Threats and Challenges in Reconfigurable Hardware Security

Ryan Kastner
Dept. of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093
kastner@cs.ucsd.edu

Ted Huffmire
Department of Computer Science
Naval Postgraduate School
Monterey, CA 93940
tdhuffmi@nps.edu

Abstract

Computing systems designed using reconfigurable hardware are now used in many sensitive applications, where security is of utmost importance. Unfortunately, a strong notion of security is not currently present in FPGA hardware and software design flows. In the following, we discuss the security implications of using reconfigurable hardware in sensitive applications, and outline problems, attacks, solutions and topics for future research.

1 Introduction

Reconfigurable hardware is increasingly being used in sensitive applications. Examples include our national infrastructures (power grids, network routers, satellites), transportation (planes, trains, automobiles), military equipment (weapons, radar, software defined radio) and medical devices. In each case, failure has serious consequences, and security is of utmost importance. Furthermore, FPGAs[1] are found in consumer electronics where we store personal information ranging from the mundane (phone numbers, calendar) to more personal (email, voice mail, financial) information.

There are many potential ways to exploit a computing device. These include both hardware and software attacks, that can be done in a physical or logical manner. These attacks can steal confidential information, modify the system to perform devious, unintended activities, perform denial of service, or even destroy the system. As reconfigurable hardware continues to become more powerful and cheaper, it becomes more attractive to designers as well as more susceptible to attackers, who will attempt to exploit any security weakness. Unfortunately, the security of reconfigurable hardware has, until recently, largely been ignored.

In this paper, we discuss potential attacks that can be done on reconfigurable hardware. These include modifying the hardware, observing sensitive information through physical side channels, adding unintended functionality through the design tools, and stealing intellectual property. We attempt to outline these and other potential attacks and provide a survey of solutions to combat them. We also mention avenues for future research.

The paper is organized as follows. The next section lays out three stages in the life cycle of an FPGA – manufacturing, application development and deployment. Then Section 3 describes some security problems found during these various stages. In each case, we first identify the problem, provide some potential attacks, survey solutions and give ideas for future research. We conclude in Section 4.

2 Reconfigurable Hardware Life Cycle

Before we discuss specific security problems, attacks and solutions, we describe the lifecycle of reconfigurable hardware. We divide the lifetime into three stages – manufacturing, application development and deployment (see Figure 1). We use these three stages to provide a coarse

[1] We use the terms reconfigurable hardware and FPGA interchangeably throughout the article.

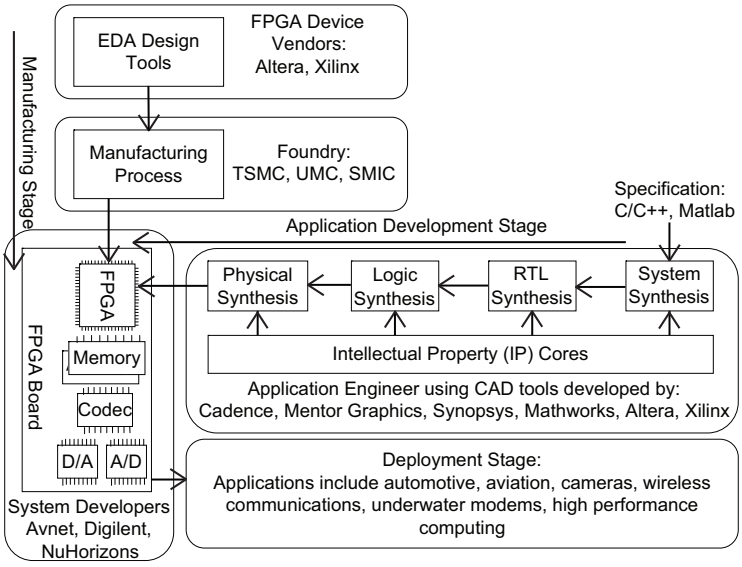

Figure 1: The life cycle of reconfigurable hardware, from manufacturing to application development to deployment. The hardware is handled by a number of different sources using a wide range of design tools. Every stage of the life cycle contains security threats that must be collectively addressed to insure the safe usage of the hardware.

grain grouping of the security issues surrounding the reconfigurable device.

2.1 Manufacturing Stage

Reconfigurable hardware is designed and manufactured by the FPGA vendors – primarily Altera and Xilinx who are the market leaders. New architectures are developed approximately every 12 to 18 months. The exact details of each architecture are patented and/or trade secrets, in order to maintain a competitive advantage. It is possible to reverse engineer them and get the specifics; however this is a difficult task requiring sophisticated invasive physical attacks. We discuss such attacks in Section 3.2.

