Volatile FPGA design security – a survey

Saar Drimer*
Computer Laboratory, University of Cambridge
http://www.cl.cam.ac.uk/~sd410

Version 0.95, December 6, 2007

Abstract

Volatile FPGAs, the dominant type of programmable logic devices, are used for space, military, automotive, and consumer electronics applications which require them to operate in a wide range of environments. Their continuous growth in both capability and capacity now requires significant resources to be invested in the designs that are created for them. This has brought increased interest in their security attributes; specifically, how well does the FPGA protect the information processed within it, how are FPGA designs protected during distribution, and how developers' ownership rights are protected while designs from multiple sources are combined. This survey, which assumes very basic logic design and security understanding, establishes the foundations for discussing "FPGA security", examines a wide range of attacks and defenses along with the current state of industry offerings, and finally, outlines on-going research and latest developments.

1 Introduction

Field Programmable Gate Arrays (FPGA) are generic semiconductor devices comprising of interconnected functional blocks that can be programmed, and reprogrammed, to perform user-described logic functions. In order to compete with the higher performance generally attributed to Application Specific Integrated Circuits (ASIC), volatile SRAM-based FPGAs are typically manufactured using the latest technology advancements and are available off-the-shelf to system developers. Together with their reprogrammability, these attributes give them advantages in many applications where the up-front costs and development time associated with ASIC design cannot be justified. FPGAs also outperform sequential microprocessors, being able to execute unique functions and parallelize operations. Since the early 2000s, FPGAs have gradually absorbed into them the capability of devices that were previously in their periphery. 2007's FPGAs have embedded processors, giga-bit serial transceivers, clock managers, analog-to-digital converters, dedicated digital signal processing blocks, Ethernet controllers, substantial memory capacity, and other dedicated functional blocks beyond the basic arrays of simple logic elements they started out with in the mid 1980s. This growth in capacity and popularity has

*The author is supported by a grant from Xilinx, Inc.

two main security implications. Firstly, greater resources are invested in designing for FPGAs, resulting in high-value designs that need to be protected. Secondly, FPGAs are increasingly being used in applications that require security features and properties that are either not available today, or that have yet to be adequately investigated. Both have brought recent attention to the security attributes of FPGAs in the military, automotive, and consumer industries, and also the research community, each with its own set of requirements and security perspective.

This document surveys the current state of the FPGA security field with the goal of being a reference to engineers and researchers interested in the challenges of providing protection to FPGA designs in both distribution and operation[1]. As the basis for our discussion, we will start with defining the FPGA usage model and its participants in Section 2. Possible attacks are described in Section 3 followed by defenses in Section 4. Section 5 covers the current state of research and finally, Section 6 deals with issues of trust, adversary classification, and security metrics.

This survey is primarily about the security of SRAM-based volatile FPGAs and does not, in most part, cover non-volatile programmable logic devices or one-time programmable devices. While some of the discussion will apply to non-volatile devices, they should be considered separately as they have a different usage and threat models. They do have attractive attributes, however, and are generally regarded as more secure, but with a compromise made on performance and capacity [76]. Finally, many of the issues discussed in this document are not necessarily exclusive to FPGAs and may apply to a wide range of integrated circuit devices, software, or pertain to security engineering practices in general. The focus here, however, is to put all those in the perspective of the FPGA usage model.

2 Usage model

The FPGA's usage model is unique, with the features that provide its inherent flexibility also being the ones exposing it to security vulnerabilities. In this section we will discuss this model, its participants, and their interaction. Typical development flows of the FPGA itself and an FPGA-based system are also described.

2.1 Principals

"A principal is an entity that participates in a security system. This entity can be a subject, a person, a role, or a piece of equipment, such as a PC, smartcard, or card-reader terminal" [10, p9]. FPGAs, FPGA vendors, designers, programming cables *etc.* are principals interacting in a system whose security we are concerned with. With our security perspective, we can consider their communication as part of the execution of a security protocol. Introduced below are the principals that partake in the design and distribution of FPGA products, along with their security requirements[2].

[1] In early 2004, Wollinger, Guajardo, and Paar wrote the first comprehensive survey of the topic [119], which is a good companion to this one.

