A Security Vulnerability Analysis of SoCFPGA Architectures

Sumanta Chaudhuri
Télécom ParisTech, Université Paris Saclay
46, Rue Barrault
Paris, FRANCE 75634
firstname.lastname@telecom-paristech.fr

ABSTRACT

SoCFPGAs or FPGAs integrated on the same die with chip multi processors have made it to the market in the past years. In this article we analyse various security loopholes, existing precautions and countermeasures in these architectures. We consider Intel Cyclone/Arria devices and Xilinx Zynq/Ultrascale devices. We present an attacker model and we highlight three different types of attacks namely direct memory attacks, cache timing attacks, and rowhammer attacks that can be used on inadequately protected systems. We present and compare existing security mechanisms in this architectures, and their shortfalls. We present real life example of these attacks and further countermeasures to secure systems based on SoCFPGAs.

1 INTRODUCTION

Lately, in the domain of computer architecture there has been a shift towards heterogeneous system architecture (HSA). A prime example of that shift is the appearance of SoCFPGAs in the market. As a result, FPGAs are now tightly integrated into the network on chip (NoC) as compared to traditional PCIe FPGA based accelerator systems. This integration has been done mainly with performance in mind but it also opens up some interesting security issues.

The security of SoCFPGAs requires extensive discussions as it comprises of traditional FPGA security issues such as tampering, power side channels, or bitstream security, as well as SoC security issues such as *secure boot*, *digital rights management (DRM)*, and attacks against the operating systems (OS). In this paper we concentrate on the attacks mounted from FPGAs on the OS (e.g Linux, Android, RTOS) running the SoC, which aim to gain control of the SoC or steal secret information from programs running in the processor subsystem (PS).

1.1 Related Work, Contribution & Scope

A major class of attacks against OS are the direct memory access (DMA) attacks [20]. A DMA attack mounted from the network processor is presented in [6]. A keylogger from GPU [10], and other attacks mounted from GPU [23] can also be found in the literature.

A second class of micro-architectural side-channel attacks are the cache timing attacks [17]. Flush+Reload [11] technique mainly applies to the last level cache and measures the time to reload a page after flushing, Prime+probe technique [16] probes the state of the cache after an encryption run to find out useful information about the encryption step. Cache timing attacks have also been used to break the kernel space ASLR in operating systems [8].

A very well known attack against operating systems (OS) for privilege escalation is the buffer overflow attack [1, 15]. Over time several counter measures have been proposed for such attacks such as $W \oplus X$ (Write XOR Execute), Address space Layout Randomisation (ASLR). The DMA or cache timing attacks from FPGA can be used as a first step for mounting such attacks.

Recently, attacks which exploit hardware failures such as Rowhammer [9] have been reported. References [18, 22] show how to use Rowhammer to gain privilege in a Linux based system. In this article we also address this issue in the context of SoCFPGAs.

Our article makes the following key contributions:

- We present a comprehensive and comparative security vulnerability (SVA) analysis of the latest SoCFPGA architectures, for the system designers. This SVA is mainly geared towards OS-centric attacks.
- To our knowledge, we also present the first cache-timing attack through the coherency port and rowhammer attack mounted from an accelerator (FPGA or GPU). In the process we highlight a security vulnerability related to the accelerator coherency ports (ACP, CCIX, OpenCAPI) [21] and in general heterogeneous system architectures.

1.2 Outline

The rest of the article is organised as follows. In section 2 we present the attacker model. In section 3 we describe the recent SoCFPGA architectures and the existing countermeasures. In section 4 we present examples of direct memory attacks. In section 5 we discuss the cache timing attacks mounted through the cache coherency port. In section 6 we describe the rowhammer attacks and countermeasures in the context of SoCFPGAs. In section 7 we present example of attacks in each category. In section 8 we conclude.

2 ATTACKER MODEL & SCOPE

2.1 Scenario A (HPC)

In our first scenario, the attacker is an ordinary user of a SoCFPGA system, such as a heterogeneous cluster, where the attacker executes his/her own code under the guise of compute intensive tasks which need FPGA acceleration. He/She has the necessary privileges such as programming the FPGA, launching an accelerator kernel etc. on that system, without being superuser.

This is very much in line with the Amazon F1 platform user model where a user either brings in his own FPGA design or buys one from the Amazon marketplace (AFI, Amazon FPGA Image). [4]

In such a scenario the ordinary user can inject malicious code in the kernel data structures or system calls to ultimately get the superuser privileges, or information about other users. The SoCFPGAs Arria X, and Zynq Ultrascale+ series designed for HPC (High Performance Computing) applications market are vulnerable in this context.