The FPGA companies are fabless, so they rely on a third party foundry, usually located in Asia, to manufacture the physical devices. These devices are then either sold directly to the system developers who create a final product, or to a third party partner company (e.g. Digilent, Avnet, NuHorizons) who create a development board with the FPGA and a set of peripherals. These boards are purchased by another entity, who customize to fit their end product. The boards are usually targeted towards users in a specific industry such as medical, image processing, wireless, high performance computing, and so on. A development board contains an FPGA along with memory, video and audio interfaces, radios, testing interface, codecs, analog to digital (A/D) and digital to analog (D/A) converters, etc., depending on the targeted application space.

2.2 Application Development Stage

At this point, the life cycle moves into the application development stage. It is during this stage where the FPGA is integrated into the final system, and is programmed to implement its intended functionality. The designer uses either the FPGA chip and builds their own platform to meet their intended application, or uses a third party FPGA development board.

The application development process involves the use of a variety of CAD tools typically from a number of dif-

ferent sources. These tools include electronic system design (ESL) tools which take a high level language (C/C++, MATLAB, SystemC) and translate it to a register transfer level (RTL) hardware description language (HDL). Examples of such tools include Xilinx AccelDSP, Mathworks HDL Coder, both of which take MATLAB and translate it to HDL. Other tools include Synphora's Pico, Forte's Cynthesizer, Mentor Graphics' CatapultC and ImpulseC that transform variants of C/C++ into HDL. The RTL is then synthesized into a logical netlist using any number of logic synthesis tools from EDA companies like Cadence, Mentor Graphics, Synopsys, and Magma. Finally, physical synthesis tools transform the logic netlist into a bitstream which is used to program the FPGA. The FPGA vendors provide physical synthesis tools, as do several of the aforementioned EDA companies.

The different tools throughout the application design stage may use Intellectual Property (IP) cores. The ESL design tools often include customized high level functions such as microcontrollers (Neos or Microblaze), signal processing cores (filters, linear transforms, audio and video codecs), compression, encryption cores, etc. These cores may come from different sources, each of which could have different levels of trust. For instance, we may receive a core from one of the tool vendors or use some open source core from an online repository like Opencores [18]. Each of these requires that you trust the maker of the core. These IP cores may be distributed in a variety of forms across all the different tools. For instance, the microcontroller IP core can be specified at the RTL, logic or physical netlist or even as a bitstream. Distribution in a form later on this list is often preferred by the vendors as it is harder to reverse engineer a core specified as a bitstream as opposed to one described in RTL.

2.3 Deployment Stage

The final stage in the lifetime of the reconfigurable hardware is when the chip is deployed into the environment. This obviously depends on the application, and FPGAs have a wide range of usage. These could be cars, planes, wireless routers, phones, the Mars Rover, satellites, weapons systems, cameras, underwater modems, and so on. The vulnerability of the device in the deployment stage is heavily dependent on the application. It involves the physical security of the device (i.e. how easily accessible and protected is it from a physical handling) as well as the functionality of the device (e.g. military devices contain more valuable designs, and therefore are more vulnerable to attack than a student project).

2.4 Securing the Life Cycle

Attackers will target the weakest link and exploit the easiest flaws found anywhere in the life cycle. Therefore, when addressing security concerns, we must understand the lifecycle of the device - from cradle to the grave (and beyond). The life cycle includes the environment, development tools, and testing equipment, among other things, that will be used on the device. Any specific information about how the device is used can and should be considered during every security decision; such additional information can only make the resulting system more secure.

A trusted system relies on a management plan and procedures to ensure the security of the system. This management plan defines the tools and procedures that must be used to develop and test the system. This often includes formal methods to ensure the system meets the required specifications, and vulnerability assessment to analyze the system's robustness against exploitation. Procedures are developed to ensure the safe delivery of the system to its users, and provide the users appropriate guidance and documentation so that they can configure and run the system without introducing vulnerabilities.

The Common Criteria (CC) [10] is a system assurance framework that provides guidance to developers and confidence to customers and consumers of the system. The CC describes ten types of assessment required to determine if a system is trustworthy:

1. Analysis and checking of processes and procedures;
2. Checks that processes and procedures are applied;
3. Analysis of the correspondence between design representations;
4. Analysis of the design representation against the requirements;
5. Verification of proofs;
6. Analysis of guidance documents;

7. Analysis of the functional tests developed and the results provided;
8. Independent functional testing;
9. Vulnerability analysis;
10. Penetration testing.