[2] In 2002, Kean [53] provided a list of principals in the context of "intellectual property" protection, which is expanded here to cover all aspects of the design flow.

FPGA vendor. In 2006, Altera [8] and Xilinx [120] together held over 80% [32] of the programmable logic market with Lattice [64] being a distant third with 6% market share. The competition for market dominance is fierce with the customers being the beneficiaries as the vendors are well-tuned to their needs. Pushing towards the "leading edge" of technology is a balancing act between reaping the rewards from the introduction of new "differentiating" features and falling over that edge due to their failure. Each company introduces a new family of devices about once every 12–18 months, each costing many millions of dollars do design and fabricate, with a failure potentially being the company's downfall. The vendors, thus, are also conservative, introducing only features that are needed by the majority of their customers, or are in-line with a long-term market strategy and foresight (for example, embedded hard processors or large memory blocks). This means that for new security features to be introduced, they must be needed by the majority of users, or alternatively, by several large ones. Resources are limited, with one feature taking the place of another and therefore, new features must always be justified by need rather than what would be nice to have; also, the cost of new features is paid for by all users, not only by ones that will use them, so all users must be willing to pay for them. This point is important to remember as we consider some of the proposed security solutions later in this survey.

Security-wise, FPGA vendors have two dominant concerns. Firstly, they need to protect their own proprietary designs and techniques from being reverse engineered, copied, exposed, or modified. Secondly, they need to provide their customers means to protect their own designs throughout the design flow and in the field. Customers' recent interest in protecting designs has made security become a competitive factor prompting FPGA vendors to pay more attention to these issues, and the gradual increase in security related feature offerings reflects this.

Foundry. All FPGA vendors are *fabless*, which means that they design the FPGAs but partner with other companies, called *foundries*, for manufacturing. They are presented here as an independent principal since they are crucial to the security of the FPGA. For example, it is possible for them to modify the designs, or they may be tasked with embedding unique secret keys and serial numbers into the devices. In the past, trusted foundry services were established which were local to the country where the devices were designed, and strict control and supervision was possible. This is significantly harder to enforce today as most advanced fabrication facilities are in Asian countries. Commercial companies may not care so much, or not have the funds for auditing the fabrication process, though governments do care, and Asian foundries would not be so inclined to be supervised by some. To attest the important of this, at least as far as governments are concerned, a 2005 report [114] by the U.S. Department of Defense discusses the "alarming" rate in which "critical" microelectronics facilities are migrating to foreign countries.

System developer. The FPGA vendor sells the FPGAs, often through distributors, to the system developer who incorporates it into a product. System developers can be divided into two groups based on their security needs and views. Using this categorization we will be able to evaluate the effectiveness of various defenses from their perspective.

- *Cost-conscious.* The goal of the commercial product designer is to meet the prod-

uct's specifications at the lowest cost, while maintaining reliability. Often, there is a trade-off between system performance and cost and a general tendency by engineers to resist additional components, design delays, and decrease in reliability that translates into higher maintenance costs and support. The typical life-cycle of a commercial product is quite short, from months to a few years, and therefore, its design and embedded secrets may only need to be protected for that long. In the high-pace commercial markets, designs and/or products are made obsolete within months to a few years, and thereafter are hardly a worth-while target for attackers. The predominant threat faced by commercial product designers is the creation of a competing cheaper product that is derived from the original. Therefore, it is sufficient to make the process of stealing the design at least as costly as re-creating it, while defenses to achieve this should have minimal impact on the total system cost. A slightly different approach would be to create a defense that is just slightly more robust than a competitor's.

- *Security-conscious.* Government contractors and security-industry system developers are concerned with protecting designs, methods of operation, and communications for a substantial length of time — from years to decades — while cost considerations may be secondary. The security-conscious designer is often interested in robust, "approved" security mechanisms, based on established protocols and algorithms. In these applications, cost-performance may be important, but not at the expense of security and reliability, and these developers are willing to pay for it. Some security-conscious designers make use of older, more mature, and hence more reliable, integrated circuits. Others take advantage of the most recent technologies that are more resistant to invasive attacks and increase the entry threshold of potential adversaries due to the higher cost of equipment and know-how.