2.2 Scenario B (Embedded)

In our second scenario, the attacker is a third party supplier of FPGA accelerator libraries to the users of an embedded system based on SoCFPGAs. (e.g in form of apps from the Android App Store).

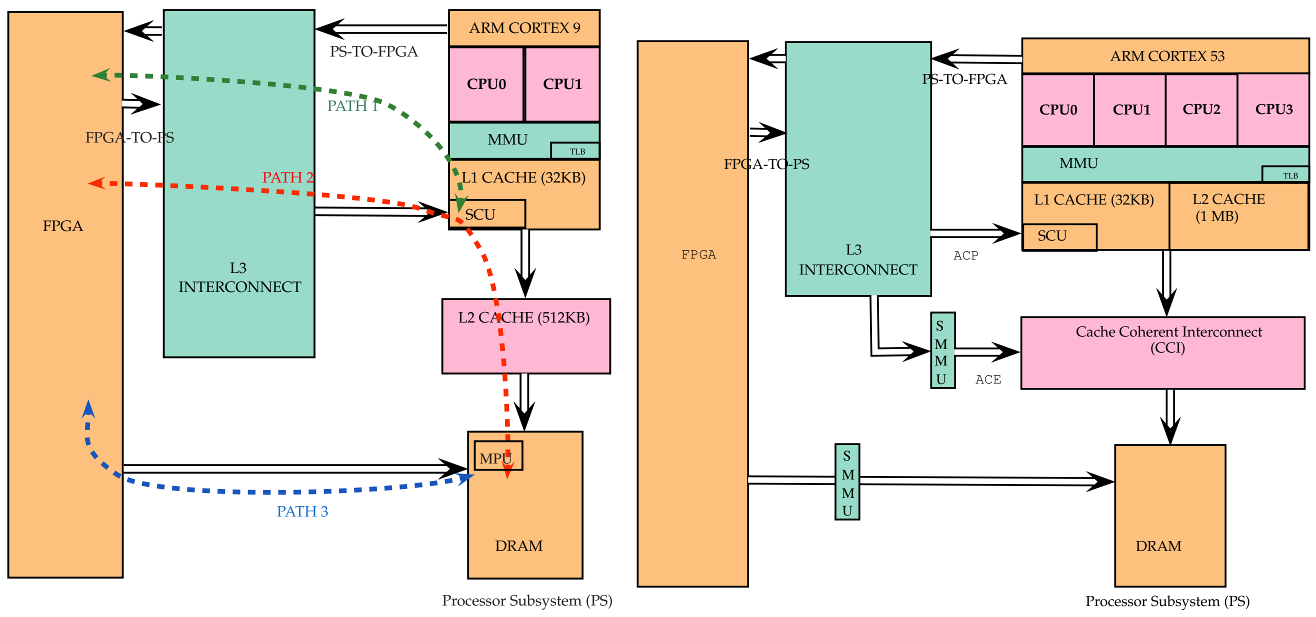
In such a scenario, the third party supplier can steal information on the real user. The SoCFPGAs Cyclone V, and Zynq 7000 series designed for the embedded systems market are vulnerable in this context.

3 SOCFPGA ARCHITECTURE & EXISTING SECURITY MEASURES

3.1 Trustzone Framework

To date, all of the SoCFPGAs are based on ARM, and consequently they use ARM's Trustzone framework to provide *secure boot flow*. Trustzone architecture has both software and hardware aspects. On

DAC '18, June 24–29, 2018, San Francisco, CA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3195979



(a) Cyclone V, Arria X, and Zynq 7000 Architecture Outline. MPU is Memory Protection Unit.

(b) Stratix X/Zynq Ultrascale+ Architecture Outline. ACE is AXI with Coherency Extensions. Stratix X does not have ACP.

Figure 1: A outline of different SoCFPGA architectures discussed in this paper. The DMA attack can use both the blue path (3) and the green path (1). The cache timing attack uses the green (1) path and red path (2), and the rowhammer attack uses the blue path (3).

the software side, Trustzone defines ”Secure” and ”Non-Secure” world abstractions with the ”Monitor” mode which handles the transition from one world to the another. These are standard in all SoCFPGAs discussed in this paper. Normally a secure Trusted Execution Environment (TEE) such as SierraTEE [19] or Toppers [14] runs in the secure world and a ”rich” OS (e.g Linux, Android, RTOS) runs in the normal world. For example in attack scenario B (pl.see 2.2) the TEE might contain mobile device’s OEM secret keys, and the normal world OS manages the user content. In this paper we address the security of the normal world OS and in particular Linux. Securing the normal world OS is also important as it might give the attacker access to mobile device sensors in scenario B, or other user’s files in scenario A.

