Emulating Covert Data Transmission on Heterogeneous SoCs

Lilian Bossuet, Carlos Andres Lara-Nino

Emulating Covert Data Transmission on Heterogeneous SoCs

Lilian BOSSUET and Carlos Andres LARA-NINO
Université Jean Monnet Saint-Étienne, CNRS, Institut d'Optique Graduate School,
Laboratoire Hubert Curien UMR 5516, F-42023,
SAINT-ETIENNE, France.
carlos.lara@univ-st-etienne.fr

Abstract

Recent works have highlighted the vulnerability of System-on-a-Chip (SoC) platforms against frequency-based covert channels. An attacker might be able to leverage vulnerabilities in the SoC's firmware, the operating system, or the design tools to gain access to the underlying hardware and perform frequency modulation. Given the diversity of threats and the constant evolution of SoC platforms, it is not practical to study this attack model using physical devices. To address this issue, we propose to employ advanced simulation techniques. Our work targets heterogeneous SoCs which feature a processor system based on the ARM architecture plus an FPGA. We employ the full system simulation of *gem5*, which allows us to create a complete virtual-system and study the interaction between its components. We present the emulation of three frequency-based covert channels in *gem5* v22 by showing that it is possible to replicate the covert transmission of data between different elements of the SoC. To ensure the repeatability of our experiments all the sources are released as Open-Source.

Keywords: Covert channels, Frequency modulation, gem5, SoC-FPGAs, Zynq UltraScale+

Please cite as:

```
@InProceedings{BL23,
title = {{Emulating Covert Data Transmission on Heterogeneous SoCs}},
author = {Bossuet, Lilian and Lara-Nino, Carlos Andres},
booktitle = {Proceedings of the 2023 Asian Hardware Oriented Security and Trust Symposium
(AsianHOST)},
pages = {1--6},
year = {2023},
publisher = {IEEE},
location = {Tianjin, China},
doi = {10.1109/AsianHOST59942.2023.10409377},
isbn = {979-8-3503-4099-0}}
```

1 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. These architectures 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 [JN16]. But also, by the interesting features found in hardware accelerators [KFS18]. 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 [Kha+18].

Understandably, the trends in the design of these platforms have been driven by the interest of obtaining greater performance figures and improving the benchmarks for emerging applications [Cla18]. However, recent studies have shed light on the vulnerabilities of these systems [Cha18]. A platform with a greater diversity of hardware components will experience grater security challenges as each part of the system can be targeted by attackers.

Covert data transmission is one of the attacks proposed against heterogeneous SoCs [BL23]. Under this threat model an adversary may leverage the shared resources between the different components of the platform in order to establish incidental channels. These can then be used to allow different applications or circuits within the SoC to exchange information. Evidently, these covert communications would be able to bypass security policies designed to isolate the components of the platform. Detecting and preventing these attacks is an active area of research.

In this paper, we detail the process to emulate frequency-based covert channels in *gem5* v22. Our contributions are as follows:

1. We present three approaches for transferring data between applications on the processing system, and from the processing system to the programmable logic.

2. We demonstrate the feasibility of these attacks by implementing the covert 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 identify a key characteristic of the *gem5* simulator, which despite being able to emulate any version of the Linux kernel, can only run a complete simulation when these kernels include certain *gem5* code. To be clear, we document that the `clk` and `cpufreq` drivers of the Linux kernel used in *gem5* must be patched with some simulation-specific code.

The rest of the paper is outlined as follows. In Section 2 we study the State of the Art on covert channels and the use of *gem5* for their emulation. Section 3 details the implementation of frequency covert-channels in a physical SoC-FPGA. Subsequently, Section 4 describes the process to carry a successful DVFS-enabled simulation in *gem5* v22, and the emulation of the covert-channel

attacks. In Section 5 we review the results and discuss our findings. Lastly, Section 6 presents our conclusions.

2 Related works

2.1 Covert channels

The literature describes several methods for the creation of covert channels. Most of them rely on shared hardware resources such as memory elements. In [Lip+16], Lipp et al. used a common library (shared memory) and cache memory attacks to exchange sensitive data between two unprivileged processes. In [Mas+15], Masti et al. evaluated the feasibility of thermal covert channels. They used the thermal sensor included in a processor core to communicate two processes running on two different cores of the same processor. A related approach was used by Tian and Szefer [TS19] to mount temporal-thermal covert 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.