It appears that FPGA vendors have a challenge in supplying security features to their customers: in a single resource-limited device, they would (ideally) like to satisfy both cost- and security-conscious designers, who have significantly different outlook on security, and what they are willing to spend for it.

Cores designer. *Cores* are ready-made functional descriptions that allow system developers to save on design cost and time by purchasing them from third-parties and integrating them into their own design. *Cores designers* sell[3] cores in the form of Hardware Description Language (HDL) or complied netlists; there are cores on offer for nearly any popular digital function[4]. An ideal model for cores distribution would enable system designers to evaluate, simulate, and integrate them into their own designs, while maintaining confidentiality of the cores; limit the number of instances that can be made of them; and, make them operational only on specific devices. A benefit of this model is that it will also enable a pay-per-use — rather than blanket licensing — payment system that would open the availability of such cores to a wider set of end-users. To date, there are

[3]FPGA vendors provide some cores for free (they make the money from selling the FPGAs), as do open-source designers.

[4]A single third-party core can occupy the entire FPGA; Kean [53] calls these cores "Virtual Application Specific Standard Products".

no mechanisms to enable this "core distribution business" that answers to all the needs specified above. A more detailed discussion of these issues is in Section 5.3.

EDA software vendor. Electronic Design Automation (EDA) tools are used for the development of printed circuit boards, integrated circuits, FPGA designs, and extensively used for simulation. The various EDA vendors provide the tools that are used by all the principals mentioned above with FPGA vendors also being EDA tool suppliers. Therefore, EDA software vendors play a pivotal role in the FPGA design flow and their contribution is critical to the security of both the FPGA and FPGA-based products.

System manufacturer. The system developer does not usually have the facilities to mass produce a product, so therefore, the designs are sent to the system manufacturer for production and often also for testing. This principal includes all parties involved in the process of making the system ready for delivery: printed circuit fabrication, assembly (where components are soldered onto the board), testing, and even packaging.

System owner. This principal possesses the FPGA-based system, which is no longer under direct control or supervision of its developer. This could be a legitimate user who purchased the system at a consumer-product store, or a government who obtained it from a fallen reconnaissance aircraft; both may be malicious (or the "enemy"), trying to pry secrets out, or circumvent protection mechanisms. Depending on the product, the system developer may restrict the owner from using certain functions in order to prevent theft of services, or from executing unauthorized code on the system. For example, set-top box developers (such as TiVo) profit from providing programming services, not from supplying the hardware itself. Therefore, they have an incentive to invest in mechanisms that prevent the theft of these services. Some cell phone manufacturers have mechanisms to prevent users from choosing a network other than the one they are supposed to be locked into. The security-conscious designer may want to have mechanisms to completely erase or destroy portions of the system when it falls into the "wrong" hands and perhaps employ the ability to "call home" upon tampering.

Trusted party When two or more principals negotiate, they sometimes need another principal whom they all trust to be honest, and accept its output or ruling. In day-to-day life this might be a court judge, while in security protocols, this principal is called a *trusted party* that may be entrusted with keys, execution of cryptographic processes, and secure storage. Often, principals do not communicate between themselves, but only do so through trusted parties, which are required to preserve confidentiality and authenticity of information. Trusted parties are best avoided to maximize robustness, though they may be essential in order to meet the security goals of a protocol, as we shall see when we discuss the FPGA design flow next.

2.2 Design and manufacturing flow

An FPGA is an ASIC for the FPGA vendor with similar design and manufacturing processes, as depicted in simplified form in Figure 1. The design files (mostly HDL)