In this article we limit our scope to the aspects of Trustzone which directly impact the interaction between the FPGA and the processor subsystem (PS). In Trustzone hardware, each master in a SoC could be declared as ”Secure”, ”Non-Secure”, or ”Per Transaction”.

3.2 Architectural Features

Firewalls. Firewalls are part of Trustzone framework. Each slave can be protected by a firewall, which blocks traffic depending on its security attribute ”Secure” or ”Non-Secure”. These firewalls are programmable at runtime. e.g during a secure payment operation access to memory can be blocked by various firewalls.

Memory Protection Unit. A memory protection unit present at the DDR controller can define memory zones, and control access to these zones from masters depending on *Address Range*, *Master Id*, and *Master Port*. e.g in Cyclone V SoCFPGA it is possible to setup 20 such rules with Zone Granularity of 1MB [3]. In Zynq 7000 devices it is possible to setup zones with 64MB Granularity [25].

Secure Cache Access. An extra bit is provided to indicate security of cache lines. The cache can store both secure and non-secure lines. The security state of the masters using the Accelerator Coherency Port (ACP) must be the same as that of the processors.

System MMU. In a processor only system, all memory access permissions are handled by the MMU (Memory Management Unit). A software thread can only see the memory pages mapped to its virtual address space. On top of it, each page has read write and execute permissions. A system MMU (SMMU) or IOMMU provides the same functionality to accelerators accessing memory. The main advantage of MMU is the ease of programmability as both the host program and the accelerator can use the same virtual addresses. It also provides fine grain security for memory access. The granularity is limited to MMU page size (4KB in most architectures).

In figures 1(a) & 1(b) we present typical SoCFPGA architectures available at the market today. Figure. 1(a) resembles the architecture of the Cyclone V, Arria V, Arria X, and Zynq 7000 devices based on ARM Cortex A9 processor. Figure. 1(b) is representative of the architectures of Zynq Ultrascale+, and Stratix X devices. Stratix X device does not have the ACP port, whereas Zynq Ultrascale+ has both ACP and CCI(Cache Coherent Interconnect).

Table 1 classifies existing architectures into four categories, showing the presence(✓) or absence of a feature(×). We refrain from categorically naming each device for two reasons

- We have only access to publicly available information through Technical Reference Manuals (TRMs). Some security related documents are available only with a non-disclosure agreement.
- Abstracting the security features and architectures from real devices will help the reader compare this study with future devices.

According to the TRMs [3, 25] Arch.1 closely resembles Cyclone V, Zynq 7000 and Arch.2 resembles Arria devices. Cyclone V & Zynq 7000 devices have firewalls and memory protection units, and only Arria X device has *Secure Cache Access* feature. None of them possess SMMUs.

Arch.3 and Arch.4 (table 1) closely resembles the Zynq Ultrascale+, & Stratix X devices respectively. The only difference between them being that Stratix X does not have any ACP port.

Table 1: Architectural Features

Features	Arch. 1	Arch. 2	Arch. 3	Arch 4.
Firewalls	✓	✓	✓	✓
MPU at DDR Controller	✓	✓	×	×
33rd 'NS' bit for cache	×	✓	×	×
System MMU	×	×	✓	✓
ACP Present	✓	✓	✓	×
CCI	×	×	✓	✓

Table 2: Security Vulnerabilities

Attack Type	Arch. 1	Arch. 2	Arch. 3	Arch. 4
DMA through CM	✓	✓	×	×
DMA through Cache	✓	✓	×	×
DMA Buffer Overflow	✓	✓	✓	✓
DMA SMMU Update	*n.a*	*n.a*	✓	✓
CTA (ASLR)	✓	✓	×	×
CTA (Program Behaviour)	✓	✓	✓	✓
Rowhammer Bit Flip in Read-Only Zone	✓	✓	✓	✓

Disassembly of libc geteuid32() system call 201

```
00072570 <geteuid>:
   72570:       b500            push    {lr}
   72572:       f04f 0cc9       mov.w   ip, #201         ; 0xc9
   72576:       f7a5 f9b3       bl      178e0 <gnu_get_libc_version+0x1a0>
   7257a:       f85d fb04       ldr.w   pc, [sp], #4
   7257e:       bf00            nop
```

Replacing system call 201(0xc9) with 73(0x49)

```
00072570 <geteuid>:
   72570:       b500            push    {lr}
   72572:       f04f 0c49       mov.w   ip, #73          ; 0x49
   72576:       f7a5 f9b3       bl      178e0 <gnu_get_libc_version+0x1a0>
   7257a:       f85d fb04       ldr.w   pc, [sp], #4
   7257e:       bf00            nop
```

DMA from FPGA: system call 73 returns 0 (effective id of root)

```
$ whoami
linaro
$./dma_from_fpga 0x49
$ whoami
root
$
```

Figure 2: The basic DMA attack, where the FPGA Trojan replaces the geteuid system call with a system call which returns 0.