While these assessments are not specifically targeted towards systems with reconfigurable hardware, they are certainly applicable. For example, the bitstream is a program analogous to machine code for microprocessors. Therefore, these assurances should apply to the application development design flow. The hardware manufacturing stage is also highly dependent on software (EDA tools), so it also at least partially relates to these assessments. This begs the question: in what ways is the FPGA life cycle different from the software life cycle? We aim to answer that question, and many others, in the following section. We do this by listing a number of security problems and attacks found during the life cycle of a reconfigurable device, and present potential solutions to mitigate these security flaws.

3 Reconfigurable Hardware Security Problems, Attacks and Solutions

In this section, we discuss some of the security problems found in various stages of the reconfigurable hardware life cycle. In each case, we describe the problem, list the attacks, provide some solutions and suggest potential topics for future research.

3.1 Trusted Hardware

3.1.1 Problem Statement

Due to the immense costs associated with fabricating chips in the latest technology node, the design, manufacturing and testing of integrated circuits has moved across a number of companies residing in various countries. This makes it difficult to secure the supply chain and opens up a number of possibilities for security threats. The problem boils down the the question: how can these fabless companies assure that the chip received from the foundry is exactly the same as the design given to the foundry – nothing more, nothing less?

3.1.2 Attacks

A *Trojan horse* is a malicious piece of hardware inserted into the integrated circuit that appears to perform a specific (normal) action, but actually performs some unintended (potentially malicious) action. E.g., additional logic that blocks access to some resource (memory, I/O pins), enables access to some restricted data, or performs wrong functionality may be added to the integrated circuit during manufacturing. One way to combat this is through testing. However, the Trojan horse could be triggered by a special code that is applied when the device is in the field, i.e., it is activated after a specific input is received. A well-informed attacker who knows the intimate details of the device could pick an extremely uncommonly occurring input. Using this input would make it quite rare that this would be triggered during normal testing. The input could be triggered using a variety of physical attacks. One invasive trigger would be to directly access the I/O pins to perform the action. Other less invasive, but much more difficult triggers (perhaps bordering on science fiction?) include using electromagnetic radiation or thermal energy to activate the Trojan horse.

A *kill switch* is the malicious manipulation of the hardware itself and/or the software that runs on the chip, that once activated, renders the chip inoperable. The kill switch could be performed through thinning certain crucial wires, so that electromigration eventually eliminates part of a wire, creating an open connection.

A *backdoor* is a form of Trojan horse where functionality is added to the circuit that enables access to the system, e.g., to disable or enable functionality. An example of this is to selectively disable the encryption core. This could be done without the user's knowledge and thus is harder to detect when the attack is occurring when compared to the kill switch, which totally disables the entire chip.

These attacks may seem unlikely, however there are many documented cases of maliciously modified hardware. This goes all the way back to the Cold War when US and Russian security agencies used software and hardware attacks to spy on each other. US agents sabotaged oil pipeline control software and allowed it to be stolen

by Russian spies. Not to be outdone, the Russians intercepted the delivery of typewriters heading to the US, and hacked them to add a keylogger [16]. More recently, there are theories that a European chip maker developed a kill switch into a microprocessor, and this was utilized to disable Syrian radars from detecting an Israeli attack [1]. This conspiracy theory is grounded in reality as hardware Trojans have been developed. For example, King et al. [13] show how to add a small amount of logic to the Leon processor to enable a number of attacks, and Agrawal et al. [2] describe how to add 400 logic gates to a public key encryption circuit which enables them to leak the key.

3.1.3 Solutions

The Department of Defense and the National Security Agency (NSA) started the Trusted Foundries Program as a method to ensure a secure supply chain for chips used by the sensitive government products. This crisis was detailed in a Defense Science Board Report (DSB) [4] and was cited earlier in a white paper by US Senator Lieberman [16]. The DSB report concludes with a recommendation to Congress to make the supply of electronics a national priority. A trusted foundry is certified as a trusted environment capable of producing leading edge microelectronics parts, and several foundries have been certified as part of this program.

Trimberger [27] discusses the risks of untrusted foundries fabricating FPGAs. He mentions that the foundry does not know which devices are going to which customers, making it difficult to formulate an application-specific attack. The regular nature of the FPGA architecture, along with the huge manufacturing volume, allows for extensive testing that is amortized over all FPGA customers. Furthermore, the functionality of the reconfigurable hardware is programmed after fabrication. The finer the granularity of programmability, the less an attacker at a foundry can know about a design. As an example, compare a microprocessor and an FPGA. An attacker knows a good deal about how the microprocessor will execute programs and the structure of the design, allowing the attacker to target specific parts of the processor. It was shown that only a minimal amount of hardware (on the order of 1000's of logic gates) is required to hack a microprocessor, e.g. to obtain high level access [13]. However, an FPGA is an array of programmable logic, and the foundry has little idea as to what and where different parts of the application will be implemented. This potentially makes it much harder to attack the system. Reconfigurability can also be used to provide security. Assuming that the attacker only modified a portion of the fabric and that we are able to detect this modification, we could simply not use that portion. We could also continually move the secure data and processing cores around the programmable logic. We could also implement the design redundantly making it less likely that an attack works at all. Redundant designs must be modified in the exact same way. It should be stated that these suggestions all border on "security by obscurity," and more formal methods must be investigated.

