An Overview of Field-Programmable Gate Array Security

Keaten Stokke, Computer Systems Design Laboratory
University of Arkansas, Fayetteville, AR
knstokke@uark.edu

1. Abstract

As commercial and government applications for Field-Programmable Gate Arrays (FPGAs) grow more prominent, the need for security grows with them to protect vital operations and sensitive data. Security, both hardware-based and software-based, remains an active area for improvement and study for researchers and developers due to the increasing number of threats to embedded systems. Because FPGAs require tight integration of both software and hardware to remain flexible, both areas must be addressed when designing secure systems. The focus of this paper is to aid designers in the development of secure FPGA-based systems by presenting a variety of attacks and exploits that show vulnerabilities within FPGAs hardware and software. The Xilinx Zynq UltraScale+ MPSoC FPGA, specifically the ZCU104, will be analyzed to showcase its built-in features for countering attacks and its methods of protecting sensitive data and Intellectual Property (IP).

1. Introduction

The age of information is alive and well due to the widespread adoption of the internet. As computers advanced in capability they began to transform everyday life across the globe from personal use at home to critical applications in commercial businesses and government entities. Each day computers are being used for new applications that are increasingly requiring more computational power to handle intensive tasks. Application-Specific Integrated Circuits (ASICs) are growing more sophisticated with processors that can pack more than a billion transistors on them. ASICs, however, are extremely expensive to design and fabricate and will inevitably grow obsolete as Moore's law continues to push the boundaries of computing power. Field-Programmable Gate Arrays (FPGAs) have been around since the ’80s, but have made a come back due to their ASIC-like performance and reconfigurability. FPGAs are Integrated Circuits (ICs) that are made to be configured by the consumer after manufacturing. This makes FPGAs an extremely attractive alternative to building an ASIC because they can be reprogrammed and repurposed for an application by using an array of Programmable Logic Blocks (PLBs) within the hardware. General-purpose processors on PCs are designed to handle a wide range of tasks from browsing the internet to creating documents whereas an FPGA acts like a reprogrammable processor that can run a dedicated application. This results in a significant increase in computing performance for applications that PCs can’t offer and at a cheaper cost in engineering time and money than ASICs.

When the first computers and PCs were developed, security was never a priority because these machines typically existed as standalone devices. When the internet became prevalent, it suddenly connected all of these computers to create a vast network that allowed near-instant communications with other machines and people across the world. It wasn't until the turn of the century that security started to become a concern when the first wave of viruses and attacks in computer systems occurred. The rapid pace at which technology has been advancing has left security for these devices behind the curve. Now, major companies and application developers have seen the consequences of major data breaches and hacks of systems that have left their users and systems vulnerable. Security for computers and embedded systems is now one of the most important areas of focus for companies, and FPGAs are no exception. FPGAs have been used for a wide range of applications that require intense security such as aerospace and defense, data centers, scientific instruments, wired and wireless communications, and high-performance computing.

Like robbers that are always interested in stealing money from banks, hackers are always interested in stealing information from embedded systems. The motives of an attacker vary from attacking systems for fun and to see if it's possible, to obtain sensitive data, such as passwords and personal information, that can be used for financial gain or internal motive. It should go without saying that hackers will always be around, so protecting the security of embedded systems and data within applications is crucial. Designers should take every measure possible to protect their applications and systems or risk losing everything to the hands of an unknown adversary that sits behind a computer from a distance. With that being said, FPGAs do not reside in the average household. They are used for computationally intensive tasks that are of great importance and require significant technical knowledge to use and program. This puts a target on FPGAs as they are likely to contain information that would be valuable in the hands of others. Whether it be commercial applications for large companies or mission-critical applications for governments, the applications and data within their FPGAs are undoubtedly valuable in the hands of their competition or enemies.

While ASICs can be designed with security in mind, they can't be updated or modified when a new security vulnerability presents itself. FPGAs benefit from their flexibility where if a security vulnerability exists within the software, they can be updated and reprogrammed to mitigate the problem. All major vendors of FPGAs, the biggest being Xilinx and Intel's Altera, offer extra security solutions that developers can take advantage of to protect their Intellectual Property (IP). The most common form of security for FPGAs is bitstream encryption and authentication. A bitstream contains the configuration data for the FPGAs Programmable Logic (PL) and is the heart of its application. This piece of data is responsible for loading the IP and system design onto the FPGA and configuring the PLBs to meet the needs of the application; it is a prime target for attackers. If a bitstream is acquired by an attacker they can reverse engineer it to reveal the functionality of IP and ultimately the entire design. While this method of security is certainly helpful in protecting IP, an attacker with unlimited time, money, and resources can surely find a way around it. This notion applies to all embedded systems. A large variety of attacks exist that can bypass security measures within embedded systems with new attacks being created and discovered with each passing day.

The goal of this paper is to present known attacks on computer hardware, with a focus on FPGAs, so that developers can be informed of the security exploits that exist in the world today. With a better understanding of the types of threats that are known to expose systems and data, developers can put up a fight by adding more security measures to their designs to reduce risk. The next section of this paper presents a background on known attacks to embedded systems, and previous works that showcase the feasibility of various attacks as well as weaknesses that FPGAs have. After that, Section IV will discuss the hardware and software security features for the Xilinx Zynq UltraScale+ MPSoC family of FPGAs because they serve a wide range of applications and contain more security features than previous FPGA families. Section V will present the recommended security practices to take when developing a design for an FPGA. Lastly, this paper will end with a conclusion in Section VI that describes the work that was presented.

I. Background and Previous Works

Hardware and software are the two major areas that make up an embedded system and that attackers target when they attempt to gain access to a device. Typically, the scope of a software developer begins and ends with the code of the application they create. They can be held responsible for encrypting and securing data within an application, but if the device their application runs on is compromised by an attack then it is out of their hands. Due to the tight integration of hardware and software on FPGAs, the developer is responsible for both which exposes them to a greater amount of risk. All ends of the development cycle have to be considered when designing FPGA-based applications. Xilinx takes responsibility for its hardware up until it gets into the hands of the consumer. They state on their webpage that they handle the supply chain responsibilities with authorized suppliers, supply chain protection, and anti-counterfeiting measures. They also take responsibility for encryption authentication and the algorithms that their hardware implements. They have a shared responsibility with the consumer to utilize things such as a secure OS, use of Arm's TrustZone, and isolated design flow protections. They leave the consumer responsible for the implementation of digital signatures (such as watermarking), user passwords, and tokens. While Xilinx takes a lot of measures to enable security in their devices, they state that no system is completely immune from attack and that with enough time, energy, resources, and money, any system can be compromised. It is up to the developer to make the decision of performance versus security because adding features for a more secure system typically leads to a reduction in performance which could render the use of an FPGA obsolete. With that being said, it is crucial for the developers to be aware of common attacks to their systems and do their best to design a system that can defend against them while functioning as expected.

FPGAs are subject to the same kinds of attacks that all Integrated Circuits (ICs) are, however, these types of attacks are less enticing because FPGAs are a commercial product and are duplicated for consumer use. Uncovering exactly how an FPGAs hardware works is certainly an advantage to an attacker, but it does not yield them the same results when compared to attacking an Application Specific Integrated Circuit (ASIC) where they can uncover secrets about the operations of the chip. It is, however, worth mentioning what kind of attacks ASICs are vulnerable to, primarily invasive attacks, so that the FPGA designer has a better understanding of the threats that exist.