3.3 Typical Accelerator Architecture

A typical FPGA accelerator consists of

- A processor thread also called *Host Program* which executes the part of the computing suitable for serial execution. It also programs the accelerator, launches and schedules "kernels" in the accelerator.
- A hardware block also known as "Accelerator Kernel" which is programmed into the FPGA, and communicates with the host program either through memory or registers.
- A Device Driver for the accelerator, which provides an API for low level actions to the ordinary users and host programs.

These accelerators could be written in a low level language like Verilog, or standard framework like OpenCL.

4 DIRECT MEMORY ATTACKS

4.1 Attacks

Direct memory attacks(DMA) are the simplest exploits that take advantage of inadequate memory protection schemes. In figure 2 we illustrate such an attack with an example. Here the attack target is the *geteuid()* function from GNU *libc* for which we show the disassembled code. We can see that this function makes a *System Call* 201 which returns the *effective user id*. This function from GNU libc is used by several programs such as *whoami* command from the bash shell. If the attacker can change the system call number (201) to another system call 36 (sync) or 73 (sigpending) which returns '0' the attack is successful. '0' is the *effective user id* of the administrative user 'root'. This is very similar to DMA attacks mounted from PCIe FPGA cards [7].

Although the attack might seem trivial, it is not so easy to provide adequate protection in each of the architectures. In the ideal case we would like to have the same permissions (r/w) for hardware running on FPGA as the thread running on host cpu. But for example, In Altera Cyclone V, we can set upto 20 rules to protect the entire memory space. However the processor memory is highly fragmented, and has much more than 20 contiguous zones. Similarly Xilinx Z7000 devices can have different memory permissions with a granularity of 64MB, whereas the host process memory zones have a granularity of 4KB.

We classify the DMAs in following categories:

DMA through Contiguous Memory. Standard accelerator drivers such as Intel OpenCL runtime uses a reserved "Contiguous Memory Area" to communicate with the FPGA. This CMA zone is necessary as FPGAs without MMUs can not use the paging scheme of the host program. We can setup two memory protection zones: one for CMA (e.g 64 MB read-write) and the rest of the memory non accessible to FPGA. However the CMA zone is still used by Linux kernel if memory is available, and consequently FPGA can modify any file or program memory mapped in this zone. Only restriction on Linux's usage of CMA zone is that it is only allowed to place "movable pages" in this zone [5]. e.g (File mappings, process pages)

DMA through Cache. In all recent SoCFPGA architectures, if it is present, the Memory protection unit (MPU) is situated at the DDR controller. This does not provide any protection for the cache lines. So if a cache line belonging to a protected zone is accessed through ACP, the FPGA can read/write this cache line at will.

DMA Buffer Overflow. For systems with SMMUs, the accelerator has the same protection (R/W/X pages) as the processor thread, but it can still perform buffer overflow within this page to jump into other programs memory space.

DMA through SMMU update window. In systems with SMMU, there is a lag between the update of the main MMU, and the SMMUs controlling the accelerator memory access, this window of time can be exploited by the FPGA, during process context switches.

4.2 Countermeasures

The official Linux kernel for SoCFPGAs maintained by the vendors [2, 24] do not setup any protected memory zones using MPU. It is upto the user to add these rules.

As shown in table 2, architectural features alone can not guarantee immunity from DMA attacks. It is necessary to activate countermeasures in software (either in OS, or device Driver) for full protection. One of the trivial countermeasures is to activate the firewalls at all times, but for practical purposes we suggest the following enhancements:

- **DMA through CM:** For these attacks a zone can be reserved away from the Linux Kernel during the operation of the accelerator. This results in a performance loss. SoCFPGAs with SMMUs are immune to this attack.
- **DMA through Cache:** SoCFPGAs with secure cache access, or SMMUs are immune to this attack. For others the only countermeasure is to block the access to cache by reconfiguring firewalls, whenever a critical code is running and using the cache. Another option is to block traffic with Ids that emanate from FPGA port in the MPU, and always use write-through cache. e.g Altera Cyclone V MPU has the capability to block traffic for a memory zone based on traffic Id [3].
- **DMA buffer overflow:** The only counter-measure for buffer overflow attacks is to add compiler checks in the verilog/OpenCL compiler to detect any attempt to overflow buffers. This is

