted the 16-byte message *"This is a covert secret message!"* encoded with a straightforward modulation strategy of Alg. 1. This approach is viable since the `cpufreq` driver in Zynq Ultrascale+ platforms offer four frequency values.

In Fig. 6a we illustrate some of the samples captured by the receiver application. In this case, the transmitting and receiving delays should be similar since both applications are running in Linux and both perform the task of opening and writing/reading a file, which is slow. So, to increase the number of samples being retrieved and thus reduce the error rate we added a delay of 350 μS after the transmission of each symbol.

F. Intrinsic Channels Between the APU and the FPGA

The second type of intrinsic channels under evaluation were those that originate from an application executed in the APU

Algorithm 1 Frequency Modulation for Low-Width Windows

```
Require: $f_1$, $f_2$, $f_3$: three frequency symbols.
  for byte in message do
    for bit in byte do
      if bit then
        fclk ← $f_1$
      else
        fclk ← $f_2$
      end if
      fclk ← $f_3$
    end for
  end for
```

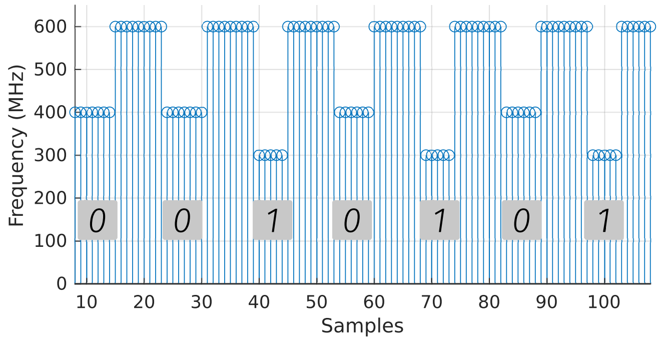
and target the reconfigurable fabric. Our intended receiver was the delay sensor based on ring oscillators. To transmit the data, we targeted one of the clocks sourcing the FPGA from the processing system. This signal was used as the sampling clock for the delay sensor.

In the case of the Linux distribution of AMD-Xilinx, the kernel also features a set of APIs (`/sys/devices`) which allow to modify the frequency of the FPGA clocks. These drivers use configuration files which can be managed from the application space. Thus, performing the modification of an FPGA oscillator is a matter of locating the adequate file, opening it, modifying its contents, and closing it again (the file must be closed for the change to be detected).

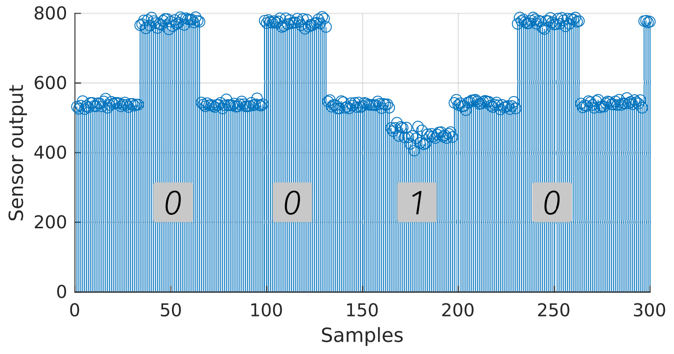
We applied a straightforward modulation strategy with a C-language application in Linux. The receiver was also a ring-oscillator based delay sensor implemented in the FPGA. We could read its output through an AXI channel. In this case, the sampling frequency of the FPGA was greater than the sending rate, so we removed the additional delay after the transmission of the symbols used in the previous attack. Instead, we used a 10 μS delay in the acquisition of samples. In Fig. 6b we illustrate the results for this experiment.

From this attack we could appreciate how the output of the delay-sensor fluctuated in function of the operation of the SoC, but also of the sampling rate. For a sampling frequency of 100 MHz, we observed a mean output value of 450 counts, for 150 MHz a mean output of 540 counts, and for 300 MHz a mean output value of 770 counts. Whereas the "noise" produced by the sensor showed a variation of $\pm$ 10 counts. We did not implement anything besides the sensor in the FPGA, thus we assumed there were no data-dependent components in the experiment. Nonetheless, there ought to be some influence from the activity of the processor system, but it was deemed negligible. In this experiment we could also observe how the transmission delay was far greater than the sampling rate, which was reduced with an additional sampling delay. The limiting factor being the requirement for the sender to perform frequency modulation through the `fclk` API.