A. Types of Attacks

Non-Invasive attacks will be discussed first. These attacks, as described by their name, are non-invasive and do not compromise the physical integrity of the hardware. Rather these attacks use methods of closely analyzing the hardware and feeding it known data to see how the hardware operates under certain conditions. Within this category of attacks exists subcategories known as black-box attacks, Side-Channel Attacks (SCAs), and Fault Injection Attacks (FIAs), all of which FPGAs are vulnerable to. A data remanence attack can be performed by bringing the hardware into an environment of -20 C which causes data to remain within the volatile memory. The attackers can then read the contents of the volatile memory to uncover data that was never intended to be exposed. Brute force attacks can be performed on hardware by continuously guessing cryptographic keys in order to decrypt sensitive data.

FIAs come in many forms and is a popular form of attack on FPGAs. A local heating FIA is performed by using a high powered laser to selectively heat small areas on the hardware. This laser can be hot enough to change a voltage threshold, but not enough to damage the components. The change in voltage threshold can cause the circuit to leak sensitive data and execute on faulty data. This attack requires a lot of trial and error on the location of the heating to determine glitches. Flash glitching is an FIA that is performed by using a magnified camera flash on the hardware components that can cause mass glitching. Tinfoil can be used to cover sections of the hardware so that selective glitching can be performed on specific areas of the hardware. This method, like local heating, requires a lot of trial and error to discover where the glitches bring out vulnerabilities. Laser glitching is very similar to the local heating FIA but uses an infrared laser to selectively glitch small areas on the hardware with greater precision. Attacks like an Electro-Magnetic (EM) FIAs make it an urgent problem to find an effective solution because prevention is difficult [28]. The same work proposed an FPGA-based emulation method for security evaluation of IC design against FIAs. The flow is automated and their scan chan-based fault injection mechanism and management approach reduces area and overhead and enables fast fault injection emulation. This work is useful for developers who want to evaluate their design and its resilience to certain forms of FIAs.

Side-Channel Attacks (SCAs) are the most popular form of attacks to FPGAs and come in a variety of forms. These attacks typically require probing the FPGA to acquire data on how the operations are being performed and then tailoring an attack based on the results. A timing SCA is performed by feeding the FPGA a known amount of data and counting the number of clock cycles it takes for a function or subroutine to execute which can leak sensitive data. This method is popular in reverse-engineering the system and for reducing the number of guesses in a brute force attack. For example, if a certain instruction was fed into a function and it required more clock cycles than another instruction, an inference can be made on how long a loop runs for inside the algorithm. Timing attacks exploit data-dependent differences in calculation time in cryptographic algorithms. A way to thwart this type of attack is by modifying the algorithms by inserting dummy operations and delays so that an accurate time of a function's execution can not be made. Clock glitching is a type of FIA that is based on the timing type of SCA. It uses information acquired from the timing attack to feed the system clock bursts are double the normal speed that can cause critical timing errors, prevent flip-flops from latching correct data, skip instructions, and prevents security fuses from setting properly. Power analysis SCAs are performed by measuring the current consumption at different locations on the hardware to determine how the outcomes of functions and subroutines. Logic circuits typically consume differing amounts of power based on their input data. Common attacks in this category are Simple Power Analysis (SPA) and Differential Power Analysis (DPA). By measuring current consumption the attacker can use this acquired data, much like in timing SCAs, to reverse engineer the system and data flow, and reduce the number of guesses when using brute force [30]. This SCA is very simple and cheap to implement; methods to deter this attack are power balancing, randomization, and obfuscation [16]. Voltage glitching is an FIA that uses the data acquired from the power analysis SCAs to give the system a burst of high or low voltage. This burst causes critical timing errors and can prevent security fuses from setting properly, changing control logic outputs, and change memory amplifier outputs. SCAs come in many other forms too such as optical, acoustic, and electromagnetic.

Invasive attacks are more geared towards reverse engineering and exposing the architecture of ASICs because they vary from chip to chip. These forms of attack typically partially or completely destroy the hardware in the process in order to gain information about their behaviors. Decapsulation is done by using acids to remove layers from the components of a chip to provide physical access to all sections of the IC. This ultimately leads to the understanding of the entire design and layout of the IC and promotes reverse engineering. Layout reconstruction is performed by using very expensive imaging technology that photographs a layer of the chip, removes the layer, and continues all the way through the IC. From this, a netlist is able to be constructed from all of the images and all functions of the IC can be reverse-engineered. Memory extraction is the process of removing the memory, ROM or RAM, and scanning the contents to extract sensitive data. IC modification is another form of attack that uses various methods of cutting to open up the IC and expose layers. Known methods are laser cutting, test point creation, and wire bonding. These methods are sophisticated and require extensive knowledge of IC design in order to determine the netlist. Lastly, various methods of micro-probing are used to acquire information about a design. Eavesdropping is performed by listening to individual control lines and data buses to extract information from the hardware which bypasses on-chip protections. A signal injection attack is used by connecting probes to control signals within the IC and injected values that can modify memory contents and forcefully bypass security controls. There are forms of FIA that exist that are qualified as invasive attacks. Such a method uses two probes and injects a high voltage that can destroy select transistors and components that can ultimately leak sensitive data.

Many variations exist for each of the types of attacks listed above. They come in different forms with different techniques that are all geared at stealing information about the design of the ASIC itself or the data that is contained within the hardware. Most invasive attacks are aimed at ASICs because the goal is to uncover the layout and design of the hardware. Because FPGAs are commercial products, they are better understood in terms of layout and design and are less unique in terms of functionality. FPGAs are still subject to invasive attacks however and they can be subject to an invasive attack that is aimed at extracting data within its memory. The biggest concern for FPGA developers is non-invasive attacks, primarily SCAs. If an attacker can extract a secret key from the FPGA without destroying it, then they can continuously extract sensitive information from that FPGA and others virtually undetected.

One important detail to note about all of the attacks listed above is that they require physical access to the hardware. In every case, the hardware needed to be physically present for the attackers to probe information from it and run their subsequent attacks. This is a very important detail because if an FPGA can be physically locked down and out of the hands of attackers, these vulnerabilities seem to disappear. Some of the listed attacks can be performed quickly, but most of them require the FPGA to be running for an extensive amount of time to extract information from it, then that information has to be analyzed before going back and trying to break the encrypted keys; this is known as the interaction phase and the analysis phase.

B. Previous Works

The authors in [6] and [32] discuss the motivations for FPGA security and some of the features that they offer to protect them. They present threats such as cloning, where adversaries can copy the FPGA programming and run it on an identical device to claim it as their own, and overbuilding where an adversary, such as a contract manufacturer, builds additional systems and inserts illegitimate bitstreams onto those systems without the designer's approval. Tampering is presented where the adversary can employ additional logic that leaks information or disable parts of the application that may disable security measures. Spoofing is where an adversary replaces the existing FPGA bitstream with their own bitstream and can capable of compromising the system, even if the original design was reverse-engineered. They acknowledge that Denial of Service (DoS) attacks are irrelevant and that there is no real protection from an adversary smashing the FPGA with a hammer, but that the adversary is most likely trying to circumvent security measures to obtain a unique key. This paper provides a basis for FPGA security from its inception to the present and contains a lot of general information about the security of these devices.

The work presented in [7] discusses the manufacturing flow of FPGAs, general security in the form of bitstream encryption and authentication, Physically Unclonable Functions (PUFs), tampering, bitstream scrubbing and key clearing, and Single-Chip Cryptography (SCC). Their work is an overview of a lot of the features that are offered on the Zynq UltraScale+ MPSoC FPGA but without much elaboration. Hardware Trojan Threats (HTTs) are discussed in [8] which is somewhat in-line with what this paper is concerned about. HTTs primarily refers to the implementation of a trojan when the hardware is manufactured; with FPGAs, we assume that the hardware is trusted when received from the vendor. The authors talk about how even with current formal verification and hardware fault detection methods HTTs are not capable of being detected due to how rare it is to activate the trigger for one. They further discuss the different types of triggers that would activate a hardware trojan and how it could compromise a system. Granted, if an HTT is embedded within an FPGA, it would undoubtedly compromise the application on the FPGA but to detect one requires extensive testing which doesn't always work. [31] presents a method of accelerating the activation of HTTs so that they can be detected that is based on the relation between world-level signal statistical properties and the bit-level transition activity to ultimately increase the transitions of rare-bits.