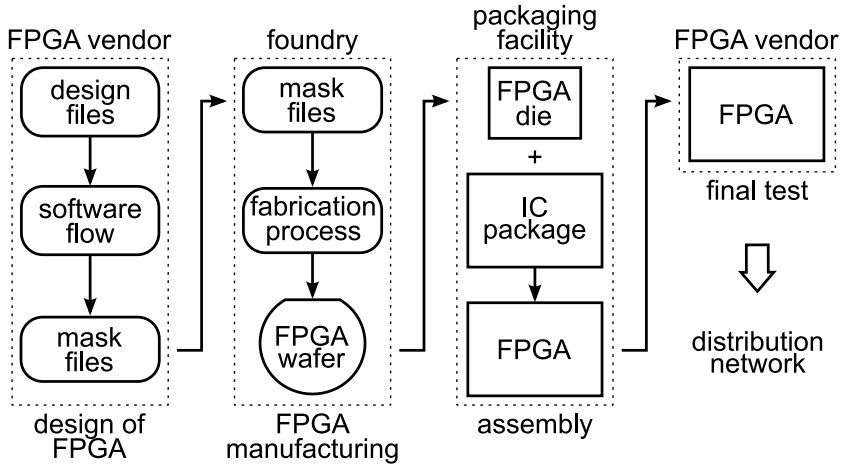

Figure 1: Simplified depiction of the FPGA design, manufacturing, packaging, and testing processes.

are processed by software tools to produce a netlist, which is then laid-out to provide the physical representation of gates and transistors. From the layout, files describing the mask sets used to manufacture the device are created and delivered to the foundry where they are physically created (this is a simplification, masks may not necessarily be manufactured by the foundry, but by another principal). The fabrication process produces wafers, then these are tested for good die which are identified and marked. The wafers are then sent for assembly where the good die are cut and attached to a carrying package. Finally, these packaged die are sent back to the FPGA vendor for final testing before they are sent for distribution.

Figure 2 shows the design and manufacturing processes of the system developer who uses an FPGA in his product. It is not meant to be a complete description, but rather to provide the points at which principals interact. It is important to review these processes as every attack or vulnerability we will discuss can be placed at one or several points of this model. In the development phase, internally- and externally-designed cores are combined to describe the desired logical functions of the FPGA along with the prototyping of the hardware in which it will operate. The software flow, as shown in Figure 3, starts with HDL *synthesis* that optimizes and translates the functional description according to the resources available in the target FPGA architecture (*e.g.* Stratix look-up table, Spartan multiplier) into a *netlist*. Netlists contain a description of the instantiated primitives and the connections between them, usually in the standardized Electronic Design Interchange Format (EDIF). Synthesis tools are available from several EDA vendors, not necessarily from the FPGA vendor of the target device, unlike the tools used for the rest of the process. The information contained in the netlist is then *mapped/fitted* to the specific primitives of the architecture and then those are *placed and routed* to a particular target device to produce a *placelist*[5], where the specific route of every interconnect and physical placement of all primitives are described. The placelist is then *encoded* to produce a *bitstream* file that when loaded onto the FPGA establishes the routing to and from all

[5]This new name is used in order to distinguish the information described in these files from the information described in netlists.

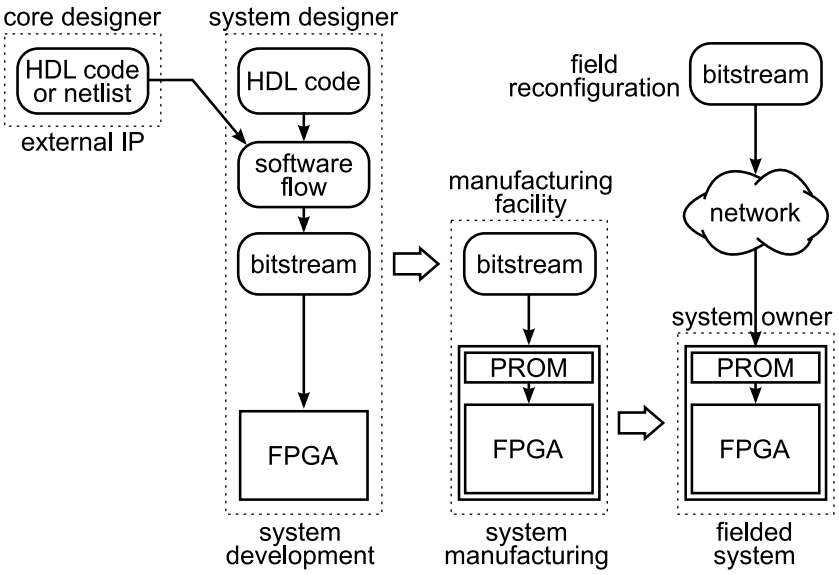

Figure 2: The development, manufacturing, and distribution of an FPGA-based system. The system developer must be assisted by several other principals such as manufacturers, and cores and EDA vendors. At the end of the development cycle the product is in the system owner's hands.