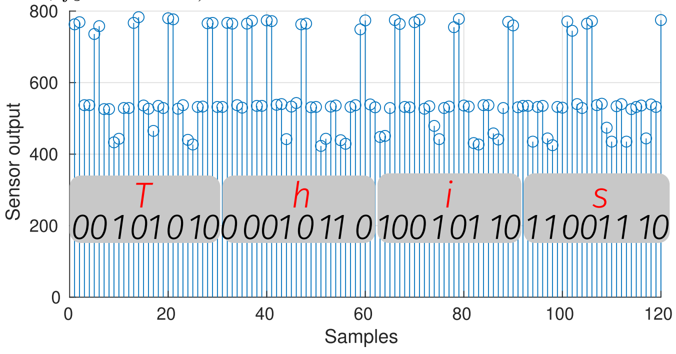
As discussed before, the FPGA clocks can also be modified by overwriting the value of their dividers in a register. This can be achieved in Linux by mapping the control registers of the SoC through the `mmap` utility. We used this approach to create a new sender application which would obtain access to the `CTRL_APB` registers, and to the `PLX_REF_CTRL` registers



(a) between different cores in the APU (f_1 = 300 MHz, f_2 = 400 MHz, f_3 = 600 MHz)



(b) from the Kernel to the FPGA: through drivers (f_1 = 100 MHz, f_2 = 300 MHz, f_3 = 150 MHz)



(c) from the Kernel to the FPGA: through registers (f_1 = 100 MHz, f_2 = 300 MHz, f_3 = 150 MHz)

Fig. 6. Frequency intrinsic channels in the Zynq Ultrascale+ SoC. The frequency-modulation is performed according to Alg. 1 with frequency symbols selected from Fig. 5. These images appeared in [9].

which contain the dividers for the FPGA clocks. We used the same modulation strategy as in previous experiments, added a small transmission delay, and removed the sampling delay from the previous experiment. The results are illustrated in Fig. 6c.

G. Power Intrinsic Channels

Some authors [25] have documented the feasibility of covert data transmissions over power intrinsic channels. These attacks leverage the characteristic of SoCs of sharing a common power distribution network. If the system under attack includes programmable voltage regulators, then these might be used to perform voltage scaling. Such regulators are normally connected to the processor through I2C buses. Therefore,

```
int iter = (int)strtoul(argv[1], NULL, 0);
volatile int i;
float a,b,c,d;
int e;
float nums[1000000];
for (i = 10; i < iter; ++i){
    a = i + i;
    b = i * i;
    c = b - a;
    d = c / (b + 1);
    e = (int)b % 1000000;
    nums[e] = d;
}
```

Fig. 7. A malicious application which stresses the core to transfer data.

an adversary with access to this bus could mount such attacks. In Linux systems, the DVFS controller modifies not only the frequency but also the voltage of the core, this has been documented in [10]. The kernel may switch between operating performance points predefined by the manufacturer. In this case, the adversary can obtain root privileges in the kernel and modify the system voltages to perform covert data transmission. Alternatively, a malicious application running on the kernel space can stress the processor to induce frequency fluctuations. This eliminates the assumption of granting root privileges to the adversary.

In a general sense, that is, even without the availability of a DVFS system, when a malicious application stresses the core, its behavior should have an impact on the power footprint of the system. The work in [25] demonstrated this phenomenon by employing power wasters in the FPGA. However, gaining access to the programmable logic and remaining undetected is challenging for an adversary. Instead, in our work, we have created a malicious transmitter application which simply performs a series of arithmetic operations. The application was coded in C and loaded into the kernel using Petalinux. The `stress` payload is presented in Fig. 7. This routine is called with a variable number of iterations (`iter`) to induce stress periods of variable length. The duration of such periods is then used to encode information. This modulation algorithm is detailed in Alg. 2. This modulation strategy allows to parameterize the activation and rest periods of the encoding which could potentially be used to optimize the transfer rate of the intrinsic channel. As there is no need to access any system files or registers then the limit is simply the recovery time of the capacitive elements in the board.

We studied the impact of the sender on the platform we monitored the main power supply of the board using a digital oscilloscope. The results are provided in Fig. 8. We applied a basic filtering of the power trace, and the activity of the transmitter became evident. This implies that the activity of the processor system has an impact on the main power supply, and therefore on the SoC as a whole. This attack highlights the challenge in mitigating intrinsic channel attacks. An adversary does not require any special privileges to induce a noticeable effect on the computing device.

Algorithm 2 Voltage Modulation for a Power Intrinsic Channel

```
Require: $n$ a given number of iterations
Require: $t$ a given wait time
  for byte in message do
    for bit in byte do
      if bit then
        stress($2n$)
      else
        stress($n$)
      end if
      wait($t$)
    end for
  end for
```

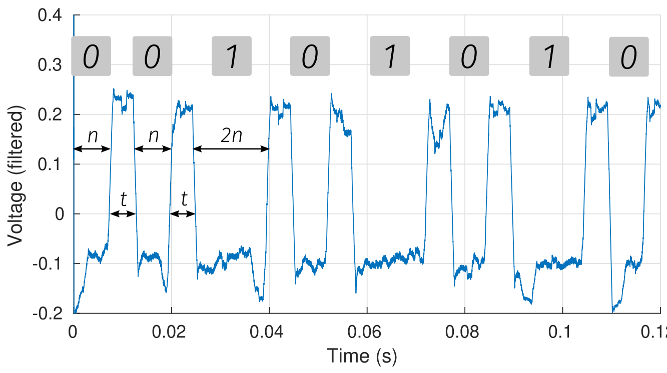


Fig. 8. The transmission of a stream of bits over a power intrinsic channel.