In [Pro+19], it was demonstrated that it is possible to exploit the cross-talk phenomenon of long wires [GER19] in the Arria 10 SoCs. A related approach was used by [Ram+18] to mount an attack on an AES core on a Cyclone IV FPGA. A potential countermeasure for these attacks was later presented by [SMS20], who proposed routing strategies to mitigate the risks of cross-talk attacks by isolating sensitive nets from other components. Most recently, [BL23] studied the characteristics of frequency-based covert channels in modern SoC-FPGA platforms.

2.2 Exploiting the frequency or voltage modulation

Works like [ZBT10] have used the Power Distribution Network (PDN) of FPGAs to transfer covert data to a receiver outside the board. 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. The work in [Gna+19] proposed to use the PDN to mount actual covert channels in the FPGA. The authors used non-combinatorial ring oscillator as transmitters and TDC-based sensors as receivers.This class of attacksunderscores the significant challenges for isolation-based protection approaches since the PDN is a common resource throughout most SoCs.

The isolation challenges persist even when the logic is implemented in different dies, so long as they share a common PDN. This was demonstrated by [GRS19] using FPGAs with 2.5D integration of multiple dies, in concrete the Virtex Ultrascale+ series. In their work, the authors managed to create covert channels across the different dies of the FPGA just by exploiting the perturbations induced on the PDN. Furthermore, in [Sch+18] the authors demonstrated that an FPGA could be used to analyze the power traces of a different FPGA within the same board.

The potential of using frequency modulation to mount covert channel attacks on multi core platforms was first studied by Alagappan et al. [Ala+17],

who demonstrated the feasibility of a covert channel using frequency modulation. Their work employed dynamic frequency adjustment to transfer sensitive data between the spy process and the receiving process. Independently, [TSS17] 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. In [BB18], 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.

2.3 DVFS in the *gem5* simulator

gem5 is a modular platform for computer-system architecture research, encompassing the system-level architecture as well as processor micro-architectures [Bin+11]. 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 [Spi+13] 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 showcase 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 [Yas+20] have proposed high-level improvements to enhance the performance of the DVFS handler in gem5.

In [For+21], 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 [BB18].

3 Experiments on the physical device

3.1 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. 1. These chips feature an *application* processing unit (APU), powered by an array of ARM Cortex-

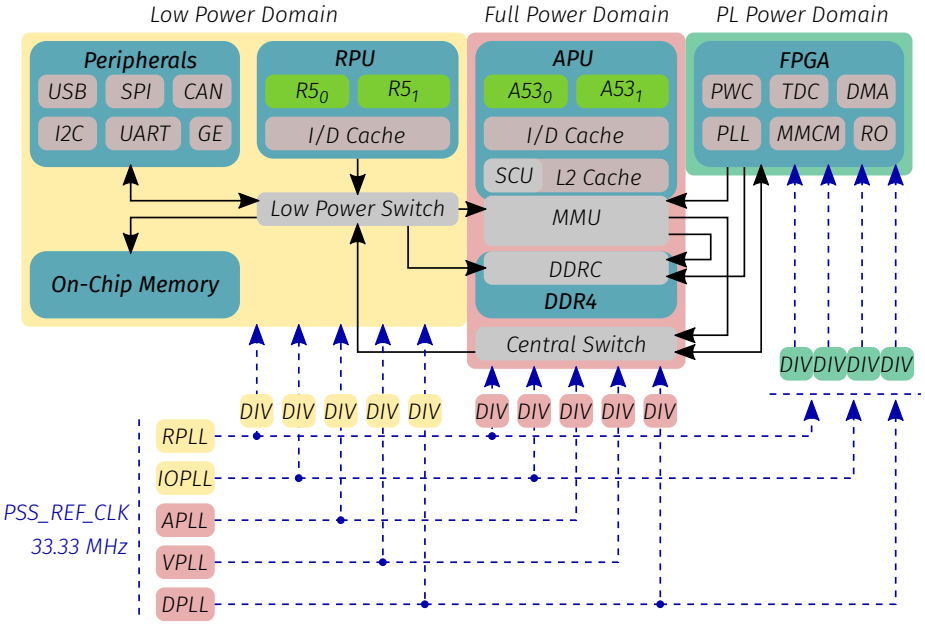

Figure 1: The architecture of the AMD-Xilinx xczu2cg.