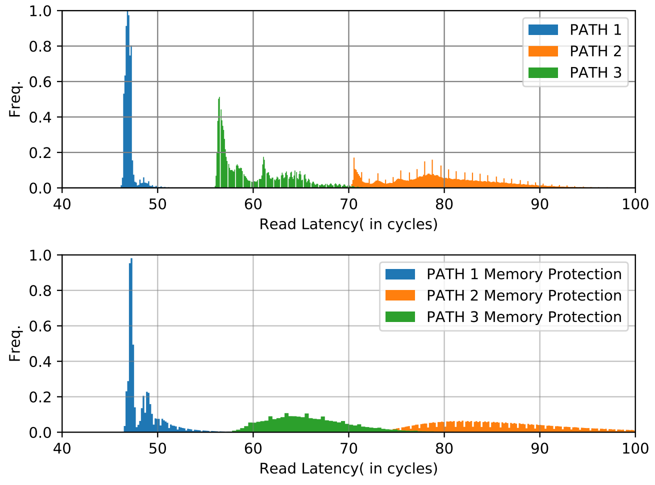


Figure 3: Latency Histogram of DDR (Main Memory) accesses through various paths. Paths 1,2, and 3 are illustrated in figure 1(a)

similar to checks performed by compilers (e.g gcc) for buffer overflow.

- **DMA through SMMU update window:** The DMA through SMMU update window is a problem for all SoCs [12]. The countermeasures proposed in [12] should be implemented for SoCFPGAs. Another method could be to block traffic through firewalls during context switch.

5 CACHE TIMING ATTACKS

5.1 Attacks

Cache timing attacks exploit information gained from the difference in latency of those memory accesses which are in the cache and those which have to be fetched from main memory. In figure 3 we show the latency histogram of the memory accesses through different paths from the FPGA to the main memory. We plotted this latency histogram by accessing a specified zone (256 MB) in DDR memory. These paths are illustrated in fig. 1(a) Path 1 is from the FPGA to the L2 cache through the ACP port. This path is activated when data is found in the cache. Path 2 shows the Cache miss trajectory, and Path 3 is the direct connection to DDR.

We can see in figure 3, that activating memory protection has no effect on the memory access latency. Figure 3 plots the latency with MPU blocking all accesses from FPGA. Although data returned by MPU is 0x0 or 0xDEADBEEF, the latency remains the same.

We classify cache timing attacks from FPGAs in two different categories:

CTA to break ASLR. Cache Timing Attacks (CTAs) have been used in processors to break kernel space ASLR [8]. The same attack can be mounted in SoCFPGAs with ACP ports. The processor thread or the *host program* continually accesses a target address (e.g libcrypto.so). The accelerator kernel in FPGA can guess the corresponding physical address by continuously scanning a memory zone (with access or protected). The target physical address will consistently have low latency. Before kernel version 4.0, the physical addresses of user processes were visible in Linux through /proc/$pid/pagemap, but to counter the threat from rowhammer [9] attacks, from kernel version 4.0-rc5 onwards only users with *cap_sys_admin* privilege can access this information [13].

CTA to guess program behaviour. Traditional cache timing attacks try to find out the workings of other programs (e.g cryptographic). For example, a certain program A frequently uses the exponent function from *libmath.so* which is a common read only library shared by all the processes. By measuring the access time of the starting address of this function, another process (or FPGA) can guess that this function is being used at that moment by program A. This is the basis of various attacks [11, 16]. In certain cases this program behaviour can be used to derive secret keys [11].

5.2 Countermeasures

There is always a trivial solution to block FPGA access to the Cache. This will certainly secure the system at the expense of performance, but it does not make sense since the use of the accelerators like FPGAs is mainly for performance. For practical purposes we suggest the following enhancements:

- **CTA to break ASLR:** The SoCFPGAs with SMMUs and *Secure Cache Access* are immune to this attack. For SoCFPGAs which use ACP to enhance performance, and which lack SMMUs can enhance security by using a SMMU programmed in the FPGA itself.
- **CTA to guess program behaviour:** Any SoCFPGA with a path from fpga to cache is vulnerable to this attack. The main application of these types of attacks is to guess cryptographic keys. So a possible countermeasure is to reconfigure all firewalls to block FPGA access to cache before launching a cryptographic application.

6 ROWHAMMER

6.1 Attacks

Rowhammer is a recently discovered vulnerability in the DRAM memories in the latest technology node($\sim$40 nm) [9]. By repeatedly accessing a DRAM row it is possible to disturb (bit-flip) the contents of the neighbouring rows. The threat of rowhammer is important as it can cause bit flips in read-only zones, or no-access zones if they are on the borders of a read-only zone.

Mounting a Rowhammer attack basically amounts to making a maximum no of accesses to a memory row within the refresh interval. The X-talk induced by rapid opening and closing of rows discharges the capacitances in the neighbouring rows [9].