V. Intrinsic Channels on gem5

A. DVFS in gem5

The *gem5* simulator offers support for dynamically scaling the frequency of the system and its voltage levels. This is achieved by modeling an energy controller which performs frequency modulation. The registers of the `EnergyCtrl` *SimObject* can be read from a bare-metal application or through drivers in the Linux kernel. While this energy controller also performs voltage scaling, the simulator does not provide a regulator *SimObject* to monitor this value.

The hardware components (PLLs, voltage regulators) of the system are modeled using software scripts (*SimObjects*): `EnergyCtrl`, `DVFSHandler`, `ClockDomain`. The `ClockDomain` *SimObject* allows to define a clock domain with a frequency number and connect it to a component of the design. The `EnergyCtrl` and `DVFSHandler` *SimObjects* allow to apply a clock domain frequency (the clock source) according to the performance level chosen by the driver, if the platform is simulating a Linux environment, or according to the register contents if the system is bare-metal.

To use the DVFS system in the *gem5* simulator, the user must integrate the three *SimObjects* described, enable the use of the `DVFSHandler`, compile the Linux kernel with the `CPUfreq` driver and define the clock sources in the device tree. Then, the simulation script must also specify the operating performance points for the platform. Recall that these are pre-defined frequency/voltage pairs. These parameters can be either defined in the simulation script or provided as arguments. The latter approach allows for greater flexibility and is hence favored.

B. The `cpufreq` Driver and gem5

Emulating Linux-based operating systems like Ubuntu is a well understood process in gem5. The community has compiled a large set of binaries which can be used to run complete simulations.[1] The sources for the kernel and other binaries can also be obtained from online repositories.[2] However, the simulator offers the potential to use just any generic kernels and binaries for any purpose that the users might be interested in. For example, if we want to emulate an Zynq Ultrascale+ platform we would seek to use the AMD-Xilinx binaries, including their distribution of Linux.[3]

It is trivial to compile a generic Linux and load it into a *gem5* simulation. However, when it comes to DVFS, there are two critical components missing on regular kernels which are required by gem5. One is an extension of the `cpufreq` driver to include the *gem5 energy control* and the *gem5 multi-core* utilities. The other is an extension of the `clk` driver to include the *gem5 energy control clock*. To emulate the proposed attacks, it was first necessary to "patch" the kernel with the missing drivers. After editing the kernel's source, it is necessary to ensure that the `CONFIG_ARCH_GEM5_ENERGY_CTRL` and `CONFIG_ARM_GEM5_MULTI_CLUSTER_CPUFREQ` are set in the configuration file. We verified that applying this strategy to the official Linux kernel as well as the AMD-Xilinx Linux kernel allows to generate kernel binaries which can be used to emulate DVFS in gem5. The kernels were patched with source codes from the publicly available gem5 repositories.[2] The method for patching a Linux kernel is fully detailed in our public repository:

`https://github.com/CarlosAndresLARA/patches`

C. The Virtual Platform

The APU of the Zynq Ultrascale+ SoC is an array of Cortex-A53 cores clocked at a top frequency of 1.3 GHz. Each one of these cores has independent instruction and data caches and shares a common L2 cache and DDR memory. Our methodology aims at reproducing a virtual platform based on this board as close as possible with common *gem5* components. In theory, any computing device can be emulated with *gem5*, but the development time required may vary. In Fig. 9 we illustrate the components of the virtual platform used in our work.

The first step to construct the simulation was to compile the *gem5* simulator targeting the ARM architecture in *optimized* mode. We then created a full-system simulation script based on a multi-core architecture. We instantiate a variable number of cores with fixed instruction and data caches, as well as a shared L2 cache. To emulate the Cortex-A53 we opted for the `CpuCluster` *SimObject* with the *MinorCPU* model. The caches were emulated using the `L1_ICache`, `L1_DCache`, and `L2Cache` *SimObjects*. The voltage and frequency domains were provided through command line arguments, using the values available for the Zynq Ultrascale+ SoCs. To enable the emulation of the trusted firmware we used the `VExpress_GEM5_Foundation` machine type. We relied on the automatic generation of *gem5* to source the device-tree blob. As binaries, we used one of the bootloaders shipped with the *gem5* simulator. The kernel was our custom Linux binary. We used an Ubuntu image found online[1] and edited it by manually cross-compiling and packaging the attack applications.

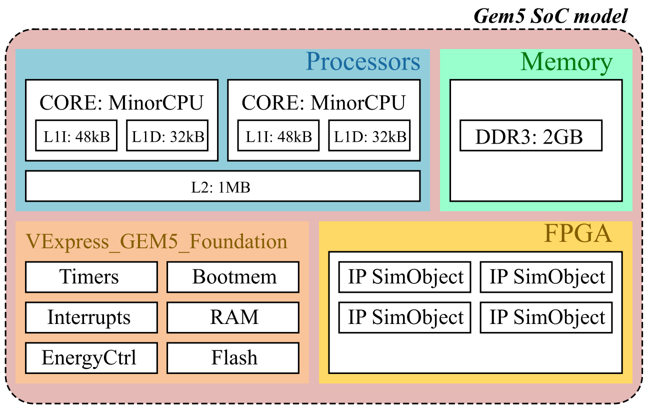


Fig. 9. The virtual platform emulated with *gem5*.