A53 cores ($A53_0$ and $A53_1$ in Fig. 1), and a *real-time* processing unit (RPU), which includes an array of ARM Cortex-R5F cores ($R5_0$ and $R5_1$ in Fig. 1). 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-FPGA offers the possibility of implementing a wide range of customized accelerators.

In Fig. 1, in blue, we illustrate part of the clock tree in the Zynq Ultrascale+ SoCs. 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 different parts of the architecture.

From Fig. 1 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 energy efficiency of the system by allowing the system to shut down complete areas of the SoC when these are 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, for example RTOS or simply run 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.

3.2 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 divider and there are clocks which use two. All the dividers 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.

3.3 Frequency detection

Our work employs ring oscillator-based sensors due to their simplicity [ZS18; Gra+20]. 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 Johnson ring-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 Johnson ring-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 of the frequency change.

3.4 Characterization of the system

To understand the limits of the proposed covert 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. As reference, we generated a digital trigger through the processor's GPIOs. We then measured the width of these pulses. 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).

3.5 Covert channels within the APU

The first covert 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 a covert channel between different cores controlled by the same operating system. We used this driver, available in Xilinx' Linux, 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 different 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.

In Fig. 2 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.

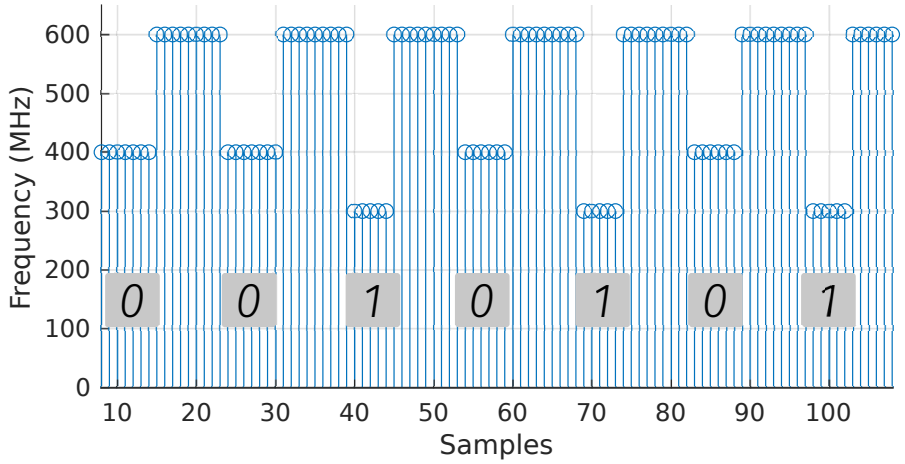

Figure 2: The transmission of a stream of bits over a covert channel from a core in the APU to a different core in the APU.

3.6 Covert channels between the APU and the FPGA

The second type of covert channels under evaluation were those that originate from an application executed in the APU 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 oscillators sourcing the FPGA from the processing system. This signal was used as the sampling clock for the delay sensor.

In the case of the Xilinx' distribution of Linux, 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 link. 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 scenario. On the other hand, we used a 10 μS delay in the acquisition of samples. In Fig. 3 we illustrate the results for this experiment.

From this experiment 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 didn't implement anything besides the sensor in the FPGA, thus we assume there were no data-dependent components in the experiment. Nonetheless, there ought to have been some influence from the activity of the processor system but it was deemed negligible.