[10] provides an interesting spin on how attacks can be carried out on FPGAs. The first model they use is that the attacker is an ordinary user of the system with extensive knowledge of the application and they execute their own code. This scenario primarily focuses on High-Performance Computing (HPC) applications where the code can be executed and hidden from others by computing with the rest of the tasks. The second scenario they explore is where the attacker is from a third-party supplier of FPGA accelerator libraries, or IP, and the code is assumed to be trusted, like an app from the App store on a smartphone, and it can steal information. The authors go into detail on a variety of attacks that leave FPGAs vulnerable and countermeasures that can be taken to avoid a security flaw. One of which is a Direct Memory Attacks (DMA) through contiguous memory, caches, buffer overflows, and SMMU window updates. Countermeasures suggested are to provide compiler checks in the Verilog/OpenCL compiler to detect any attempt to overflow buffers and to use secure cache access (SMMU) or reconfigure firewalls to limit access to the cache. Another attack discussed is the rowhammer attack that is performed by repeatedly accessing a row in the Dynamic-RAM (DRAM) which makes it possible to disturb the contents of neighboring rows. This type of attack is easily carried out on SoCFPGAs and countermeasures are obfuscation of the physical address, Error Correction Codes (ECCs) which can only detect up to two errors, refresh the row after a certain number of consecutive reads, use of a monitoring program in the CPU to detect unusual activity, and use of formal verification.

The work demonstrated in [11] is highly relevant to the discussion of this paper where the authors launched an attack on a Zynq-7000 SoC FPGA through the use of malicious hardware blocks. They were able to successfully modify the FPGAs public key at run-time using a malicious IP core. Their IP implemented hidden functions that scan the RAM memory, identifies the header of the public key and overwrites the stored public key with a key known to the attacker. However, one of the prevention methods to this attack was to use the UltraScale+ family of FPGAs because they contain a System Memory Management Unit (SMMU) that can dynamically manage the access rights of the peripherals. A survey of modern FPGA security features is presented in [12]. An attack they discuss that has not been mentioned yet is a bitstream readback attack. Early in FPGA history, a feature called bitstream readback was developed to assist in debugging. By using a single bit in the bitstream, this feature can be enabled or disabled. With older FPGAs, an adversary can use the Joint Test Action Group (JTAG) programming interface to flip this bit and enable a readback attack where the entire bitstream can be stolen. However, this attack is not as relevant as it used to be because bitstream encryption is a common practice now where even if an adversary managed to get ahold of it, they wouldn't be able to decrypt it. Furthermore, using the UltraScale+ family of FPGAs, this kind of attack could prevent because of the anti-tampering mechanisms that the CSU provides. It should be acknowledged that hardware faults, like bit flipping, can be caused by environmental factors such as radiation, extreme temperature, and sudden power change. ECCs usually can spot and fix this problem within a system if they are applied to the design, but these faults can be used as an advantage for adversaries.

BRAM tampering is demonstrated in [13] to break AES encryption, and while the attack is practical, it has little relevance to the FPGA discussed in this paper. BRAM is commonly used for other IP cores within a design that is running on an FPGA and the authors state that if this is the case the attack might not work properly. The premise of the attack works by attacking the bitstream and loading the modified version onto the GPA; they even propose a method for operating on an encrypted bitstream. However, they state that the most effective countermeasure against the attack is simply authenticating the bitstream on the FPGA which the UltraScale+ MPSoC is capable of. Running parallel to this work, [26] demonstrates the ability to reconstruct an AES key from decayed memory. By introducing the FPGA to very low temperatures, data that resides in the DRAM, that should be erased when the device is powered off, continues to exist in what is known as a cold-boot attack. By retaining the data in DRAM with low temperatures, a memory dump can be performed that gives the attacker access to all of the data and ultimately the AES encryption/decryption keys. [25] shows that even if an attacker has no information about the original Register Transfer Layer (RTL) design or the synthesis parameters used to create a LUT-based design, it is algorithmically possible to identify AES cores by means of static analysis. They protect against this by configuring the S-Boxes in an AES core after the bitstream is loaded as a partial defense against Trojan insertion via bitstream.

Both [9] and [24] discuss the vulnerability of neighboring long wires on FPGAs. Due to their proximity, information can be extracted from a wire by the wire that neighbors it, which ultimately can lead to leakage of data. Coupling between long wires can be used to recover adjacent information from channels, even if the information is not freely available from the device vendor. A SCA can extract a key from an AES circuit that has been auto-placed and routed by the tools that program the FPGA, such as Vivado. Because the process is automated, typically the developer has no control over where wires are placed and routed and they lack control over the implementation of their design. Ways around this would be ensuring that any IP that contains sensitive information is to be protected and isolated from other parts of the physical layout within the design, however, this is non-trivial to implement. It is difficult for an adversary to locate which wires may contain sensitive data and the neighboring wires that are used to access the leakage of that data. An adversary would have to have extensive knowledge of the design and its layout in order to find wires that can be used to snoop on neighboring wires.

A SCA resistant soft processor was implemented in [15]. When each of the four cores was implemented on a protected softcore they showed improved resistance to 1st order DPA attacks over the unprotected soft core. This comes at a cost though of a two-fold increase in LUTs and slices, and a 22% reduction in throughput which all greatly sacrifice the performance of the FPGA. [17] had similar work where two implementations of a lightweight and high-speed ECC processor was implemented with countermeasures to protect against DPA and SPA. For each of these cases, a lot of work went into the development cycle of processors that ultimately might not be suited for many applications and sacrificed a lot of performance for the sake of security from attacks. The UltraScale+ MPSoC has voltage monitors that can protect against disruption to the FPGA if probes were attached to perform these kinds of attacks. In [19] the authors discussed private circuits II and fault injection attacks on a Spartan 6 FPGA. Particularly, they detect exploitable ciphertext outputs from a LUT within the critical path. They perform overclocking and EM injections to obtain delay faults that were measurable in ChipScope and thus allowed them to view ciphertext.

The work in [27] presents a technique for the detection of timing faults through the use of shadow registers in FPGAs. Shadow registers essentially represent a duplicate in logic which adds a lot of overhead but can yield many security benefits such as the prevention of bit-flipping. Their proposed work is advertised as low-overhead and doesn't disturb the original design through the automated flow for adding the necessary circuitry. This paper isn't particularly concerned about security more so than it is about reliability which is important for the subject matter of this paper.

II. Zynq UltraScale+ MPSoC

A. *Hardware security*

Implementing hardware-based security is of crucial importance to protect the applications and data on the board. Safeguards need to be in place for the possibility of an attacker having physical access to the hardware. Most software protections are rendered useless if an attacker can extract information directly from the hardware. Typically the target for an attacker is the secret keys that are used for encryption and decryption of data on a device. Once this key is solved, the attacker can decrypt all of the information on the board and successfully acquire the data they seek. Xilinx has made an active effort for security by releasing their UltraScale+ MPSoC family which comes with hardware security baked into the FPGAs that can also be used by the developer for their applications.