We created a custom *SimObject* and added it to *gem5* to emulate the behavior of the delay sensor in the FPGA. This module would read the control registers of the *EnergyCtrl SimObject* and produce an output through a debug flag. The output was modeled using our observations from Fig. 6b as a base offset according to the frequency plus a ± 10 random component. At this point we did not expand on the data-dependent component of the sensor output, but it would be interesting to implement more complex models as function of the state of different *SimObjects* in the simulation, for example the contents of the caches.

D. APU-to-APU Intrinsic Channels

The first intrinsic channel to be emulated was straightforward. We cross-compiled the sender and receiver applications and loaded them into the file system. Then we launched the simulation and started the applications. The results are shown in Fig. 10a. The main challenge to emulate the results from Subsection IV-E was determining the transmission delay of the sender. We observed that while the physical cores can maintain a constant delay due to the availability of a real-time clock, the delays in *gem5* depend on the operating frequency of the processor.

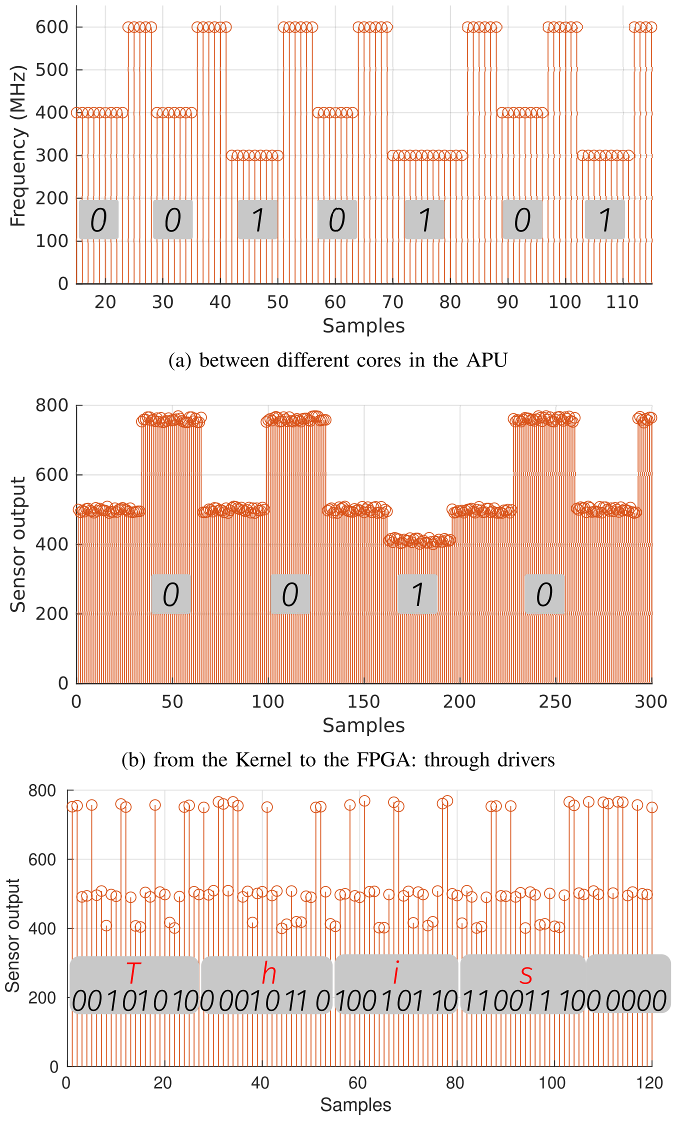
E. APU-to-FPGA Intrinsic Channels

Next, we emulated the intrinsic channels between a Cortex-A53 core and the delay sensor *SimObject*. For this scenario we modified the sender to also transfer some customization parameters to the receiver, for example the frequency symbols that would be used and the transmission delay. The results

[1]https://www.gem5.org/documentation/general_docs/fullsystem/guest_binaries

[2]https://gem5.googlesource.com

[3]https://github.com/Xilinx/linux-xlnx



(a) between different cores in the APU

(b) from the Kernel to the FPGA: through drivers

(c) from the Kernel to the FPGA: through registers

Fig. 10. Frequency intrinsic channels emulated with *gem5*. These images appeared in [9]. These results are associated with the experimental results shown in Fig. 6.

are shown in Fig. 10b. In this case, the main challenge was to identify the target registers since *gem5* does not include FPGA clocks. Instead, the `EnergyCtrl` *SimObject* allows to create multiple clock domains and assign a frequency to each domain through a couple of control registers. The delay-sensor *SimObject* was simply pointed to these registers. Thanks to the parametrization of our system, it was easier to achieve the desired results. We could adjust the transmission delay from the live simulation until the number of samples per bit was equivalent to the results observed in Subsection IV-F.

Finally, this dynamic modulation strategy was used to replicate the attacks where the kernel application has direct access to the registers. These results are illustrated in Fig. 10c. In this case we show more data and demonstrate the decoding of the message. As it can be noted, the difference in the transmission delay can accumulate over many samples causing the transmission rates between the real experiments and the emulation to diverge.