In SoCFPGAs mounting rowhammer attacks is particularly easy. Because while rowhammering from CPU, for rapid accesses to DDR memory it is necessary to flush the cache. In ARM architecture cache flushing can not be done from the user mode but requires a special security co-processor(CP15) instruction. Having a FPGA directly connected to the DRAM alleviates this barrier to rowhammering. In figure 6(a) we present the rowhammering results from a Cyclone V SoC on the educational FPGA board DE1SoC. We can see that number of bit-flips induce depends on the pattern as noted in the original rowhammer experiments [9]. We can also see the increase in bit flips associated with refresh interval. We used an Activation Interval (Time between two reads) of $\sim$45ns. This is the minimum possible value for DDR3.

6.2 Countermeasures

Rowhammering can always flip bits in a read-only zone, which is a serious security vulnerability. In all the architectures discussed, there is a direct link from the FPGA to the DRAM memory, and as such this can't be hindered through existing security measures.

Rowhammer is random in nature, so the attacker needs to place the attacking spots on rowhammer sites through trial and error. Standard security measures such as ASLR (Address Space Layout Randomisation) can make this difficult for the attacker.

Other countermeasures could be

- Obfuscation of physical address. (This is used in Linux Kernel, since version 4.0)
- Error correction codes. Standard DRAM error correction schemes can detect upto two errors (SECDED [3]), but it has been shown that rowhammering can induce multiple bit-flips.

Disassembly of log10 function at python(2.7) numpy umath.so library

```
00006418 <log10@plt>:
    6418: 46c04778 ; <UNDEFINED> instruction: 0x46c04778
    641c: e28fc600 add ip, pc, #0, 12
    6420: e28cca44 add ip, ip, 0000068, 20 ; 0x44000
    6424: e5bcfd68 ldr pc, [ip, #3432] ; 0xd68
```

Replacing the jump to log10 function by jump to mod function

```
00006418 <log10@plt>:
    6418: 46c04778 ; <UNDEFINED> instruction: 0x46c04778
    641c: e28fc600 add ip, pc, #0, 12
    6420: e28cca44 add ip, ip, 0000068, 20 ; 0x44000
    6424: e5bcfd68 ldr pc, [ip, #3376] ; 0xd30
```

Figure 4: DMA code injection in numpy umath.so library.

- Target Row Refresh: refresh a row after a certain number of consecutive reads. proposed in LPDDR4 memory standard.
- Use of monitoring program in the CPU side to detect unusual hammering activity.
- Use of formal verification of FPGA IP code to detect hammering behaviour.

7 EXPERIMENTS WITH TROJANS

In this section we present three experiments, one in each category presented in this article. All of them assume the attacker model that we presented in section 2.

- The attacker is an ordinary user of the system, an user of the HPC cloud in scenario A, and code from third party library or apps in scenario B. He/she can launch programs only with user privileges.
- We conduct these experiments on an educational board DE1SoC comprising of Cyclone V SoCFPGA from Altera.
- We report the runtime, and used resources in table 3. We also compare these resource to a standard HPC kernel ”Asian Option Pricing”. The ”Asian Option Pricing” pricing kernel is taken from OpenCL design example.
- All trojans are implemented in OpenCL (Intel OpenCL SDK for FPGA 16.1) which follows the standard accelerator model presented in section 3.3.
- We use the Linux Kernel 3.18, with an Ubuntu 14.04 root file system, and the standard OpenCL runtime driver (16.1) from Altera for these experiments.

7.1 DMA code Injection

The goal of this trojan experiment is to modify the *log10* function from the *python numpy* library to *mod* function by code injection through DMA. We used the *DMA through CM* method to inject code. Python numpy loads a shared library called *umath.so* whose disassembled version is shown in figure 4. We can see that at address *6424* the ’log10’ (offset 3432) function is called. We want to replace this by the offset of the function mod (offset 3376). In this experiment, only Contiguous Memory (CMA) zone is writable by FPGA, the rest of the memory is no-access.

The trojan works in the following way.

- The host program creates memory pressure by allocating arrays, and using those arrays in memory.
- In a separate process, the numpy module from python is loaded which forces the load of the umath.so library.
- The FPGA accelerator kernel scans through the CMA area for the following string ”e28fc600;e28cca44;e5bcfd68” which is the signature of the log10 function.
- The above process is iterated several times until it succeeds in finding the above signature. Due to created memory pressure once in a while the shared library umath.so will be loaded in the CMA area.
- As the FPGA has read/write permissions in the CMA zone, it can now modify the log10 function at will.
- As this is a shared read-only library, shared by all python processes, other users will see the erroneous calculations.

It is difficult to perform this attack on GNU libc. GNU Libc is a special library and is used by almost all the running programs in linux, thus it is not possible to relocate libc. The above attack on a python log10 function is an illustrative example but can lead to catastrophic failures in scenario B (e.g drones) and sabotage other users’ programs in scenario A (FPGA cloud).