The DARPA Trust in Integrated Circuits program acts as a follow-on program for commercial off-the-shelf devices that are not fabricated in a trusted foundry. Government officials realized that it is not feasible to develop all integrated circuits in a trusted environment, and they found it necessary to develop methodologies in an attempt to develop verification methods to trust circuits that are not homemade. One of the three thrusts of this program are "ensuring trust when employing field programmable gate arrays (FPGAs) in military systems first awards were made in".

3.1.4 Future Research

There is a pressing need to secure the entire semiconductor manufacturing supply chain. This includes trusted design kits, tool flows, design libraries, packaging, and assembly. Further research is required to understand malicious hardware "viruses". What can we learn from software red team exercises? How can we categorize these viruses, worms, Trojans, etc.? How can we analyze them? Is there an anti-virus program for hardware? Should it be based on signatures or behavior/anomaly? What about trusted delivery? Tamper resistance? What is the set of best practices to mitigate the threat of malicious hardware? What can we learn from the (failed?) efforts to detect subversion in software? What is a good defense strategy for hardware development?

3.2 Physical attacks

3.2.1 Problem Statement

Physical attacks monitor characteristics of the system in an attempt to glean information about its operation. These attacks can be invasive, semi-invasive or non-invasive. Invasive attacks include probing stations, chemical solvents, lasers, ion beams and/or microscopes to tap into the device and gain access to its internal workings. This involves de-packaging and possibly removing layers of the device to enable probing or imaging. Side channel attacks are less invasive attacks that exploit physical phenomena such as power, timing, electromagnetic radiation, thermal profile, and acoustic characteristics of the system.

3.2.2 Attacks

Passive, non-invasive attacks that physically probe the target device to glean information about its data or internal workings. Perhaps the easiest attack is to probe the wires on the board and/or the pins into the chip. However, the number of printed circuit board layers and modern packaging (ball grid arrays) make this increasingly difficult. More invasive "sand and scan" attacks remove passivation layers via chemicals, lasers or focused ion beams. This could theoretically provide access to internal wires to read data or access and manipulate the functionality of the device.

Semi-invasive attacks remove the packaging, but they do not physically damage or alter the chip. These attacks include analyzing the electromagnetic radiation emitted during the functioning of the chip. Other *side channels* are commonly used to gain information from physical characteristics of the system. For example, timing information, power consumption, supply voltage variations and thermal radiation provide an unintended source of information which can be exploited to break the system. Many side-channel attacks require considerable technical knowledge of the internal operation of the system.

Data Remanence is residual information that exists even after an attempt to erase or remove it. The data could remain for a variety of reasons including physical properties of the storage device or incomplete removal operations. An example of the former is that DRAM cells contain their contents for seconds to minutes after power-down, even at room temperature and even if removed from a motherboard [6]. An example of the latter occurs when a user deletes a file; it is removed from the file system but remains on the hard drive until another file overwrites it. Leakage of sensitive information through data remanence is possible if the device falls into the wrong hands. There are a number of general techniques to combat data remanence including overwriting, degaussing, encryption and physical destruction.

The question of how long the state of the FPGA remains after the device is powered off was studied by Tuan et al [28]. They found that data remanence was design and data dependent as the 1 and 0 bits discharge at different rates. Furthermore, the remanence varied across bits used to control the logic and those bits in the bitstream used to program the interconnect. This is due to the fact that they are powered by different supply voltages. They also quantify the effect that temperature plays on the remanence. It is well-known that SRAM bits at lower temperatures hold charge longer than those at higher temperatures.

The security implications are that FPGAs can retain the state of the bitstream for up to 2 minutes even after power off (when supply voltages are floating). Furthermore, since the remanence time depends on the state (1 or 0) as well as the type of programming bit (logic or interconnect) it may be possible to determine information about the unencrypted bitstream by studying the remanence of the bits. They showed that grounding the power supply reduces the remanence time; 20% information loss (this is the point where design becomes secure [21]) is achieved in less than 1 millisecond.