F. Emulating Power Intrinsic Channels

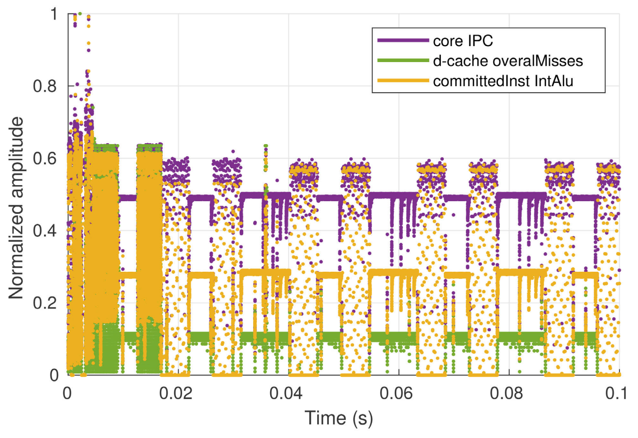
gem5 is a cycle-accurate architectural simulator. Therefore, there is a big question of whether it is possible to approximate physical behaviors with *gem5*. The most evident magnitude would be to approach the power footprint of the system from the simulation activity. This has been studied in [10] with moderate success. In that work, the authors intended to investigate the statistical relationship between the simulation activity and the application data. And the emulation of power intrinsic channels by observing the simulation statistics while the operating system processes a large payload (e.g. the AES trusted application found in OP-TEE). We complement that work by showing that *gem5* can indeed emulate power intrinsic channels even with smaller applications. And that it is possible to reproduce the behavior observed in a physical device.

We employed the virtual platform described in Subsection V-C and enabled the statistics reporting of *gem5* as described in [10]. We then implemented the malicious application used in Subsection IV-G in the virtual environment. We set a statistics dump rate of $1e-6$ and fixed reference values for n, t. We also disabled the DVFS system to remove its influence. We limited the scope of our analysis to the statistics associated with the processor system's activity. Some of these observations are illustrated in Fig. 11a. As can be noted, statistics like the instructions per cycle, the activity of the data cache, and the type of committed instructions can indeed provide insights on the workload of the operating system.

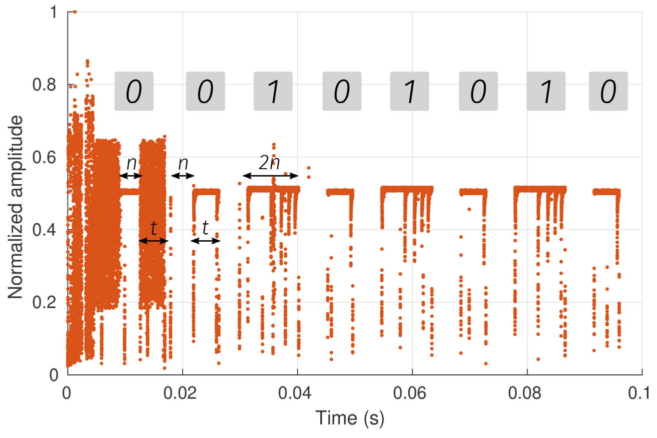
We assume that the statistics reported by *gem5* have a physical cost. In the physical device these statistics represent a given number of operations which have taken place in the device. These operations involve some energy expenditure. Therefore, we can approximate the power footprint of the SoC with a function of different statistics and some gain constants. For example, a basic power model like the one described in [10] can be used to approximate the dynamic power of the virtual platform. This model is presented in Fig. 11b. Normally, from a physical device we have access to the product of all the “implicit” statistics of the SoC. But with *gem5* the process is inverted in the sense that we have the fine-grained statistics, and we seek to approximate the power model. The results from Fig. 11b are meant to be analogue to those in Fig. 8. In this case we can also observe a small timing discrepancy which can be reduced by adjusting the t parameter in the virtual platform.

VI. Discussion and Perspectives

gem5 is an interesting platform for the analysis of intrinsic channels. It enables the emulation of the behavior of software routines and custom hardware modules. The latter are described as python classes which can define their behavior using multiple programming languages. For example, in this work we have emulated a Hardware Trojan to function as a receiver in our covert communication attacks. In this sense,



(a) simulation statistics



(b) transmission of a stream of bits

Fig. 11. A power intrinsic channel emulated with *gem5*. This result is associated with the experimental result shown in Fig. 8.

gem behaves as a logical emulator which allows us to verify the correctness of the system. However, at a lower level it also lets us approach the system's physical behavior. A *gem5* simulation produces statistics which register the status of the architecture at regular intervals. As shown in [10], these statistics might reveal insights on the physical behavior of an equivalent device. Nonetheless, there are multiple limitations which ought to be addressed to improve the usability of *gem5* for the emulation of intrinsic channels and other physical attacks.