7.2 Cache Timing Attacks to break ASLR

The goal of this trojan experiment is to determine physical addresses given a virtual address. This information is deliberately obfuscated since Linux kernel 4.0. Although we use Linux kernel 3.18 due to dependency of the Intel OpenCL SDK, we don’t use the pagemap feature to determine physical addresses. This attack works even if the whole main memory is marked as no-access.

The attack works in two steps.

- In the first phase, the host program does nothing, and the FPGA accelerator kernel scans the memory space with a step of 4KBytes and offset equal to the virtual address offset. Virtual to physical address mapping is done with the granularity of the MMU page size (4K). Both virtual and physical addresses have the same offset.
- In the 2nd phase, the host program, accesses the target virtual address (e.g beginning of the GNU libc), and at the same time FPGA measures the latency, so the target address has a high probability to get low latency.
- We repeat this for several iterations to increase the probability of finding the right address.
- next we rank addresses w.r.t the latency data from both phases and we use the following equation to calculate final ranks.

$$rank = \frac{\left|rank_{latency}(phase1) - rank_{latency}(phase2)\right|}{rank_{latency}(phase2)} \quad (1)$$

  The rationale behind above ranking is to eliminate addresses which were present in the cache during both phases.

Due to ASLR, the FPGA kernel can not know the exact location of GNU libc in the memory. Through this attack it can pinpoint the physical location of this important library. This information can then be used for further attacks. In figure 5(a) & 5(b) we can see the 1/latency value of the target address and the differential rank. In fig 5(c) we can see for more than 300 iterations the target address comes up in the first 5 ranks. The address can be deduced accurately by using several different offsets within the same page.

7.3 Rowhammer: Create an error map

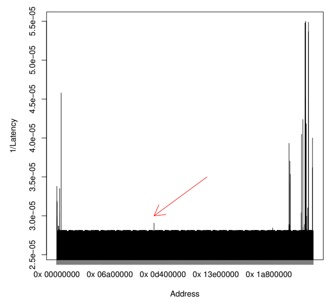
The goal of this trojan experiment is to create a rowhammer error map of the memory device with only user privileges. This is a first step in all the reported attacks by rowhammering [18, 22]. In this case we assume that the FPGA kernel has the same memory permissions as the host program. The attack works in the following way:

- The host program allocates a 4KB buffer, and writes a 32 bit signature at address 0 of the buffer.
- The FPGA kernel searches for this signature and determines the physical address of the buffer. Note. We can also use CTA to find the physical address if there are no read permissions.
- Next the FPGA kernel rowhammers this block and creates an error map. as this is a dummy buffer we are sure that it would not lead to kernel malfunction/hang.
- This process is repeated. If buffers are allocated in the same zone we restart the process until the whole zone is covered.

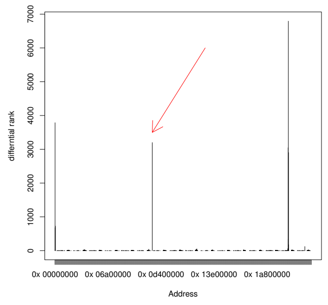
In this way we are able to create an error map of any zone in the user space. This information can then be used to perform attacks described in [18, 22]. The error map is shown in figure 6(b). In a SoCFPGA with SMMU only a temporary map based on virtual addresses can be constructed.

8 CONCLUSION

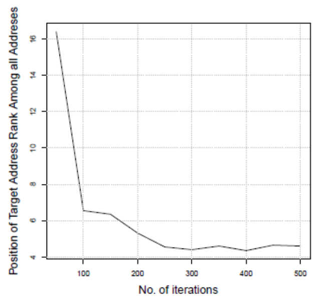
In this article we have presented various security vulnerabilities related to recent SoCFPGA architectures, and we have limited our



(a) 1/Latency plot for phase 2

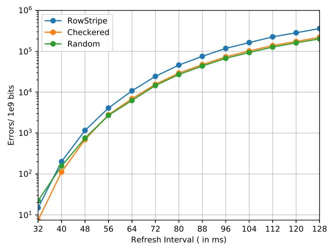


(b) Differential Ranking

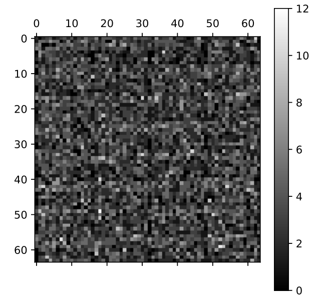


(c) Target address rank with iteration.

Figure 5: The raw cache access latency data are plotted in 5(a) for phase 2. 5(b) plots the differential ranking metric. We can see the target address stands out from the rest (red arrow). 5(c) shows target address comes up within first 5 for more than 300 iterations



(a) Error rate with Refresh Interval



(b) The error map of a 256 MB zone. Each square represents 64KB.