The Zynq UltraScale+ MPSoC family of FPGAs is the perfect example to demonstrate Xilinx's continued focus on security for consumers. Three versions exist within this family, each designed for specific applications. The CG variant focuses on power performance, solid-state drives, and motion control which makes it ideal for industrial IoT applications. The EG variant which contains specialized processing elements (PEs) that make it ideal for next-generation wired and wireless infrastructure, cloud computing, and aerospace and defense applications. Lastly, the EV variant is aimed at multimedia, surveillance, Advanced Driver Assistance Systems (ADAS), and embedded vision applications. While only the ZCU2014 (EV variant) is available for use in our lab, the board shares the same core security features with the other variants which means there is no need to focus on one variant of the UltraScale+ MPSoC family. The boards are manufactured with a plethora of security features that are embedded within the hardware that the designer can access through the Vivado tools. In fact, within the Zynq UltraScale+ Device technical reference manual, the entirety of Chapter 12 is dedicated to the security features that this family provides [1]. The features can be described by the following four statements: encryption and authentication of configuration files, hardened crypto-accelerators for use by the user application, secure methods of storing cryptographic keys, and methods for detecting and responding to tamper events.

The CSU is at the center of device security and is composed of two main blocks: a Crypto Interface Block (CIB) that contains the AES-GCM, DMA, SHA-3, RSA, and PCAP interfaces, and the Secure Processor Block (SPB) that contains a triple-redundant MicroBlaze processor for controlling boot operations, a PUF, private RAM, and the necessary control/status registers to support all secure operations. The primary responsibilities of the CSU are to handle all the features described above. The CSU can be enabled by going to Advanced Properties and enabling CSU support within the Zynq MPSoC IP within the Vivado tools.

The SPB operates by using a triple-redundant MicroBlaze processor that is not accessible to the user. This unit is responsible for handling the secure boot of the FPGA when requested. It contains its own internal, uninterruptible clock source, dedicated RAM (protected by ECC) and ROM (protected by SHA-3 integrity check), and a PUF for the generation of a device-unique encryption key. The CIB contains the following: an SHA-3/384 hardened core, an AES-GCM hardened core, an RSA exponential multiplier accelerator hardened core, a Processor Configuration Access Port (PCAP), secure key management (BBRAM and eFUSE key storage), and a secure stream for managing data exchanges between cryptographic cores.

The tamper monitoring and response feature is crucial when protecting the physical status of the FPGA. It has various "alarms" that go off when the physical environment around the FPGA is changing. Such alarms are activated by temperature and/or voltage changes reaching their min/max allowed thresholds, if health checks of the PL fail, and if an external activity is detected on the PSJTAG interface pins or other I/O ports. Once any of the tamper signals are triggered, the CSU is notified and can respond in an appropriate way. Such responses are a lockdown of the board and system and zeroization of keys and bitstream, such as in Battery-backed RAM (BBRAM). These events can also be simulated by the designer so that they can ensure it functions properly in the event of the FPGA being tampered with.

The PUF in the UltraScale+ MPSoC acts as a digital fingerprint that can uniquely identify that specific board. Typically, PUFs are based on unique physical variations that occur during manufacturing and are used to identify an individual IC. The goal of the PUF is to use the random variations to generate a key that is tied to the physical hardware it is attached to. Challenge-response questions are sent in pairs to the PUF which gives a response if valid for that device. Each PUF has a unique and unpredictable way of handling these challenge-response pairs which make them great for providing secret key generation and storage as well as a source of true randomness (compared to pseudo-randomness). The Ring-Oscillator (RO) PUF structure relies on delay loops and counters and is a circuit that is composed of an odd number of NOT logic gates in a ring. This PUF essentially operates by timing the delay it takes for all the gates to be traversed and if they are given the correct voltage they will respond with an output of true. Using this technique a unique device key can be generated that will identify the device. [18] presents a method of using 14 ROs on a Spartan-3E FPGA to detect hardware trojans due to the outsourcing of IC manufacturing and design. The aim was to detect the impact of parasitic electrical activity that would be caused by a hardware trojan, but the result was that this was an inadequate choice for a detection method. This was certainly an interesting way to use an RO-based PUF, but not the reason for their use on the UltraScale+ MPSoC. A concern of ROs is that they change with age, which is what the work in [21] addresses due to the limited amount of research on the issue. The aim of their work was to determine the effects of aging on an RO-based PUF so that better ECCs could be used to retain its longevity. They found that under high frequencies, used to model accelerated aging, that four months' time was equal to the average use over 31 years and that the reliability of the PUF dropped to 96%. Lastly, the work in [34] showed that decapsulation, an invasive attack that is typically used for ICs, of a Spartan XC3S200 does not significantly influence the frequencies of the implemented ROs. By using an EM SCA, they were, however, able to extract RO frequencies that can be used to compute correct responses to a given challenge. It is suggested that RO PUFs are vulnerable to SCAs if they use overlapping and sequential RO measurements.

B. Software security

One of the features that are available to the developer in terms of software security is the secure boot of the FPGA. This is a process where the loaded OS boot up images and code are authenticated by the hardware before they initiate the startup sequence. This feature is used so that the hardware only runs on software that is deemed trusted by the developer. The UltraScale+

MPSoC family comes equipped with two methods of secure boot that are available to the developer to ensure the security of the board. The first method is the hardware root of trust which uses asymmetric authentication with optional encryption to provide confidentiality, integrity, and availability (CIA) of the boot sequence and the configuration files. Maintaining CIA is the primary focus of information security. The second method is the encrypt only boot which does not utilize asymmetric authentication but requires that all configuration files loaded on the FPGA must be encrypted and authenticated used AES-GCM. [33] shows that this process is not an atomic operation on the Zynq-7000 and they demonstrate a way to break the secure boot process. They propose a countermeasure that is implemented by creating a flexible and lightweight security-enhanced wrapper for cores with an AXI interface. The wrapper checks the addresses requested over AXI from the underlying core and it attempts to access memory outside of the allowed range, an alarm signal is sent. This prevents the core from modifying configuration settings within the secure boot process.

Another option to secure the software on the FPGA is to encrypt and authenticate the bitstream. A bitstream can be protected with 256-bit AES encryption and HMAC/SHA-256 to prevent unauthorized copying of the design. Bitstreams are always first loaded from Flash into DDR, for systems that have DDR, regardless of security settings [1]. In DDR-less systems, the bitstream remains in the Flash for the authentication, decryption and loading steps. When the bitstream is authenticated, it is first validated using the RSA algorithm. If authentication passes, it is then sent from its source location (DDR or Flash) to the PCAP or to the AES decryptor and then onto PCAP. Vivado has a similar method for encrypting custom IP that is developed in Verilog, SystemVerilog, and VHDL source files. IP blocks can be encrypted so that they act as a black box where data goes in and the results come out, but the code and functions inside the IP are locked down so that theft can be deterred [35]. Different levels of access rights exist for this encryption so that compliant third-party tools can read the source files, so it is up to the developer to set these access rights that are dependent on the level of security they need. However, IP encryption does not prevent the connectivity tracing within Vivado that appears in the netlist. Net names can be traced through the encrypted file so it's best to obfuscate them [4]; this can be done by making I/O and register variable names unrelated to the purpose they serve (I.e.: input [31:0] image_data → input [31:0] i). It is worth noting that both of these methods can be used on other FPGA families and are some of the only security features they have to offer. The UltraScale+ MPSoC family has the benefit of using these features along with the extra security features that are implemented primarily within the CSU. [22] proposes a complete hardware Application Programming Interface (API) for authenticated ciphers that can be used as a tool for developers, however, it is an open-source project that could be subject to compromise.