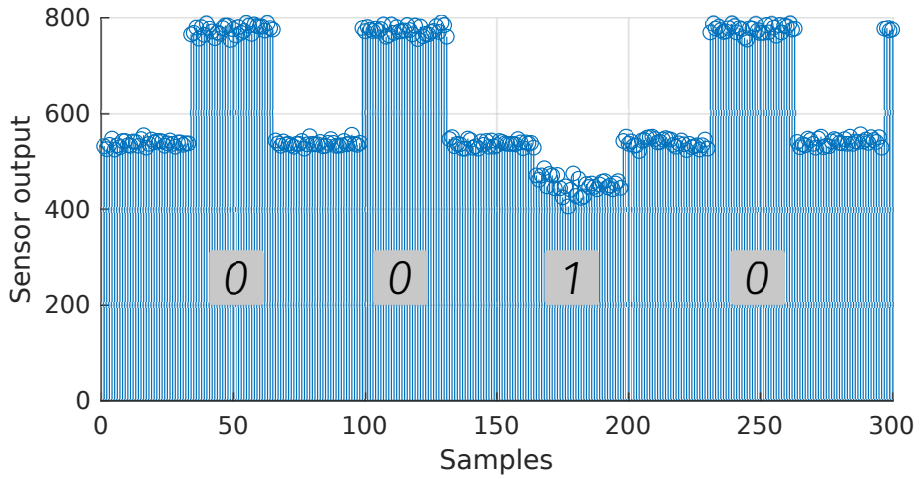

Figure 3: The transmission of a stream of bits over a covert channel from the Linux kernel to the FPGA.

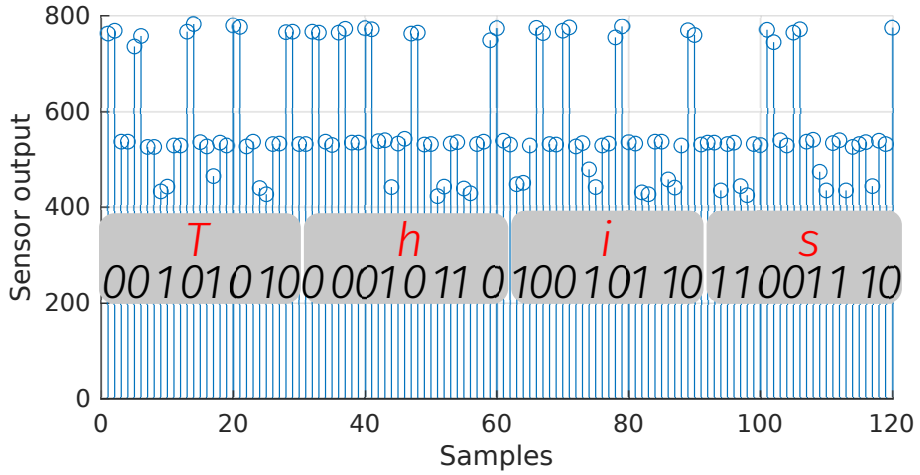

Figure 4: The transmission of a stream of bits over a covert channel from the Linux kernel to the FPGA. In this case the clocks were modified by editing the register values from the kernel space.

We could also observe how, for this experiment, 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 in particular to the `PLX_REF_CTRL` registers which con-

tain 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. 4.

4 Emulation

4.1 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.

4.2 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 most simulations. The sources for the kernel and other binaries can also be obtained from online repositories. 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 Xilinx binaries, including their distribution of Linux.

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 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 repositories available in `gem5.googlesource.com`.

4.3 The emulated platform

The APU of the Zynq Ultrascale+ Soc-FPGAs 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.

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 boot-loaders shipped with the *gem5* simulator. The kernel was our custom Linux binary. We used an Ubuntu image found in `www.gem5.org/documentation/general_docs/fullsystem/guest_binaries` and edited it in Linux to manually cross-compile and package the applications.

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. 3 as a base offset according to the frequency plus a ± 10 random component. At this point we didn't 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.

4.4 APU-to-APU covert channels

The first covert 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. 5. The main challenge to emulate the results from Subsection 3.5 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.

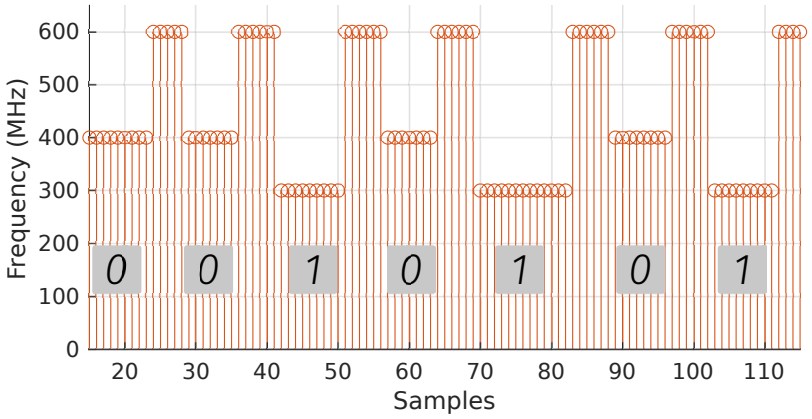

Figure 5: Emulating a covert channel from a core in the APU to a different core in the APU. These results are related to Fig. 2.