3.2.3 Solutions

Most physical attacks are not specific to FPGAs and have been studied in the context of other hardware devices. Physical attacks are perhaps the hardest to prevent; fortunately they are also the most technologically sophisticated, requiring substantial amounts of money and determination. For example, it is conceivable for focused ion beams to alter the wiring and bypass security features. However, this requires expensive equipment and significant know-how, especially when attacking modern devices fabricated with nanometer feature sizes. There are a number of solutions to physical attacks for microprocessors and ASICs. To a large extent, these can also be used for reconfigurable hardware. For instance, se-

cure co-processors such as the IBM 4758 [22] encapsulate hardware within a tamper-sensing and tamper-responding environment. Similar packaging can be used for FPGAs.

Timing, supply voltage and electromagnetic side channels for discovering the key(s) in cryptographic applications have been studied extensively [14, 5, 15]. None of these are specific to reconfigurable hardware, though the techniques are certainly applicable, at least to some extent. One of the few FPGA specific solutions to physical attacks is work by Yu and Schaumont, which provides a solution to the differential power attack [29]. Since techniques that make ASIC circuits resistant to side channel attacks do not necessarily translate to FPGAs, they developed a logic duplication and symmetric routing technique to reduce the risk of side channel attacks. Their methodology ensures that no matter what the input, the output power remains the same. They do this by creating a complementary design that is identical to the original design in terms of power dissipation, i.e., there is a 0 to 1 transition for every 1 to 0 transition and vice-versa.

3.2.4 Future Research Directions

Research directions for physical attacks largely revolve around red teaming to identify potential weaknesses. Often such weaknesses are quickly classified. Existing side channels are not particularly well studied, especially those specific to reconfigurable hardware. Obviously there are differences between ASICs and FPGAs in timing, power, temperature dissipation and other potential phenomena that could be exploited as a side channels. How to exploit and then eliminate or at least mitigate side channels remains an open and interesting research question.

3.3 Design Tool Subversion

3.3.1 Problem Statement

Programming an FPGA relies on a large number of sophisticated CAD tools that have been developed by a substantial number of people across many different companies and organizations. For example, AccelDSP [7] translates MATLAB [24] algorithms into RTL HDL, logic synthesis translates this HDL into a netlist, and a physical synthesis tool uses a place-and-route algorithm to convert this netlist into a bitstream. To further complicate the situation, IP cores are an increasingly common methodology for design reuse. These are distributed in many different forms, including RTL HDL, netlists or as a bitstream.

In the end, these different design tools and IP produce a set of inter-operating cores, and as Ken Thompson said in his Turing Award Lecture, "You can't trust code that you did not totally create yourself" [25]. You can only trust your final system as much as your least-trusted design path. Subversion of the design tools could easily result in malicious hardware being inserted into the final implementation. For instance, if there is a critical piece of functionality, perhaps an encryption core which holds secret keys, there is no way to verify that this core cannot be tampered with or snooped on without completely building the entire system yourself, which includes all of the design tools that you use for synthesis as well as the reconfigurable hardware. Alternatively, you must develop ways in which you can trust the hardware and the design tools. We discussed the former in Section 3.1, and we tackle the later in this section.

Domain separation and isolation is a fundamental principle of secure systems, and plays a large role in design tool subversion. Saltzer and Schroeder define complete isolation as a "protection system that separates principals into compartments between which no flow of information or control is possible" [19]. Of course, no system works in complete isolation, and systems must allow for communication or sharing of data.

Often, specific portions of the chip are more sensitive than others. A prime example is an encryption core. Providing assurances on isolation of cores allows us to use IP cores with various levels of trust. In this way, design tool subversion problems are to a large extent similar to those found in multilevel security (MLS). The IP cores may hold information with different sensitivities (e.g. security levels). They should allow access to information and communication only to those IP cores that have appropriate permission and prevent IP cores from obtaining information for which they lack authorization.

3.3.2 Attacks

Covert channels are an attack where rogue IP cores use a shared resource to transmit information without the authorization or knowledge of the shared resource. For example, a malicious core may transmit information about

the internal state of the reconfigurable hardware using an IP core that controls some I/O (audio, video, radio).

Side channels, which are described in Section 3.2.2, are another potential avenue of attack. The IP cores have access to internal resources of the chip, allowing them to tap into side channels in a much easier way than at the chip or device level.

A *bypass* circumvents security features through rerouting, spoofing or other means. An example of bypass is to reroute data from one core to another, or even to I/O pads and therefore out of the chip. Bypass is risky because, unlike narrow bandwidth covert channels that are difficult to exploit, bypass can present a large, easily exploitable overt leak in the system.