For the developers to implement some of the security features that the UltraScale+ MPSoC family has to offer, such as the cryptographic cores, they need to use the XilSecure library in the Vivado SDK [2]. This library provides the user with the ability to use the SHA-3/384 engine for 284-bit hash calculation, AES engine for symmetric key encryption and decryption, RSA engine for signature generation, signature verification, and lastly general encryption and decryption. All of these features are contained within the CSU, but in order for developers to use them for their applications, they must use the XilSecure library and proper function calls. These cores within the CSU grant the developer the ability to encrypt and decrypt data within their applications for an extra layer of security. The XilSKey library provides the APIs for programming and reading eFUSE bits and for programming the battery-backed RAM (BBRAM) [2]. The BBRAM is a module that is composed of a static RAM array and is available for storing the devices AES key with its own power supply so the key is not lost if the FPGA loses power. Within the Processing System (PS) the eFUSE holds the AES key, the user fuses, PPK0 and PPk1 hash, SPK ID and some user feature bits which can be used to enable or disable some features. FPGAs are suitable for implementing cryptographic algorithms because there are a lot of bit-level operations, such as shifting and permutation. With the growing logic density and performance of FPGA devices and development tools, it is now feasible to implement a cryptographic system on a single FPGA chip. However, such designs have to be partitioned in a way that isolated subsystems do not leak information to each other. For example, strong isolation is expected to segregate plain text (red text) and ciphertext (black text) [5].

Because the UltraScale+ MPSoC FPGA family, along with the Zynq-7000 series, contains ARM processors, they have an added feature that is known as the Arm TrustZone [3]. This architecture grants developers the ability to “carve-out” a hardware subset of the full SoC. This works by defining processors, peripherals, memory address and even areas of the L2 cache to run as secure or nonsecure hardware. This splits the FPGA into two sections secure world and a non-secure world, with the addition of the monitor mode which handles the transition from one world to another [10]. This allows trusted applications to be run on the secure side with protected memory regions and additional security features [14] while delegating untrusted or unverified IP to the nonsecure side so that they don’t compromise the entire system. This is a useful feature for a developer to take advantage of if they are not confident with IP in their design, however, this should be implemented conservatively because an entire system typically cannot be contained within the secure side. The work in [14] proposes a Dynamic Information Flow Tracking (DIFT) for ARM processors on FPGAs that monitors the data flow of the program to make sure that no unauthorized operation is done in the current branch of execution. [20] implements a feature similar to the ARM TrustZone by inserting a reconfigurable hardware isolation layer between the network link and the target to detect and eliminate Internet Relay Chat (IRC) protocol botnets.

The sophistication of the SoC family of FPGAs from Xilinx enables developers to load a Linux based kernel onto the FPGA, known as PetaLinux. PetaLinux is a simple tool that will help set up the Linux kernel such that it can see all of the special FPGA hardware and be able to interact with it in a Linux based software environment [36]. It consists of three main parts: the pre-configured binary bootable images, customizable Linux, and the PetaLinux SDK which includes tools to assist with configuration, build, and deployment of the software on the FPGA. This is used to give the developers more flexibility in how they interact, create, and modify applications on the FPGA. The Linux kernel it uses is known to be open-source which causes security concerns for those using it. A solution is to replace this customizable kernel with Security-enhanced Linux (SELinux). This modified kernel was originally developed by the NSA as far back as 2000. Its intent was to create a kernel that supports access control security policies. SElinux essentially is a modification of the Linux kernel such that there are more strict security measures in the Linux system, primarily it allows developers to implement strict access control. It sets the FPGA up in such a way that programs running in Linux are under much more rigorous rules than just basic read/write/execute privileges. The reason it is valued is that the kernel is a basic set of programs that run and manage the machine and connect all parts from the machine, whether it be a couple of CPUs, a hard drive, or a special memory that is created as an IP in the FPGA. With SELinux installed on an FPGA, it provides tighter control on the way programs and IP are accessed and modified which allows developers to add more assurances on the data and ultimately the application. While SELinux provides a number of security benefits for developers, it should not be viewed as their silver bullet. Other security practices should be put into place to help lock down the loaded OS such as firewall policies, which are tough to implement on an FPGA. [23] provides work that implements the mandatory access control (MAC) security architecture on FPGA-based heterogeneous systems. Their work explores the design and implementation of a security framework for controlled haring of FPGA hardware modules in a MAC-based OS/Hypervisor environments. This means that the software security policies will propagate down to the hardware level. This work is prototyped with SELinux and its utility is demonstrated by evaluating the trade-offs between security performance and execution overhead.

Another software method for developing secure designs is to formally verify the design so that it performs only the functions it is meant to and nothing else. This is crucial for assuring that a system has no extra unintended functionality, features, or bugs that can be exploited. Software designs will often contain “hooks” that are intended for future enhancement or possible optimization, but they can be used to introduce unintended functionalities that could be used in a malicious way. Formal Verification (FV) is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics. Properties that can be specified in FV are functional properties, timing properties, structure properties, fault-tolerance properties, and equivalence at various design stages [5]. Commonly used FV techniques include equivalence checking, model checking, symbolic model checking, and automated theorem proving [5]. Equivalence checking is routinely applied in ASIC development. One common use is to prove that the register-transfer-level (RTL) design and its corresponding synthesis netlist implement exactly the same functionality. Formal verification can be applied to the development cycle of IP within a design that is made in-house so that a level of assurance is added to the design, as shown in Figure 1. FV is used because standard simulation techniques of a design are not enough to assure the absence of bugs. Simulations potentially identify the presence of a bug but do not ensure the absence of a bug. FV exhaustively explores all state space to uncover incorrect behaviors and identifies enough properties to check [5]. Typically when a 3rd party IP is used within a design, it comes with a testbench so it can be simulated to show that it performs the required functions. However, as just described, this does not mean that the 3rd party IP is absent of bugs or malicious code. Formally verifying 3rd party IP can be a challenging and even impossible task because the IP can contain extensive amounts of code that the user does not understand or have knowledge of, or the design can be locked or encrypted by the 3rd party so the code cannot be accessed or viewed by the user. Even if the acquired 3rd party IP was formally verified prior to delivery and the results of that verification were provided to the user, they still don't have the means to verify themselves that the process was done in a manner that assures the code does not contain malicious functions or bugs; for all they know malicious function could have been added to the 3rd party IP and formally verified but not presented. This makes FV a difficult area to implement when designing secure, complex, and high-assurance systems.

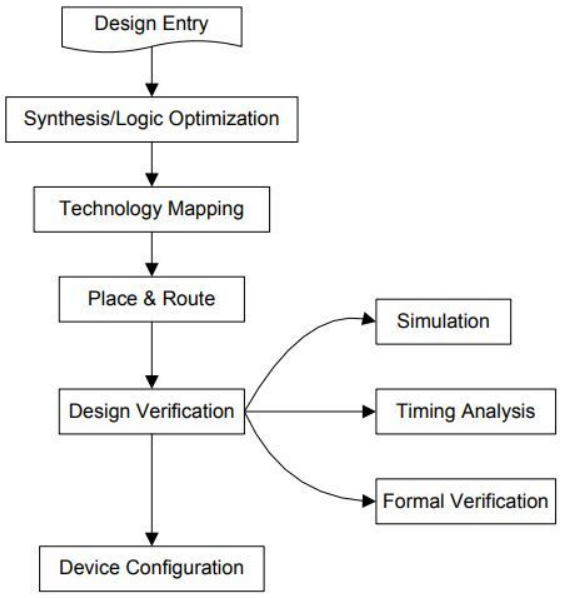

Figure 1: A typical FPGA-based design flow [5].