4.5 APU-to-FPGA covert channels

Next, we emulated the covert 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 are shown in Fig. 6. 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 parameterization 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 3.6.

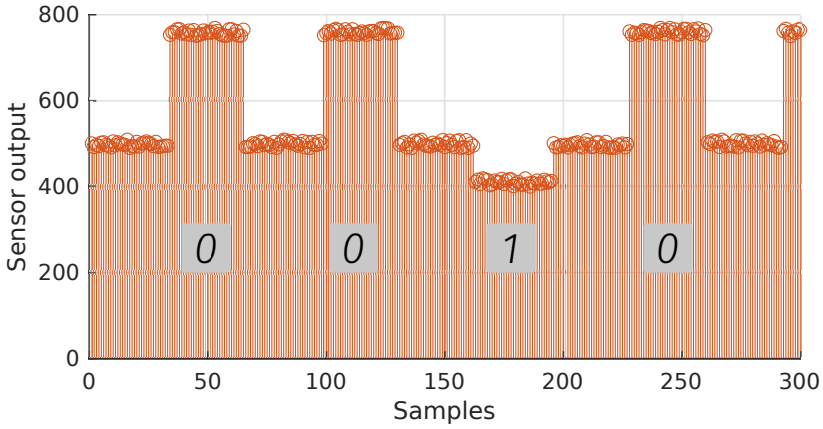

Figure 6: Emulating a covert channel from the Linux kernel to the FPGA. These results are related to Fig. 3.

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. 7. 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.

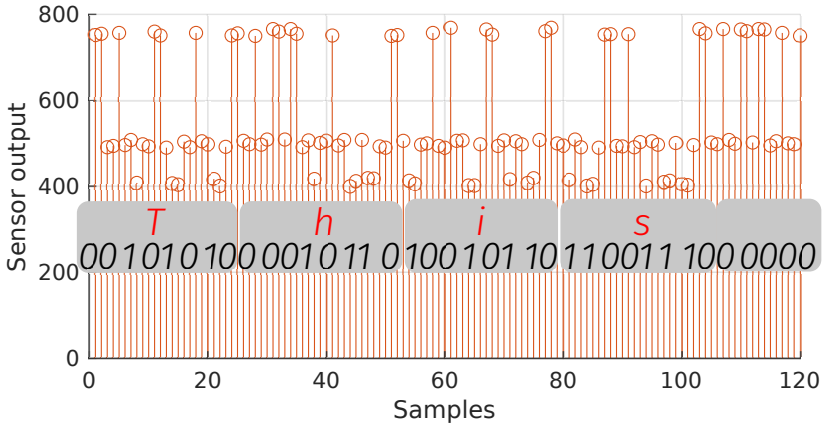

Figure 7: Emulating a covert channel from the Linux kernel to the FPGA. These results are related to Fig. 4.

5 Discussion

The proposed methodology allows to study the implementation of frequency-based covert channels in ARM systems. However, given the flexibility of *gem5* it is relatively 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 PLL to account for technology variations. The 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 be also possible to emulate non-ARM SoCs and study the proposed attacks in these platforms.

The simulations scripts used in this paper are provided under an open-source license so that future researchers can further explore the extent of frequency-based covert channels and design countermeasures. For example, architectural protections which seek to limit the interaction between the kernel space and the hardware can be implemented simply by changing the simulation binaries. Other protections such as those based on circuit modifications would require editing the *gem5* sources, but since the software is open source that is not really an issue. The files used in this paper can be retrieved from `https://github.com/CarlosAndresLARA/dvfs-gem5`.

6 Conclusions

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 parameterization of different components and values. This can bring the results closer to the expected observations.