3.3.3 Solutions

Moats and drawbridges are a statically verifiable method to provide isolation and physical interface conformance for multi-core designs [8]. It employs a logical separation technique that spatially separates the IP cores on an FPGA, relying on design tools that enforce restrictions on the placement and routing of specific cores. "Moats" are buffer zones surrounding the core that one wishes to isolate. Static analysis and interconnect tracing are performed on the bitstream to ensure that only specified communication connections ("drawbridges") between cores are permitted.

A similar idea called "fences" was simultaneously proposed by McLean and Moore [17]. Though they do not provide extensive unclassified details, they appear to be using a similar technique to isolate regions of the chip by placing a buffer between them (aka *fence*) which is analogous to moats. They have been working closely with the NSA and have performed an exhaustive vulnerability assessment, and they have met the standards of the NSA Fail Safe Design Assurance (FSDA) specification.

Sanitization or *redaction* is the process of removing sensitive information, so that it may be distributed to a broader audience. The familiar case of redaction is the "blacking out" of text in a document that corresponds to sensitive information. A similar process could occur with sensitive data stored in reconfigurable hardware. A bitstream sanitization procedure called *configuration scrubbing* for applications that perform partial reconfiguration is described in [8]. This uses the ICAP to zero out all of the flip-flops and configuration bits in the area previously occupied by the core being swapped out. This ensures that the next core that occupies that space of the logic fabric cannot obtain any data stored by the previous core.

3.3.4 Future research

The problem of trusting the design tools is a difficult one, and there is much room for research in this area. Possible solutions include the development of trusted tools that are stripped-down versions of commercial design tools. This is unattractive because each synthesis stage has substantial opportunities for optimizations, which the current tools heavily employ. Furthermore, even simple versions of these tools would be quite complex. Other ideas include verification of each stage of the design flow by applying formal methods, model checking and/or theorem proving. This is currently done to some extent in design tools; however, the scope of what can be formally verified is small. Methods to increase the scope are needed.

Other questions for future research in this area include: Can we apply configuration management and/or static analysis to HDL code? Is there a way to develop "safe" hardware languages? Can we assure the redaction of sensitive information once it leaves the IP core?

3.4 Design Theft

3.4.1 Problem Statement

The bitstream necessarily holds all the information that is required to program the FPGA to perform a specific task. This data is often proprietary information, and the result of many months/years of work. Therefore, protecting the bitstream is the key to eliminating intellectual property theft. Furthermore, the bitstream may contain sensitive data, e.g. the cryptographic key.

3.4.2 Attacks

Cloning is the unauthorized copying of the design. This attack is dangerous because once the bitstream is in hand, it is rather easy to buy another FPGA and use that bitstream to create a replica of the system from which the bitstream was stolen. Such a counterfeit device can be sold more cheaply, since a significant portion of the costs are due to the design of the application running on the FPGA.

In many systems, the bitstream is stored externally in non-volatile external memory. In such a case, the attacker can either directly read the memory or eavesdrop on the bus when the FPGA is powered up and the bitstream is feed into the device.

Reverse engineering is the process of discovering properties of the design, e.g., by translating the bitstream into some higher level form. This is done with the intention of making a device that does a similar thing or analyzing the internal data or structure of the device, e.g., to find out the encryption key.

A *readback attack* directly obtains the bitstream from the functioning device. Many FPGAs allow the configuration to be directly read out of the device either through JTAG, ICAP or another similar bitstream programming interface.

3.4.3 Solutions

Design theft is a serious financial threat to application developers. FPGA vendors have realized this and have developed a number of solutions to enhance the security of the bitstream. As such, this is probably the most researched problem that we will discuss. Unfortunately, we do not have the space to describe every piece of research in this area, and we focus on the basic tenets.

One way to protect the bitstream is to program the FPGA at a secure facility and constantly keep the FPGA powered during field deployment. The bitstream is only resident within the FPGA, and it cannot be easily stolen assuming that bitstream readback functions are not present. The downside is that the device must be powered constantly, and it does not allow for reprogramming. It should be noted that this is very similar to the security provided by antifuse and other non-volatile FPGAs. Of course, non-volatile FPGAs do not require power to keep the bitstream resident, as it is burned in during manufacturing.