The Higher-Order Logical (HOL) interactive theorem prover is one way to approach FV. It uses complex mathematical theorems to prove the correctness of functionality within a design. Another option is ReWire, which is a purely functional open-source programming language for designing, implementing, and formally verifying secure hardware designs for FPGAs. ReWire is a language for high-level synthesis based on the functional language Haskell. Functional languages are a commonly proposed approach to alleviating the well-known FPGA programmability problems. One of the benefits of ReWire is that programs can generate synthesizable VHDL. Cadence provides a variety of tools to implement verification within systems and Synopsys provides its VC Formal software as well. Those two are perhaps the most "main-stream" methods that are available to designers, but FV doesn't scale well so options are slim. FV requires exact parameters and definitions of designs so it can go about proving all possibilities that the functions can generate. Because designs and functions vary extensively on a case-by-case basis, it is difficult to develop a single program to handle all FV. [29] presents a combination of access monitors and formal runtime verification within FPGAs. They implemented the first known prototype that realizes proof-carrying hardware on real FPGAs by embedding a virtual FPGA overlay into a ReconOS/Zynq system.

III. Recommended Security Practices

The previous sections have presented the security weaknesses on FPGAs, such as the types of attacks that FPGAs are susceptible to, and the security features that are present on the Zynq UltraScale+ MPSoC which has security measures built-in to the hardware. Now, this section will discuss a recommended cycle of development to ensure a secure deployment and operation of an FPGA-based design. Aside from the development cycle, hypothetical scenarios will be described that could ultimately pose risk for FPGA operations based on a variety of assumptions such as board location, physical access, remote access, and internet connectivity. It is important to mention here again that it is assumed the FPGA received from the vendor is deemed trusted. This is because the vendor takes measures to prevent malicious or counterfeit hardware and it is too difficult to verify individual components on the FPGA.

A. Development Cycle

The development cycle is perhaps the most crucial step in securing a design. The methods used during this process will determine the security of a system for as long as it is used in the field before it is replaced and/or reprogrammed. A layered approach is required to secure the FPGA from the hardware, up to the design, and lastly the data within the design. The first step is to analyze the environment in which the system is being developed. One of the most overlooked aspects of this step is that the design is usually implemented on a PC that has internet access. It's easy to understand why as the internet provides a tremendous amount of resources to assist in code development or even acquiring 3rd party IP. However, if the PC is compromised through an attack via the network it is on then the design itself can be compromised. Attackers can remotely access PCs and modify a design by flipping a bit in the bitstream or inserting a trojan into the design. By using a computer off of the main network, the design is much safer from adversaries. It is recommended to design, test, and load a design onto an FPGA via a PC that is not internet-connected. If data from the internet is needed on that PC, then loading simple designs or code onto a flash drive and manual transferring them over to the non-connected PC would be preferable to having the computer be compromised where it is harder to track which files were manipulated.

When scanning the physical environment where the design will be built, it is important to understand who has access to the area as well as the computer where development will occur.

Development should preferably be done in a room that has requires the scan of credentials to enter. This will allow for the tracking of people who enter the room as well as timestamps on when they entered and exited the area. Credential access to the room should be limited to only people with the understanding of the systems the rooms contain. For example, a web designer should not be able to access an embedded system room because they should have no use for the systems in that area. Minimizing who has access to an area to those of knowledge and trust is of crucial importance. The computer where development will occur should be password protected and only accessible by one user. This prevents unauthorized modifications or design choices that the primary user is unaware of. Lastly, the room should ideally have video surveillance that runs at all times to monitor who access the room, the systems and computers, and if monitors have unauthorized access from remote adversaries. Saving this data will allow building security to trace back who was responsible for a potential attack or mistake made during the development cycle. These steps should prevent unauthorized users from tampering with the development cycle of an FPGAs design.

Starting with IP, when the actual design is under development all 3rd party IP should undergo thorough testing with the provided testbench to ensure that the functions perform as expected. If possible, any 3rd party IP should be formally verified as well, but for reasons stated earlier this can be a difficult step to achieve. IP developed in-house should undergo FV so that design behaves as intended and adversaries can’t take advantage of any unintended functionality. Individual IP block encryption is needed to prevent theft of the design and the code that it is composed of. Within the IP, extra steps can be taken to add security such as variable obfuscation, redundant code, watermarking, and extra processes that can throw off SCAs. Lastly, the bitstream should be encrypted and paired with onboard authentication to ensure that the legitimate bitstream is used at all times. These steps will ensure that the development of the hardware-based design is secure and only accessible by the developer.

When configuring the UltraScale+ MPSoC FPGA, the CSU should be activated and embedded within the design to enable the added security features. Anti-tampering can be added to the design to trigger alarms when the temperature or voltage of the FPGAs environment fluctuates beyond a defined threshold. The anti-tampering measure also ensures that the developer can define which physical connection ports on the FPGA are eligible to be accessed during normal operations. If the FPGA is not meant to interact with any of the physical ports then the developer can decide what action will be taken if a port is attempting to be accessed. The CSU also grants the developer access to the crypto cores when developing the software that will run on the processors so that data can be encrypted and authenticated with protocols like AES. The CSU will handle the secure boot of the FPGA if it is enabled, which it should be. When developing the software design that runs on the FPGAs processors, the secure libraries should be used to provide encryption to the application. These libraries can also help define where and how secret keys are stored on the FPGA and if select keys should be erased if tampering is detected. If a Linux-based operating system is intended to run on the FPGA then

SELinux should be used. This grants extra MAC security features that the programmer can define within the application. Access rights and rules can be put in place so that only the properly authorized functions can access other IP or parts of memory on the FPGA.

B. Deployment

When the FPGA is ready to perform its duties and is set to be deployed into a real scenario, it is important to consider the environment where it will exist. One of the most common traits between all attacks to FPGA systems is that the adversaries need physical access to the hardware in order to carry out their malicious intent. The FPGA should be deployed in an environment that is protected with credential access and surveillance, at the very least behind a locked door. Because of their flexibility and capability, this isn't always a realistic option, such as in avionics. However, warehouse and data center deployments can certainly be safeguarded with this procedure. Routine checkups on the FPGA in operation should be performed so that nominal behavior can be detected. If any malicious behavior is reported on the FPGA then the bitstream should be analyzed and the hardware should be reprogrammed and redeployed. This area is tricky because of the many uses for FPGAs so this should be considered when deemed appropriate. For IoT related infrastructure or any interconnected FPGA, security becomes a concern in terms of remote access. Just as an adversary can compromise a PC from across the globe, they certainly could lock on to the FPGAs' IP address if it is connected to the internet. Firewall systems and access control are crucial for these systems so that unauthorized access is prevented. Unfortunately, not a lot of research is published in the area of remote access of FPGAs so this is another tough area of assurance. The developer can, however, enable data transmissions that will report FPGA data to the user remotely to ensure it is behaving as expected if it is connected to the internet. All of this to say is that when possible the FPGA should be deployed in a secured area so that the risks of tampering and attacks are minimized.

C. CSU Capability

One of the best security features that the CSU enables for a developer is the anti-tampering mode. This feature can disable ports and trigger alarms on the FPGA that can detect when a port is attempting to be accessed. This prevents a variety of SCAs because it disables adversaries from being able to access ports like the JTAG port that would otherwise allow a readback attack where the entire bitstream can be obtained. Bitstream spoofing is also prevented with standard bitstream encryption and authentication which is a method available on other FPGAs, but reinforced with the limited access to physical ports on the hardware with the anti-tampering feature. Data remanence attacks that require the FPGA to be introduced to temperatures as low as -20 C can also be prevented because the board has temperature readings that can zero out keys and the bitstream is the external temperature fluctuates beyond a defined threshold. FIAs, like local heating and voltage glitching, can also be prevented with voltage alarms that are triggered if a voltage on the FPGA goes beyond a defined threshold. The Arm

TrustZone feature can prevent 3rd party IP from accessing IP or memory outside of defined bounds which ensures that it won't compromise the integrity of the application.