One of the main challenges for the use of *gem5* is the slow learning curve for beginners. The *gem5* project is quite active, but the documentation lags at times. The "know how" about certain functionalities might get lost as a result and new users might be left without help. Additionally, some changes adopted by the main *gem5* distribution are quite large but not quite thorough. This can render years of parallel development outdated. Thus, modules that rely on a particular version of the kernel can have limited usability. To prevent this problem, we have avoided modifying the platform as much as possible. However, even basic simulation scripts can become obsolete given the dynamicity of the *gem5* community. The statistics-reporting of the simulator is affected by these problems. Being one of the earliest components of the system it has remained operational through many iterations of changes. However, as statistics are not usually harvested in large volumes the dumping mechanism itself remains quite rudimentary. Some authors have opted for harvesting data using the applications themselves or the debug utilities of the kernel [36]. But these approaches are useful when we know what exactly it is that we are looking for. Indeed, leakage verification and analysis benefit from "all" of the information to conduct a holistic study.

Another interesting perspective in the use of *gem5* is the emulation of countermeasures. Once the attacks are known, for example as reported in this paper, then it is possible to propose countermeasures. *gem5* would remove the technology dependency which so often limits the reproducibility of results. The results obtained in this work provide confidence that the results shown in the simulator would also apply to the physical device. For example, logical protections seeking to limit the interaction between the kernel space and the hardware could be implemented simply by changing the simulation binaries. Or leakage randomization strategies could be studied by constructing power models from the simulation statistics.

Overall, *gem5* offers great potential with its modularity. It is simple to create virtual models of hardware components and to include them into a full system simulation. This enables the analysis of individual modules in a larger context such a SoC with a full operating system. Nonetheless, approaching the behavior of hardware elements is always a challenge. For example, a simple solution would be to create *SimObjects* which produce statistics based on their input parameters. Power leakage could be approached through the Hamming weight of some function of these data. But there are better methods for generating synthetic data if we already know the model. Indeed, the relevance of employing automated tools in the analysis of physical attacks is discovering unintended or hidden behaviors. A potential improvement could be to read hardware descriptions from netlists or to support simulation activity files to replicate the functionality of circuits.

VII. Conclusion

In this paper, we have presented our results regarding the use of *gem5* to emulate the covert transmission of data on heterogeneous SoCs. Our results illustrate that despite the differences between the real and the emulated platform, the emulation is flexible enough to allow for parametrization of different components and values. This can bring the results closer to the expected observations.

The proposed methodology allows to study the implementation of power and frequency-based intrinsic channels in ARM systems. However, given the flexibility of *gem5* it is simple to modify the parameters of the system to emulate a different platform. In this case we only need to adjust the response time of the virtual models to account for technology variations. The *gem5* simulator allows to emulate any kernel which is to be run in the physical device (given enough processing resources) hence the interaction of the same drivers can be replicated. Furthermore, the simulator offers support for different processor architectures such as RISC-V so it would also be possible to emulate non-ARM SoCs and study the proposed attacks in these platforms.

In this paper, we filled a gap in the state of the art by emulating power intrinsic channels and demonstrating that the simulation's behavior approaches that of a physical device. We have also demonstrated that an adversary may leverage the simple behavior of an unprivileged application to mount power intrinsic channels on the target device. This removes many of the assumptions found in the literature.

References

[1] L. Johnsson and G. Netzer, "The impact of Moore's law and loss of dennard scaling: Are DSP SoCs an energy efficient alternative to x86 SoCs?" *J. Phys., Conf. Ser.*, vol. 762, Oct. 2016, Art. no. 012022.

[2] L. Eeckhout, "Is Moore's law slowing down? What's next?" *IEEE Micro*, vol. 37, no. 4, pp. 4–5, Jul. 2017.

[3] C. Kachris, B. Falsafi, and D. Soudris, *Hardware Accelerators in Data Centers*, 1st ed., Cham, Switzerland: Springer, Aug. 2018.

[4] C. Jin, V. Gohil, R. Karri, and J. Rajendran, "Security of cloud FPGAs: A survey," 2020, *arXiv:2005.04867*.

[5] A. Khawaja, J. Landgraf, R. Prakash, M. Wei, E. Schkufza, and C. J. Rossbach, "Sharing, protection, and compatibility for reconfigurable fabric with AmorphOS," in *Proc. 13th USENIX Symp. Operating Syst. Design Implement.* Berkeley, CA, USA: USENIX Association, Oct. 2018, pp. 107–127.

[6] A. Clark, "Xilinx machine learning strategies for edge," Xilinx, San Diego, CA, USA, 2018. [Online]. Available: https://web.archive.org/web/20220407054353/https://www.xilinx.com/publications/events/machine-learning-live/san-diego/xilinx_machine_learning_strategies_for_edge.pdf

[7] S. Chaudhuri, "A security vulnerability analysis of SoCFPGA architectures," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[8] L. Bossuet and C. A. Lara-Nino, "Advanced covert-channels in modern SoCs," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2023, pp. 80–88.

[9] L. Bossuet and C. A. Lara-Nino, "Emulating covert data transmission on heterogeneous SoCs," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2023, pp. 1–7.

[10] L. Bossuet, V. Grosso, and C. A. Lara-Nino, "Emulating side channel attacks on gem5: Lessons learned," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops (EuroSPW)*, Jul. 2023, pp. 287–295.

[11] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "ARMageddon: Cache attacks on mobile devices," in *Proc. 25th USENIX Secur. Symp. (USENIX Security)* Berkeley, CA, USA: USENIX Association, Aug. 2016, pp. 549–564.