The next logical step for protecting the bitstream is through encryption. Encryption relies on an invertible non-linear function making the bitstream unreadable to anyone who does not possess the key. Bitstream encryption was first suggested in a patent by Austin [3]. The bitstream is encrypted and stored in external memory. It is then read into the FPGA, decrypted using a key stored in non-volatile memory, and used to configure the FPGA. This requires that either all devices have the same key (a very bad idea which was done in Actel 60RS family of SRAM programmed FPGA [12]), or each device has a custom key burned. The Xilinx Virtex II family used a variant of this idea, but it stored the key in a volatile register which was powered by a battery [26]. Since we only need to power the key, a small battery will last for quite some time (on the order of 10 years). There is a dedicated decryption core resident on the Virtex family architectures, and features such as disabling readback, eliminating read/write to the key registers, and bitstream integrity checks are in place to restrict access to the key.

Some FPGAs include an interface that provides the device access to its bitstream. This enables partial reconfiguration, performs bitstream decompression, and is used to decrypt the bitstream. This can also be used to detect and correct bitstream errors, whether it be from single event upsets, or intentional malicious attacks performed in the field.

Digital watermarking can be performed on the design, which embeds a hidden signature that can be later used to prove design theft. These signatures can vary significantly, targeting different parts of the design. One example of digital watermarking would be to modify the routing of wires in such a way that indicates a certain signature. When done properly, this is hard to detect and can be shown with great certainty that it is an intentional watermark and not a random occurrence in the physical synthesis tools. An article by Kahng et al [11] presents the fundamentals of integrated circuit watermarking and describes a method for watermarking FPGAs at the physical level, which involves manipulating unused portions of the configuration bitstream.

Encryption provides confidentiality of the design, but it does not say anything about its authenticity. Authentication gives an assurance on the identity or source of the bitstream. This allows the FPGA device to confirm the developer of the bitstream that it is loading, and makes certain that it is not modified in any way. Furthermore, authentication can be used for the reverse process – to ensure that an IP core is only run on authorized FPGAs. Simpson and Schaumont [20] describe a mutual authentication of IP cores and the reconfigurable hardware. They use physically unclonable functions (PUFs) [23] to create a unique signature for each specific piece of hardware. This along with an identity of the IP core developer is de-

livered to a third party initiating a protocol which verifies the identity of both the hardware and software.

3.4.4 Future Research

As we mentioned, this is a well researched area, particularly in industry, as IP theft has substantial financial implications. However, there is still room for more research in this area including: How can we make the bit-stream protection mechanisms more resistant to an attack from a determined adversary, such as a foreign government? What about the malicious insider attack (e.g., Xilinx or Altera employees)? How strong is the authentication in these remote update channels? How can we perform bit-stream encryption in applications that use partial reconfiguration? PUFs provide a great way to uniquely identify a specific FPGA. This is a fundamental need for many security primitives. How can we best utilize this? And what is the best way to generate PUFs on FPGAs?

3.5 System Security

3.5.1 Problem Statement

The problem that we are dealing with here is the ability to make assurances that during the deployment of the device it does not do anything potentially harmful. This requires that we perform monitoring of the system and provide the system with information on what it can and cannot do.

3.5.2 Attacks

A *denial-of-service (DoS)* attack is an attempt to make a computer resource unavailable to its intended users. A simple example of a DoS attack is reprogramming, turning off, or destroying an FPGA that functions as a network router.

There are many physical attacks that can be used during deployment to steal the sensitive data/information that is stored within the FPGA, e.g., passwords, personal and financial information (credit card numbers, social security numbers), encryption keys, and so on. These physical attacks are described in more detail in Section 3.2.

Reconfiguration obviously plays an important role in the life cycle of reconfigurable hardware. Indeed, the ability to reprogram the hardware to fix design errors, provide system updates and/or perform partial reconfiguration is an attractive feature. However, this opens a number of security holes in the system. The updates must be delivered to the system in a secure manner. The updates may be performed through a physical reprogramming of the FPGA; however, it is certainly feasible and desirable for the FPGA to be updated remotely, e.g., through the network. In both cases, the updating agent must somehow authenticate itself. Authentication is done using a variety of methods, each of which has its benefits and drawbacks. Biometric techniques such as fingerprint readers or retina scanners are obviously very secure, but they require physical access to the system, and they add cost to the system. Less secure methods include passwords and public key cryptography. These methods are well-studied techniques, and applying such techniques employed in other secure systems to FPGAs systems is certainly possible. Authentication techniques using PUFs (as described in Section 3.4.3) is another potential solution.

3.5.3 Solutions

To provide memory protection on an FPGA, we proposed the use of a reconfigurable reference monitor that enforces the legal sharing of memory among cores [9]. A memory access policy is expressed in a specialized language, and a compiler translates this policy directly to a circuit that enforces the policy. The circuit is then loaded onto the FPGA along with the cores. We can use moats and drawbridges to protect the reference monitor from routing interference and to prevent the reference monitor from being bypassed.