Figure 6: The Rowhammer Exploit

Table 3: Synthesis Results & Runtime

	EXP. A	EXP. B	EXP. C	Ref. Kernel
LEs	8145	17767	9724	216146
FFs	15434	33202	18058	375931
RAMs	275Kbit	269kbit	283Kbit	345Kbit
DSP	0	0	0	1064
Freq.(MHz)	143.55	131.84	149.99	143.55
Runtime	∼10min	∼10s	16 Hours	NA

scope to mainly OS-centric security issues. We have discussed various attacks and mitigation techniques, and we have presented illustrative examples of attacks in each category. The target audience for this paper are the system designers who plan to use SoCFPGAs in their system and do not want to compromise on security nor on performance. Although some architectures (with SMMU) are more robust, architectural features alone does not guarantee the security of SoCFPGAs, and it is necessary to take security measures on a case-by-case basis depending on the application.

REFERENCES

[1] *Buffer Overflow Attacks,James C. Foster and Vitaly Osipov and Nish Bhalla and Niels Heinen*. 2005.

[2] Altera. Rocketboard linux kernel. https://github.com/altera-opensource/linux-socfpga. Accessed: 2017-11-21.

[3] ALTERA. *Altera Cyclone V Technical Reference Manual*. ALTERA, San Jose, CA, USA, 2016.

[4] AMAZON. *Amazon EC2 F1 Instances*. AMAZON. Accessed: 2017-04-12.

[5] J. Corbet. Pagemap: security fixes vs. abi compatibility. https://lwn.net/Articles/642069/. Accessed: 2017-04-12.

[6] L. Duflot, Y.-A. Perez, and B. Morin. What if you can't trust your network card? In *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection*, RAID'11, pages 378–397, Berlin, Heidelberg, 2011. Springer-Verlag.

[7] U. Frisk. pcileech: Direct memory access (dma) attack. https://securityonline.info/pcileech-direct-memory-access-dma-attack-software-2/. Accessed: 2017-11-21.

[8] R. Hund, C. Willems, and T. Holz. Practical timing side channel attacks against kernel space aslr. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 191–205, May 2013.

[9] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. ISCA '14, pages 361–372, Piscataway, NJ, USA, 2014. IEEE Press.

[10] E. Ladakis, L. Koromilas, G. Vasiliadis, M. Polychronakis, and S. Ioannidis. You can type, but you cant hide: A stealthy gpu-based keylogger. 2013.

[11] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy*, pages 605–622, May 2015.

[12] A. Markuze, A. Morrison, and D. Tsafrir. True iommu protection from dma attacks: When copy is faster than zero copy. *SIGARCH Comput. Archit. News*, 44(2):249–262, Mar. 2016.

[13] M. Nazarewicz. A deep dive into cma. https://lwn.net/Articles/486301/. Accessed: 2017-11-21.

[14] U. of Nagoya. Toyohashi open platform for embedded real-time systems. http://www.toppers.jp/en/safeg.html. Accessed: 2017-11-21.

[15] A. One. Smashing the stack for fun and profit. *Phrack*, 7(49), November 1996.

[16] D. A. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: The case of aes. In *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, CT-RSA'06, pages 1–20, Berlin, Heidelberg, 2006. Springer-Verlag.

[17] C. Percival. Cache missing for fun and profit. In *Proc. of BSDCan 2005*, 2005.

[18] M. Seaborn and T. Dullien. Exploiting the DRAM rowhammer bug to gain kernel privileges.

[19] Sierraware. Sierratee trusted execution environment. https://www.sierraware.com/open-source-ARM-TrustZone.html. Accessed: 2017-11-21.

[20] P. Stewin and I. Bystrov. Understanding dma malware. In *Proceedings of the 9th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'12, pages 21–41, Berlin, Heidelberg, 2013. Springer-Verlag.

[21] J. Stuecheli, B. Blaner, C. R. Johns, and M. S. Siegel. Capi: A coherent accelerator processor interface. *IBM Journal of Research and Development*, 59(1):7:1–7:7, Jan 2015.

[22] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. CCS '16, pages 1675–1689, New York, NY, USA, 2016. ACM.

[23] G. Vasiliadis, M. Polychronakis, and S. Ioannidis. Gpu-assisted malware. *Int. J. Inf. Secur.*, 14(3):289–297, June 2015.

[24] Xilinx. The official linux kernel from xilinx. https://github.com/Xilinx/linux-xlnx. Accessed: 2017-11-21.

[25] XILINX. *Zynq 7000 Technical Reference Manual*. XILINX, 2016.