[12] P. Mata and N. Rao, "Flush-reload attack and its mitigation on an FPGA based compressed cache design," in *Proc. 22nd Int. Symp. Quality Electron. Design (ISQED)*, Apr. 2021, pp. 535–541.

[13] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Cham, Switzerland: Springer, 1007, pp. 279–299.

[14] J. Zhang, C. Shen, and G. Qu, "Mex+sync: Software covert channels exploiting mutual exclusion and synchronization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 12, pp. 4491–4504, Dec. 2023.

[15] M. Alagappan, J. Rajendran, M. Doroslovacki, and G. Venkataramani, "DFS covert channels on multi-core platforms," in *Proc. IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SoC)*. Berkeley, CA, USA: USENIX Association, Aug. 2015, pp. 865–880.

[16] S. Tian and J. Szefer, "Temporal thermal covert channels in cloud FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2019, p. 298.

[17] G. Provelengios, C. Ramesh, S. B. Patil, K. Eguro, R. Tessier, and D. Holcomb, "Characterization of long wire data leakage in deep submicron FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2019, p. 292.

[18] I. Giechaskiel, K. Eguro, and K. B. Rasmussen, "Leakier wires: Exploiting FPGA long wires for covert- and side-channel attacks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 3, pp. 1–29, Aug. 2019.

[19] C. Ramesh et al., "FPGA side channel attacks without physical access," in *Proc. IEEE 26th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2018, pp. 45–52.

[20] Z. Seifoori, S. S. Mirzargar, and M. Stojilović, "Closing leaks: Routing against crosstalk side-channel attacks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2020, p. 197.

[21] A. Fellah-Touta, L. Bossuet, and C. A. Lara-Nino, "Combined internal attacks on SoC-FPGAs: Breaking AES with remote power analysis and frequency-based covert channels," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops (EuroSPW)*, Jul. 2023, pp. 281–286.

[22] D. Ziener, F. Baueregger, and J. Teich, "Using the power side channel of FPGAs for communication," in *Proc. 18th IEEE Annu. Int. Symp. Field-Program. Custom Comput. Mach.*, May 2010, pp. 237–244.

[23] C. Marchand, L. Bossuet, and E. Jung, "IP watermark verification based on power consumption analysis," in *Proc. 27th IEEE Int. Syst.-on-Chip Conf. (SOCC)*, Sep. 2014, pp. 330–335.

[24] L. Bossuet, P. Bayon, and V. Fischer, "Electromagnetic transmission of intellectual property data to protect FPGA designs," in *VLSI-SoC: Design for Reliability, Security, and Low Power*. Cham, Switzerland: Springer, 1007, pp. 150–169.

[25] D. R. E. Gnad, C. D. K. Nguyen, S. H. Gillani, and M. B. Tahoori, "Voltage-based covert channels using FPGAs," *ACM Trans. Design Autom. Electron. Syst.*, vol. 26, no. 6, pp. 1–25, Jun. 2021.

[26] I. Giechaskiel, K. Rasmussen, and J. Szefer, "Reading between the dies: Cross-SLR covert channels on multi-tenant cloud FPGAs," in *Proc. IEEE 37th Int. Conf. Comput. Design (ICCD)*, Nov. 2019, pp. 1–10.

[27] F. Schellenberg, D. R. E. Gnad, A. Moradi, and M. B. Tahoori, "Remote inter-chip power analysis side-channel attacks at board-level," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2018, pp. 1–7.

[28] J. Haj-Yahya et al., "IChannels: Exploiting current management mechanisms to create covert channels in modern processors," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2021, pp. 985–998.

[29] M. Alagappan, J. Rajendran, M. Doroslovacki, and G. Venkataramani, "DFS covert channels on multi-core platforms," in *Proc. IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, Dec. 2017, pp. 1–6.

[30] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: Exposing the perils of security-oblivious energy management," in *Proc. 26th USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, Aug. 2017, pp. 1057–1074.

[31] Y. Guo, D. Cao, X. Xin, Y. Zhang, and J. Yang, "Uncore encore: Covert channels exploiting uncore frequency scaling," in *Proc. 56th Annu. IEEE/ACM Int. Symp. Microarchitecture*, New York, NY, USA, Oct. 2023, pp. 843–855.

[32] E. M. Benhani and L. Bossuet, "DVFS as a security failure of TrustZone-enabled heterogeneous SoC," in *Proc. 25th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2018, pp. 489–492.

[33] N. Binkert, "The Gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, pp. 1–7, Aug. 2011.

[34] V. Spiliopoulos, A. Bagdia, A. Hansson, P. Aldworth, and S. Kaxiras, "Introducing DVFS-management in a full-system simulator," in *Proc. IEEE 21st Int. Symp. Modeling, Anal. Simulation Comput. Telecommun. Syst.*, Aug. 2013, pp. 535–545.

[35] Y. H. Yassin, M. Jahre, P. G. Kjeldsberg, S. Aunet, and F. Catthoor, "Fast and accurate edge computing energy modeling and DVFS implementation in GEM5 using system call emulation mode," *J. Signal Process. Syst.*, vol. 93, no. 1, pp. 33–48, Jan. 2021.