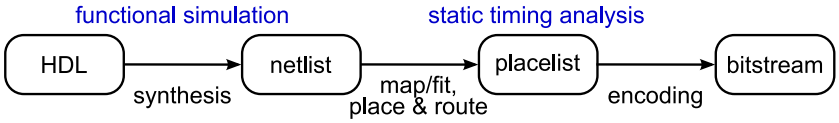

Figure 3: Expanded view of the software flow used to process a functional description in a high-level language into a bitstream file that is programmed into the FPGA to have it perform this functionality.

the instantiated elements by setting the state of memory cells, pass gates, and routing switches. The attribute settings and initial content for all primitives that are used is also set by the bitstream, with end result of the FPGA performing the logic functions that were initially described by the HDL. As SRAM FPGAs are volatile, they must receive the bitstream on every power-up from an external source, which is traditionally a non-volatile (NV) device, EEPROM or Flash, placed nearby on the printed circuit board. The design is simulated at the HDL, netlist, and post-PAR stages, and is also verified for correct operation when "executed" on the FPGA itself in its intended hardware setting. *Static timing analysis* takes into account the architecture and actual delays after the place and route process in order to verify that timing violations, such as of setup and hold timing allowances, do not occur. When the prototyping process is done, the system is manufactured, the bitstream is programmed into the device or NV storage, and undergoes final testing before being shipped. In the field, the finished product is in the hands of the system owner and thus, no longer under the control of the designer. At this stage, the functional definition, or bitstream, can be updated via the process of *field reconfiguration*. For example, a firmware upgrade to a digital camera may be done remotely by the user

plugging it into an Internet-connected PC, or an upgrade to a car's processor be done by a service technician at a service station.

3 Attacks

Security is an arms-race. Defenses against malicious tampering or use are put in place only to be broken and later be replaced by other, hopefully better, measures. Smartcards and microcontrollers have been in the midst of such a race for the past two decades, starting with naive, if present at all, security mechanisms and incrementally improving as exploits were discovered; examples are the work of Anderson, Kömmerling, Kuhn, and Skorobogatov [11, 12, 59, 94]. The interest in finding vulnerabilities in smartcards emerged in the mid-1990s when they started being used for pay-TV application, and later for banking; Anderson *et al.* [13] provide a recent survey of the security properties of cryptographic processors.

We are now seeing the beginning of such an arms-race for FPGAs. With FPGAs being used in more applications that necessitate various degrees of security, and as these designs become more valuable, attackers search for vulnerabilities and developers for defenses. This section examines possible attacks against FPGAs; a subset of those are unique to FPGAs though most apply to other devices and systems but are considered here for their relevance to FPGAs. The first few attacks are closely related to the usage model, bitstreams and their distribution, followed by attacks on the FPGA while it is operating and physical attacks, ending with system-level attacks.

3.1 Cloning, overbuilding fraud, and mislabeling

FPGAs are generic, which means that a bitstream made for one device can be used in any other of the same family and size. As such, attackers can, and do, *clone* bitstreams by recording them in transit to the FPGA and use them in other systems or products, usually cheaper clones that compete with the originals. Since cloning requires no more than a logic analyzer and a competent technician, it is considered to be the worst security vulnerability of volatile FPGAs. The attacker, who does not need to understand the details of the design regards it as a black-box, and only needs to invest in copying the circuit board the FPGA is mounted on, saving on the significant development costs. The original system developers have two main concerns with regards to cloning. Firstly, the cloned systems take away significant profit as the FPGA design represented in these bitstreams is the result of many months of development. Secondly, if the clone is presented as the original to the consumer, in addition to sales loss, it erodes the original developer's reputation because the fake is almost universally of poorer quality. Since this is the path of least effort for the attacker, an increase in cost for a successful attack may act as an effective prevention mechanism to make cloning unprofitable, at least for a while.