The main contributions of our work have been to identify key limitations of the *gem5* simulator for the emulation of DVFS in generic kernels. We have also demonstrated that it is possible to approach the emulation of heterogeneous SoCs (that is, those with a reconfigurable core) through creation of custom *SimObjects* and how these can be used to derive accurate results. This is the main advantage of *gem5* over alternatives such as qemu which can only emulate the logical behavior of the platform.

Our work builds on publicly available resources which are accessible to anybody who is interested in this field of research. We employ the Open-Source *gem5* simulator as well as freely available kernel files which can be used without any cost. To facilitate the reproducibility of our results we also intend to make all the source code required under an Open-Source license in a public repository.

Data availability

The multiple sets of statistics used in our experiments as well as the scripts created for processing the data can be accessed freely on `https://github.com/CarlosAndresLARA/dvfs-gem5`.

Acknowledgements

The authors acknowledge the support of the French Agence Nationale de la Recherche (ANR), under grant ANR-19-CE39-0008 (project ARCHI-SEC).

References

[Ala+17] Murugappan Alagappan, Jeyavijayan Rajendran, Miloš Doroslovački and Guru Venkataramani. "DFS covert channels on multi-core platforms". In: *VLSI-SoC'17*. Dec. 2017. DOI: 10.1109/VLSI-SoC.2017.8203469.

[BB18] El Mehdi Benhani and Lilian Bossuet. "DVFS as a Security Failure of TrustZone-enabled Heterogeneous SoC". In: *ICECS'18*. Dec. 2018. DOI: 10.1109/ICECS.2018.8618038.

[Bin+11] Nathan Binkert et al. "The Gem5 Simulator". In: *SIGARCH Comput. Archit. New* 39.2 (Aug. 2011), pp. 1–7. ISSN: 0163-5964. DOI: 10.1145/2024716.2024718. URL: https://doi.org/10.1145/2024716.2024718.

[BL23] Lilian Bossuet and Carlos Andres Lara-Nino. "Advanced Covert-Channels in Modern SoCs". In: *HOST'23*. May 2023. DOI: 10.1109/HOST55118.2023.10133626.

[Cha18] Sumanta Chaudhuri. "A Security Vulnerability Analysis of SoCFPGA Architectures". In: *DAC'18*. Sept. 2018. DOI: 10.1109/DAC.2018.8465932.

[Cla18] Alvin Clark. *Xilinx Machine Learning Strategies For Edge*. Presented in Xilinx Machine Learning Live Presentations. 2018. URL: https://web.archive.org/web/20220407054353/https://www.xilinx.com/publications/events/machine-learning-live/san-diego/xilinx_machine_learning_strategies_for_edge.pdf.

[For+21] Quentin Forcioli et al. "Virtual Platform to Analyze the Security of a System on Chip at Microarchitectural Level". In: *EuroS&PW*. Sept. 2021. DOI: 10.1109/EuroSPW54576.2021.00017.

[GER19] Ilias Giechaskiel, Ken Eguro and Kasper B. Rasmussen. "Leakier Wires: Exploiting FPGA Long Wires for Covert- and Side-Channel Attacks". In: *ACM Trans. Reconfigurable Technol. Syst.* 12.3 (Aug. 2019). ISSN: 1936-7406. DOI: 10.1145/3322483.

[Gna+19] Dennis R. E. Gnad, Cong Dang Khoa Nguyen, Syed Hashim Gillani and Mehdi B. Tahoori. *Voltage-based Covert Channels using FPGAs*. Cryptology ePrint Archive, Report 2019/1394. Dec. 2019.

[Gra+20] Joseph Gravellier, Jean-Max Dutertre, Yannick Teglia, Philippe Loubet Moundi and Francis Olivier. "Remote Side-Channel Attacks on Heterogeneous SoC". In: *CARDIS'20*. Mar. 2020. ISBN: 978-3-030-42068-0. DOI: 10.1007/978-3-030-42068-0_7.

[GRS19] Ilias Giechaskiel, Kasper Rasmussen and Jakub Szefer. "Reading Between the Dies: Cross-SLR Covert Channels on Multi-Tenant Cloud FPGAs". In: *ICCD'19*. Nov. 2019. DOI: 10.1109/ICCD46524.2019.00010.