[36] Q. Forcioli et al., "Virtual platform to analyze the security of a system on chip at microarchitectural level," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops (EuroSPW)*, Sep. 2021, pp. 96–102.

[37] Y. Le Corre, J. Großschädl, and D. Dinu, "Micro-architectural power simulator for leakage assessment of cryptographic software on ARM Cortex-M3 processors," in *Proc. 9th Int. Workshop Constructive Side-Channel Anal. Secure Design*. Cham, Switzerland: Springer, 2018, pp. 82–98. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-89641-0_5

[38] B. Gigerl, V. Hadzic, R. Primas, S. Mangard, and R. Bloem, "COCO: Co-design and co-verification of masked software implementations on CPUs," in *Proc. 30th USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, Aug. 2021, pp. 1468–1469.

[39] D. Walters, A. Hagen, and E. Kedaigle, "SLEAK: A side-channel leakage evaluator and analysis kit," MITRE Corp., Bedford, MA, USA, Tech. Rep. AD1107774, 2014.

[40] B. Sá, J. Martins, and S. Pinto, "A first look at RISC-V virtualization from an embedded systems perspective," *IEEE Trans. Comput.*, vol. 71, no. 9, pp. 2177–2190, Sep. 2022.

[41] J.-Y. Hwang et al., “Xen on ARM: System virtualization using Xen hypervisor for ARM-based secure mobile phones,” in *Proc. 5th IEEE Consum. Commun. Netw. Conf.*, Jan. 2008, pp. 257–261.

[42] S. Yazdanshenas and V. Betz, “Quantifying and mitigating the costs of FPGA virtualization,” in *Proc. 27th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2017, pp. 1–7.

[43] D. R. E. Gnad, S. Rapp, J. Krautter, and M. B. Tahoori, “Checking for electrical level security threats in bitstreams for multi-tenant FPGAs,” in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2018, pp. 286–289.

[44] T. M. La, K. Matas, N. Grunchevski, K. D. Pham, and D. Koch, “FPGAD efender: Malicious self-oscillator scanning for Xilinx UltraScale + FPGAs,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 3, pp. 1–31, Sep. 2020.

[45] J. Krautter, D. R. E. Gnad, F. Schellenberg, A. Moradi, and M. B. Tahoori, “Active fences against voltage-based side channels in multi-tenant FPGAs,” in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–8.

[46] O. Glamoćanin, A. Kostic, S. Kostic, and M. Stojilovic, “Active wire fences for multitenant FPGAs,” in *Proc. 26th Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, May 2023, pp. 13–20.

[47] A. R. Díaz-Rizo, A. E. Abdelazim, H. Aboushady, and H.-G. Stratigopoulos, “Covert communication channels based on hardware trojans: Open-source dataset and AI-based detection,” in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2024, pp. 101–106.

[48] R. P. Dick, G. Lakshminarayana, A. Raghunathan, and N. K. Jha, “Analysis of power dissipation in embedded systems using real-time operating systems,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 5, pp. 615–627, May 2003.

[49] G. W. Roberts and M. Ali-Bakhshian, “A brief introduction to time-to-digital and digital-to-time converters,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 57, no. 3, pp. 153–157, Mar. 2010.

[50] I. Vornicu, R. Carmona-Galán, and Á. Rodríguez-Vázquez, “Arrayable voltage-controlled ring-oscillator for direct time-of-flight image sensors,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 11, pp. 2821–2834, Nov. 2017.

[51] M. Zhao and G. E. Suh, “FPGA-based remote power side-channel attacks,” in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 229–244.

[52] J. Gravellier, J.-M. Dutertre, Y. Teglia, P. L. Moundi, and F. Olivier, “Remote side-channel attacks on heterogeneous SoC,” in *Proc. 18th Smart Card Res. Adv. Appl. Conf. (CARDIS)*. Cham, Switzerland: Springer, Mar. 2020, pp. 109–125. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-42068-0_7

Lilian Bossuet (Senior Member, IEEE) received the M.Sc. degree in electrical engineering from INSA, Rennes, France, in 2001, and the Ph.D. degree in electrical engineering and computer sciences from the University of South Britanny, Lorient, France, in 2004. Since 2017, he has been a Professor with Université Jean Monnet Saint-Étienne, where he is currently the Head of the Computer Science Department, Hubert Curien Laboratory. He is also the Head of the Secured Embedded Systems and Hardware Architecture Group, Hubert Curien Laboratory. His main research interests include hardware security, security of embedded systems, IP protection, PUF design and characterization, secure-by-design crypto-processor, and reconfigurable architecture. He has published more than 200 refereed publications in these areas.

Carlos Andres Lara-Nino received the master’s and Ph.D. degrees in computer sciences from CINVESTAV, Mexico, in 2016 and 2020, respectively. From 2021 to 2024, he was a Post-Doctoral Fellow with the Hubert Curien Laboratory, Jean Monnet University, and CNRS, France. He is currently a Researcher with the Security and Privacy Research Group (CRISES), Universitat Rovira i Virgili, Spain. His research interests include digital systems design, data processing with reconfigurable hardware, information and hardware security, and cryptography.