3.5.4 Future Research

Since this research area lies in the deployment stage of the life cycle, we must rely upon solutions to many of the problems in earlier life cycle stages, which are described in previous sections. One broad research question that encompasses the entire lifetime is: How do we achieve high-assurance FPGA systems using formal methods, system evaluation, etc.? Other research possibilities involve applying security primitives to multi-core systems to enforce multilevel security/information flow (e.g., Bell and LaPadula). Another promising technique in this area is the use of data tags, which carry information about where the data has resided, and policies about what we can and

should be done with that tagged data. Another topic that spans across problem areas and life cycle stages is: how can we prevent a denial-of-service attack launched by a "malicious core"? Other cross-cutting topics are: can we develop a secure form of virtual memory to provide resource arbitration for FPGAs? And how can we apply dynamic security results to the problem of partial reconfiguration?

4 Conclusions

We addressed problems of reconfigurable hardware security by providing attacks, solutions and areas for future research. We started with a discussion of the life cycle of reconfigurable hardware. We divided the life cycle into the manufacturing stage, the application development stage and finally the deployment stage. Each stage has unique security implications, and we attempted to address the major problems facing the design and use of FPGAs in sensitive applications. We specifically addressed the topics of trusted hardware, physical attacks, design tool subversion, design theft and system security.

References

[1] S. Adee. The hunt for the kill switch. In *IEEE Spectrum Online*, May 2008.

[2] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. Trojan detection using IC fingerprinting. In *Proceedings of IEEE Symposium on Security and Privacy*, 2007.

[3] K. Austin. Data security arrangements for semiconductor programmable devices. *US Patent 5,388,157*, February 1995.

[4] Defense Science Board. High performance microchip supply, February 2005.

[5] K. Gandol, C. Mourtel, and F. Olivier. Electromagnetic analysis: concrete results. In *Proceedings of CHES*, 2001.

[6] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *Usenix Security Symposium*, 2008.

[7] T. Hill. AccelDSP synthesis tool floating-point to fixed-point conversion of matlab algorithms targeting fpgas, April 2006.

[8] T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, and R. Kastner. Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2007.

[9] T. Huffmire, S. Prasad, T. Sherwood, and R. Kastner. Policy-driven memory protection for reconfigurable systems. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, Hamburg, Germany, September 2006.

[10] ISO/IEC. ISO/IEC 15408 - common criteria for information technology security evaluation, version 2.3, August 2005.

[11] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe. Constraint-based watermarking techniques for design ip protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(10), 2001.

[12] T. Kean. Secure configuration of field programmable gate arrays. In *Proceedings of the 11th International Conference on Field Programmable Logic and Applications (FPL '01)*, Belfast, UK, August 2001.

[13] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou. Designing and implementing malicious hardware. In *Proceedings of Usenix Workshop on Large-Scale Exploits and Emergent Threats*, 2008.

[14] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems. In *Proceedings of Crypto*, August 1996.

[15] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Proceedings of Crypto*, August 1999.

[16] J. I. Lieberman. White paper: National security aspects of the global migration of the u.s. semiconductor industry, June 2003.

[17] M. McLean and J. Moore. Securing FPGAs for red/black systems: FPGA-based single chip cryptographic solution. In *Military Embedded Systems*, March 2007.

[18] OpenCores. http://www.opencores.org.

[19] J. Saltzer and M. Schroeder. The protection on information in computer systems. *Communications of the ACM*, 17, 1974.

[20] E. Simpson and P. Schaumont. Offline HW/SW authentication for reconfigurable platforms. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2006.

[21] S. Skorobogatov. Low temperature data remanence in static ram. In *Cambridge University Technical Report UCAM-CL-TR-536, ISSN 1476-2986*, June 2002.

[22] S. W. Smith and S. H. Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks (Special Issue on Computer Network Security)*, 31, 1999.

[23] G. E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. In *Design Automation Conference (DAC)*, 2007.

[24] The Math Works Inc. MATLAB User's Guide, 2006.

[25] K. Thompson. Reflections on trusting trust. *Communications of the ACM*, 27(8), 1984.

[26] S. Trimberger. Method and apparatus for protecting proprietary configuration data for programmable logic. *US Patent 6,654,889*, 2003.

[27] S. Trimberger. Trusted design in FPGAs. In *Proceedings of the 44th Design Automation Conference*, San Diego, CA, USA, June 2007.

[28] T. Tuan, T. Strader, and S. Trimberger. Analysis of data remanence in a 90nm fpga. In *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC)*, 2007.

[29] P. Yu and P. Schaumont. Secure fpga circuits using controlled placement and routing. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS 2007)*, October 2007.