[JN16] Lennart Johnsson and Gilbert Netzer. "The impact of Moore's Law and loss of Dennard scaling: Are DSP SoCs an energy efficient alternative to x86 SoCs?" In: *Journal of Physics* 762 (Oct. 2016), p. 012022. DOI: 10.1088/1742-6596/762/1/012022.

[KFS18] Christoforos Kachris, Babak Falsafi and D. Soudris. *Hardware Accelerators in Data Centers*. 1st. Cham: Springer, Aug. 2018. ISBN: 978-3-319-92791-6. DOI: 10.1007/978-3-319-92792-3.

[Kha+18] Ahmed Khawaja et al. "Sharing, Protection, and Compatibility for Reconfigurable Fabric with AmorphOS". In: *USENIX'18*. Oct. 2018. ISBN: 978-1-939133-08-3. URL: https://www.usenix.org/conference/osdi18/presentation/khawaja.

[Lip+16] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice and Stefan Mangard. "ARMageddon: Cache Attacks on Mobile Devices". In: *USENIX'16*. Aug. 2016. ISBN: 978-1-931971-32-4. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/lipp.

[Mas+15] Ramya Jayaram Masti et al. "Thermal Covert Channels on Multicore Platforms". In: *USENIX'15*. Aug. 2015. ISBN: 978-1-939133-11-3. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/masti.

[Pro+19] George Provelengios et al. “Characterization of Long Wire Data Leakage in Deep Submicron FPGAs”. In: *FPGA’19*. Seaside, CA, USA, Feb. 2019. ISBN: 978-1-450-36137-8. DOI: 10.1145/3289602.3293923.

[Ram+18] Chethan Ramesh et al. “FPGA Side Channel Attacks without Physical Access”. In: *FCCM’18*. Sept. 2018. DOI: 10.1109/FCCM.2018.00016.

[Sch+18] Falk Schellenberg, Dennis R.E. Gnad, Amir Moradi and Mehdi B. Tahoori. “Remote Inter-Chip Power Analysis Side-Channel Attacks at Board-Level”. In: *ICCAD’18*. Nov. 2018. DOI: 10.1145/3240765.3240841.

[SMS20] Zeinab Seifoori, Seyedeh Sharareh Mirzargar and Mirjana Stojilović. “Closing Leaks: Routing Against Crosstalk Side-Channel Attacks”. In: *FPGA’20*. Seaside, CA, USA, Feb. 2020. ISBN: 978-1-450-37099-8. DOI: 10.1145/3373087.3375319.

[Spi+13] Vasileios Spiliopoulos, Akash Bagdia, Andreas Hansson, Peter Aldworth and Stefanos Kaxiras. “Introducing DVFS-Management in a Full-System Simulator”. In: *MASCOTS’13*. Aug. 2013. DOI: 10.1109/MASCOTS.2013.75.

[TS19] Shanquan Tian and Jakub Szefer. “Temporal Thermal Covert Channels in Cloud FPGAs”. In: *FPGA’19*. Seaside, CA, USA, Feb. 2019. ISBN: 9781450361378. DOI: 10.1145/3289602.3293920.

[TSS17] Adrian Tang, Simha Sethumadhavan and Salvatore Stolfo. “CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management”. In: *USENIX’17*. Aug. 2017. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tang.

[Yas+20] Yahya H. Yassin, Magnus Jahre, Per Gunnar Kjeldsberg, Snorre Aunet and Francky Catthoor. “Fast and Accurate Edge Computing Energy Modeling and DVFS Implementation in GEM5 Using System Call Emulation Mode”. In: *Journal of Signal Processing Systems* 2021.93 (May 2020), pp. 33–48. DOI: 10.1007/s11265-020-01544-z.

[ZBT10] Daniel Ziener, Florian Baueregger and Jürgen Teich. “Using the Power Side Channel of FPGAs for Communication”. In: *FCCM’10*. June 2010. DOI: 10.1109/FCCM.2010.43.

[ZS18] Mark Zhao and G. Edward Suh. “FPGA-Based Remote Power Side-Channel Attacks”. In: *S&P’18*. May 2018. DOI: 10.1109/SP.2018.00049.