The electronic industry has been facing increased amounts of counterfeit hardware in the last decade, especially sourced from Asian markets. Aside from cloning of systems, *over-building* or *run-on fraud* is a major concern to many big companies. When a product is manufactured at a third party facility that fabricates, assembles and tests the hardware

before it is shipped to the customer, it may manufacture more than the ordered quantities and sell the excess without incurring development costs. They may even sell the designs themselves (PCB layout, bitstreams) to competitors, encroaching on the original designer's profits. To avoid this, companies qualify facilities as "trusted" and supervise/audit them, but this is hard to do in many countries, and may even be unaffordable for small companies. Preventing run-on fraud is challenging, and no robust solution is yet to be implemented.

Mislabeling of FPGAs is also a problem for both FPGA manufacturers and system developers. Modifying or erasing markings on an IC package is trivial, and system designers have been doing so for years to make the reverse engineering of a system incrementally harder. But when FPGAs are not through manufacturer-endorsed authorized distributors[6], how can the buyer be sure that the packaging's markings match what is inside the package? If it is a completely different device, or even a smaller FPGA family member, that would be quite simple to verify, albeit only after the purchase. Speed grades are harder to measure, though, and a slower die may be marked and sold as faster ones for a premium. There is no way for the buyer, or seller to attest to the authenticity of devices, aside from programming it with a bitstream and observing correct results, though for verifying the speed grade expensive equipment may be required. For commercial companies it is probably safest to buy devices from the vendors or their distributors rather than on-line, though for hobbyists that only need low quantities for cheap, the risk may be worth it.

Numbers for these types of fraud are hard to come by as companies are not ready to disclose their losses unless they must. An industry consortium of large hardware development companies called the Alliance for Gray Market and Counterfeit Abatement has estimated that in 2006 [7] one in ten purchased products were fake either by over-building or cloning. These types of fraud are hard to prevent, especially when they occur in places where ownership rights are not enforced. We will discuss some counter measures against the issues above in section 4.

3.2 Reverse engineering the bitstream

We can define *bitstream reversal* as the transformation of an encoded bitstream file into a logical functional description that is equivalent to the original design which produced it. In effect, reversing the process described in Figure 3 to arrive at a netlist or HDL. Partial bitstream reversal can be further defined as the extraction of data from the bitstream, such as keys, BRAM/LUT content, or memory cell states, without reproducing full functionality. Reverse engineering is legal — with some restrictions — in many countries for interoperability reasons or discovery of infringement of patents and other rights [77]. Full bitstream reversal would, of course, reveal the entire design and the data could be used to produce another bitstream which is completely different from the original such that it is hard to prove that reversing actually occurred. If the designer relied on the obscurity of the bitstream's encoding to protect keys, these would likely to be compromised as well; partial reversal can also be fruitful if, for example, clues about the type of cipher or filter co-efficients used are sought after.

[6] Obsolete devices can still be found on-line years after the manufacturers abandoned them.

In the early 1990s, start-up NeoCAD has created complete FPGA design development tools for various vendors' FPGAs. Contrary to common belief, and according to Xilinx [67, 111], NeoCAD did not in fact reverse engineer the bitstream, but rather, the bitstream generation executable so that their tools could generate compatible bitstreams for Xilinx FPGAs. In 1995 NeoCAD was acquired by Xilinx to become its software division. In the late 1990s, start-up Clear Logic was able to use Altera's software-generated bitstreams to produce pin-compatible, smaller, cheaper, laser-programmable ASICs (they were also more secure since they did not require an external bitstream source). Altera subsequently sued for damages while requesting the halting of Clear Logic's operations. In 2001, Clear Logic was barred from asking its customers to use Altera's software tools since it violated their End User License Agreement (EULA), and in late 2005, Altera won the case and was awarded damages [9, 115]; Clear Logic had already ceased operations by 2003. Although both cases involved the *generation* of compatible bitstreams, neither company was able to completely reverse engineer them to obtain the original functional description. NeoCAD reverse engineered a binary executable and Clear Logic used existing bitstreams to laser-program devices made from reverse engineering the FPGA's mask sets. We must also remember that compared to today's FPGAs, the ones of NeoCAD and Clear Logic's time were much less sophisticated; so even if they did reverse engineer bitstreams, the task would have been considerably easier than doing it today.