The hardened crypto cores that are within the CSU also give the developer an edge with enabling them to encrypt and decrypt data within the application. This can protect from SCAs where even if the information is leaked from the board, a required key will be needed to decrypt the data which renders their attack useless. By applying encryption to the IPs in the design, the bitstream, and to the data that is used in the application, the FPGA is much more secure from the theft of data and the design. This also negates the vulnerability f neighboring long wires where data can be leaked through wires that run next to wires used by the application. By enabling the secure boot feature, the FPGA will authenticate the bitstream and programs that are loaded onto the board in a secure fashion with a triple-redundant Microblaze soft processor and independent memories and clock so that the boot process isn't compromised by the rest of the design that may already be loaded onto the board or in the current bitstream. Secure key storage, like eFUSEs and BBRAM, is another huge feature that enables the secret keys that protect encryption in the FPGA to be placed in secure memory with the option to have some of the erased and zeroed out if an alarm is triggered. The developer also has control over these keys by using the secure libraries in the Vivado SDK.

While the CSU in the UltraScale+ MPSoC provides a great number of security benefits that can prevent attacks, it isn't perfect and methods still exist that can compromise the FGPA. Power attacks, like DPA and SPA, are still a valid attack against this family of FPGAs because the adversary only needs to be able to take measurements of the current in the board to be able to orchestrate an attack. It is impossible for the FPGA to know if physical wires are being listened to and recorded to launch an attack on a specific component that could uncover a secret key. EM attacks are also another vulnerability that FPGAs are susceptible to because the tools to carry out an EM attack do not physically interact with the FPGA, they sit above components and take readings that could be used to break secret keys. These attacks can be prevented within an IP design that adds redundancies and extra processes that can thwart the attempt to gain valid data measurements from the FPGA. HTTs are not discussed in great detail in this paper because the FPGA is assumed to be trusted when the developer has taken delivery of the hardware from the vendor. However, HTTs are still a problem for systems because if a malicious component is physically attached to the board it can compromise a design or leak data to an adversary and the user has no way to prevent it from happening or removing the component, much less detected which component is at fault. Methods like flash glitching are hard to prevent because the attack is external to the board and causes a mass wave of glitches that can compromise the application and leak data. FPGAs are physical devices that are subject to the same vulnerabilities as all circuits are and glitches can't be prevented within the design or within the software, unfortunately. With these FPGA vulnerabilities listed, it is important to ensure that the FPGA is operating in a safe environment where it will be free from these sorts of compromises that are launched from an adversary.

IV. Conclusion

The goal of this work was to introduce the readers to the potential attacks that FPGAs are susceptible to and how the UltraScale+ MPSoC FPGA, and its many security features, is the optimal choice when designing secure systems. This paper has presented a variety of attacks, some that FPGAs are vulnerable to, some that apply to ASICs, as well as the security capabilities of the Xilinx Zynq UltraScale+ MPSoC FPGA. A general overview is provided that describes the many features that the CSU brings to the UltraScale+ MPSoC FPGA and how it offers greater security when compared to other Xilinx FPGA families. Hardware attacks were listed to give the reader an understanding of the types of threats that exist to embedded systems to provide them the ways in which these attacks are carried out to better prevent them. Lastly, a section dedicated to recommended development practices was given so that the proper steps can be taken to ensure the safety of an FPGA that is running an application in the field. While many actions can be taken to add security to a system, only a select few will actually apply to the application depending on the circumstances. A lot of security protocols that can be added to IP within a system come at the cost of overhead that would lower the quality of the application. Most attacks that FPGAs are vulnerable to require physical access to the hardware to take measurements and launch an attack on the application or the design that is stored in the hardware. Above all, it is most important to ensure that, depending on the application and the sensitivity of its data, the physical hardware is in a trusted environment where adversaries can't walk up to the hardware and tamper with it for extended periods of time.

The work in this paper discusses the many features that the CSU provides to the UltraScale+ MPSoC family of FPGAs, however, it should be mentioned that this paper doesn't go into extensive detail on each of the features. The measures put in place by the CSU are extensive and have hundreds of pages in the technical reference manuals to adequately describe the details of the security features, which this paper does not have the time to dive in to. It is strongly encouraged for anyone attempting to develop a secure system with these features to read the reference manuals to gain a full understanding of the systems they are working with. This paper should be used to present all of the tools that are available with this family of FPGA that the developer can leverage in their design. A large number of previous works are presented to demonstrate the feasibility of such attacks to FPGAs and how they work. Lastly, this paper showcases the features of the CSU to describe how they can prevent a variety of attacks on FPGAs that are described in the previous works.

V. Acknowledgments

The work in this paper was funded and performed under a contract from our sponsor Dennis Stanley at Honeywell Federal Manufacturing & Technologies. I would like to extend my gratitude to him for the opportunity for me and my colleagues at the University of Arkansas.

VI. References

1. *Zynq UltraScale+ Device Technical Reference Manual*. Xilinx, 26 July 2019, www.xilinx.com/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf.
2. *Xilinx Standalone Library Documentation OS and Libraries Document Collection*. Xilinx, 5 Dec. 2018, www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/oslib_rm.pdf.
3. *Programming ARM TrustZone Architecture on the Xilinx Zynq-7000 All Programmable SoC User Guide*. Xilinx, 6 May 2014, www.xilinx.com/support/documentation/user_guides/ug1019-zynq-trustzone.pdf.
4. *Vivado Design Suite User Guide Creating and Packaging Custom IP*. Xilinx, 7 June 2017, www.xilinx.com/support/documentation/sw_manuals/xilinx2017_2/ug1118-vivado-creating-packaging-custom-ip.pdf.
5. Hu, Yalin. "Exploring Formal Verification Methodology for FPGA-Based Digital Systems ." *Semanticscholar.org*, Sandia National Laboratories, Sept. 2012, pdfs.semanticscholar.org/0a71/0fd0139f9a79d9ebb8856818ae9f508c484e.pdf.
6. Trimberger, Stephen M., and Jason J. Moore. "FPGA Security: Motivations, Features, and Applications." *FPGA Security: Motivations, Features, and Applications - IEEE Journals & Magazine*, 8 July 2014, ieeexplore.ieee.org/document/6849432.
7. Trimberger, Steve, and Jason Moore. "FPGA Security: From Features to Capabilities to Trusted Systems." *FPGA Security: From Features to Capabilities to Trusted Systems - IEEE Conference Publication*, IEEE, 5 June 2015, ieeexplore.ieee.org/document/6881481.
8. Shila, Devu Manikantan, and Vivek Venugopal. "Design, Implementation and Security Analysis of Hardware Trojan Threats in FPGA." *Design, Implementation and Security Analysis of Hardware Trojan Threats in FPGA - IEEE Conference Publication*, IEEE, 14 June 2014, ieeexplore.ieee.org/document/6883404.
9. Provelengios, George, et al. "Characterization of Long Wire Data Leakage in Deep Submicron FPGAs." *ACM Digital Library*, ACM, 20 Feb. 2019, dl.acm.org/citation.cfm?id=3293923.
10. Chaudhuri, Sumanta. "A Security Vulnerability Analysis of SoCFPGA Architectures." *ACM Digital Library*, ACM, 24 June 2018, dl.acm.org/citation.cfm?id=3195979.
11. Jacob, Nisha, et al. "Compromising FPGA SoCs Using Malicious Hardware Blocks." *Compromising FPGA SoCs Using Malicious Hardware Blocks - IEEE Conference Publication*, IEEE, 31 Mar. 2017, ieeexplore.ieee.org/document/7927157.

12. Druyer, R., et al. “A Survey on Security Features in Modern FPGAs.” *A Survey on Security Features in Modern FPGAs - IEEE Conference Publication*, IEEE, 1 July 2015, ieeexplore.ieee.org/document/7238102.
13. Ziener, Daniel, et al. “Configuration Tampering of BRAM-Based AES Implementations on FPGAs.” *Configuration Tampering of BRAM-Based AES Implementations on FPGAs - IEEE Conference Publication*, IEEE, 5 Dec. 2018, ieeexplore.ieee.org/document/8641692.
14. Wahab, Muhammad Abdul, et al. *A Small and Adaptive Coprocessor for Information Flow Tracking in ARM SoCs - Semantic Scholar*. IEEE, 1 Jan. 1970, ieeexplore.ieee.org/document/8641695.
15. Diehl, William, et al. “Side-Channel Resistant Soft Core Processor for Lightweight Block Ciphers.” *Side-Channel Resistant Soft Core Processor for Lightweight Block Ciphers - IEEE Conference Publication*, IEEE, 6 Dec. 2017, ieeexplore.ieee.org/document/8279819.
16. Kabin, Ievgen, et al. “Horizontal Address-Bit DPA against Montgomery KP Implementation.” *Horizontal Address-Bit DPA against Montgomery KP Implementation - IEEE Conference Publication*, IEEE, 6 Dec. 2017, ieeexplore.ieee.org/document/8279800.
17. Salman, Ahmad, et al. “A Scalable ECC Processor Implementation for High-Speed and Lightweight with Side-Channel Countermeasures.” *A Scalable ECC Processor Implementation for High-Speed and Lightweight with Side-Channel Countermeasures - IEEE Conference Publication*, IEEE, 6 Dec. 2017, ieeexplore.ieee.org/document/8279769/authors#authors.
18. Lecomte, Maxime, et al. “Thoroughly Analyzing the Use of Ring Oscillators for on-Chip Hardware Trojan Detection.” *Thoroughly Analyzing the Use of Ring Oscillators for on-Chip Hardware Trojan Detection - IEEE Conference Publication*, IEEE, 9 Dec. 2015, ieeexplore.ieee.org/document/7393363.
19. Rakotomalala, Henitsoa, et al. “Private Circuits II versus Fault Injection Attacks.” *Private Circuits II versus Fault Injection Attacks - IEEE Conference Publication*, IEEE, 9 Dec. 2015, ieeexplore.ieee.org/document/7393338.
20. Hategekimana, Festus, et al. “Hardware Isolation Technique for IRC-Based Botnets Detection.” *Hardware Isolation Technique for IRC-Based Botnets Detection - IEEE Conference Publication*, IEEE, 9 Dec. 2015, ieeexplore.ieee.org/document/7393319.
21. Gehrer, Stefan, et al. “Aging Effects on Ring-Oscillator-Based Physical Unclonable Functions on FPGAs.” *Aging Effects on Ring-Oscillator-Based Physical Unclonable Functions on FPGAs - IEEE Conference Publication*, IEEE, 9 Dec. 2015, ieeexplore.ieee.org/document/7393289.

22. Homsirikamol, Ekawat, et al. “A Universal Hardware API for Authenticated Ciphers.” *A Universal Hardware API for Authenticated Ciphers - IEEE Conference Publication*, IEEE, 9 Dec. 2015, ieeexplore.ieee.org/document/7393283.
23. Hategekimana, Festus, et al. “Inheriting Software Security Policies within Hardware IP Components.” *Inheriting Software Security Policies within Hardware IP Components - IEEE Conference Publication*, IEEE, Sept. 2018, ieeexplore.ieee.org/document/8457632.
24. Ramesh, Chethan, et al. “FPGA Side Channel Attacks without Physical Access.” *FPGA Side Channel Attacks without Physical Access - IEEE Conference Publication*, IEEE, Sept. 2018, ieeexplore.ieee.org/document/8457631/.
25. Swierczynski, Pawel, et al. “Protecting against Cryptographic Trojans in FPGAs.” *Protecting against Cryptographic Trojans in FPGAs - IEEE Conference Publication*, IEEE, 6 May 2015, ieeexplore.ieee.org/document/7160059.
26. Riebler, Heinrich, et al. “Reconstructing AES Key Schedules from Decayed Memory with FPGAs.” *Reconstructing AES Key Schedules from Decayed Memory with FPGAs - IEEE Conference Publication*, IEEE, 13 May 2014, ieeexplore.ieee.org/abstract/document/6861629.
27. Stott, Edward, et al. “Timing Fault Detection in FPGA-Based Circuits.” *Timing Fault Detection in FPGA-Based Circuits - IEEE Conference Publication*, IEEE, 13 May 2014, ieeexplore.ieee.org/document/6861594.
28. Xu, Song, et al. “IC Security Evaluation against Fault Injection Attack Based on FPGA Emulation.” *IC Security Evaluation against Fault Injection Attack Based on FPGA Emulation - IEEE Conference Publication*, IEEE, 9 Dec. 2016, ieeexplore.ieee.org/document/7929554.
29. Wiersema, Tobias, et al. “Memory Security in Reconfigurable Computers: Combining Formal Verification with Monitoring.” *Memory Security in Reconfigurable Computers: Combining Formal Verification with Monitoring - IEEE Conference Publication*, IEEE, 12 Dec. 2014, ieeexplore.ieee.org/document/7082771.
30. Yang, Jianlei, et al. “Power Supply Noise Aware Evaluation Framework for Side Channel Attacks and Countermeasures.” *Power Supply Noise Aware Evaluation Framework for Side Channel Attacks and Countermeasures - IEEE Conference Publication*, IEEE, 12 Dec. 2014, ieeexplore.ieee.org/document/7082770.
31. Li, He, and Qiang Liu. “Hardware Trojan Detection Acceleration Based on Word-Level Statistical Properties Management.” *Hardware Trojan Detection Acceleration Based on Word-Level Statistical Properties Management - IEEE Conference Publication*, IEEE, 12 Dec. 2014, ieeexplore.ieee.org/document/7082769.
32. Zhang, Jiliang, and Gang Qu. “A Survey on Security and Trust of FPGA-Based Systems.” *A Survey on Security and Trust of FPGA-Based Systems - IEEE Conference Publication*, IEEE, 12 Dec. 2014, ieeexplore.ieee.org/document/7082768/.

33. Jacob, Nisha, et al. *How to Break Secure Boot on FPGA SoCs through Malicious Hardware - Semantic Scholar*. 2017, www.semanticscholar.org/paper/How-to-Break-Secure-Boot-on-FPGA-SoCs-through-Jacob-Heyszl/198300942c9367ae287f3cb8005233606a9dce5c.
34. Merli, Dominik, et al. "Semi-Invasive EM Attack on FPGA RO PUFs and Countermeasures." *ACM Digital Library*, ACM, 14 Oct. 2011, dl.acm.org/citation.cfm?id=2072276.
35. Bhunia, Swarup, et al. "Fundamentals of IP and SoC Security - Design, Verification, and Debug: Swarup Bhunia." *Springer*, Springer International Publishing, 2017, www.springer.com/gp/book/9783319500553.
36. "PetaLinux Tools Documentation." *Https://Www.xilinx.com/*, Xilinx, 22 May 2019, www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug1144-petalinux-tools-reference-guide.pdf.