The bitstream's encoding is largely undocumented and obscure, but not encrypted or confidential in a cryptographic sense, with FPGA vendors keeping this encoding a secret as they do for the chip's own design and layout information. As we shall see in Section 4.3, several design protection schemes rely on the continued secrecy of this encoding with vendors having some commitment to keeping it that way. The obscurity, complexity, and size of the bitstream makes the reverse engineering process difficult and time consuming, though theoretically possible. There are no reports of successful reversal of modern FPGA bitstreams *as defined above* or even a cost estimate that is backed up by data and empirical analysis. The possibility of legal action is certainly an effective deterrent in academic and commercial environments, although for some organizations and in certain countries, these are less of a concern. The incentives of full reversal increase proportionally with the value of the designs bitstream embody and with the popularity of FPGAs. Since there is a rapid growth in both, we can expect much more activity in this area.

Decoding static data from bitstreams — such as RAM and LUT content — is not new (see for example Ziener *et al.* [24], devices' datasheets, and vendors' own tools) and any experienced developer should recognizes this to be the case. The harder part of the process, however, is to automate the process of converting the placelist to a netlist from which the original functional design can be extracted. A topical Internet search yields quite a few hits for projects and services, though most of which seem to be half-baked or stale, with companies providing such a service making claims that are hard to verify. The one exception is "ULogic", a "free software project aimed at netlist recovery from FPGA closed bitstream formats" [113]. Ulogic's developers Note and Rannaud have produced a report [81] in which they describe how their tool can convert Xilinx's bitstreams into placelists. The Xilinx Design Language (XDL) is a largely undocumented plain text representation of placelists (which are otherwise in unintelligeable form) supported by the Xilinx tools from at least ISE version 4.1i. Using the XDL, developers can directly manipulate placed and routed designs, and even bypass the entire software flow to create

XDL representations from scratch. From XDL, the files can be converted back to a placelist and encoded to a bitstream. This allows the iterative process of producing an XDL design, converting it to a placelist and then to a bitstream; by changing single bits at a time, one can create a database that correlates placelist data to bitstream bits. Note and Rannaud have simply automated this process to create a "de-bit" tool in addition to "xdl2bit", which is a bitstream generation tool that is (supposedly) equivalent to Xilinx's "bitgen" (for some device families), but is much faster. The authors rightly assess the security risks to deployed systems due to their development by stating that the step of "making sense" of the data is still missing, namely, the full reversal to a "true netlist".

The eventual goal of the Ulogic project is producing a netlist, and there may be less public efforts with similar aims; the increased value embodied in bitstream will inevitably drive more people and organization to invest time to accomplish automated full reversal. If reverse engineering is a concern, or within reach of a potential adversary, it is probably prudent to no longer rely on bitstream encoding, though it is still unclear what is the actual cost of full reversal. Certainly, hiding keys in look-up tables and RAMs is not a good strategy because it only requires a partial reversal and basic knowledge of the bitstream's construction. In Section 4 we will discuss some solutions that increase the efforts required to be invested by an attacker.

3.2.1 Open formats and hardware

The "open hardware" design community's discussion [55, 89] on FPGAs, and of other open format matters, is quite active and is worth noting in this context. Open hardware advocates are pushing towards a hardware equivalent of the software open source efforts and seeking manufacturers' support. The dominant argument is that open bitstreams and architectures would enable third parties — perhaps open-source — to develop vendor-independent tools that may include support for unique functions and languages otherwise not made available by the vendors' tools. Megacz [73] demonstrates this by the creation of a complete open-source Application Programming Interface (API) for bitstream manipulation for an Atmel FPSLIC FPGA, after its configuration bitstream format was posted [38] on the *comp.arch.fpga* Usenet newsgroup in late 2005. One of Ulogic's stated goals is demonstrating to FPGA vendors that third-party tools can be better than their own, demonstrated by a nimble bitstream encoder. Up to now FPGA vendors have been resisting this approach — with the rare exception of the Xilinx XC6200 FPGA in 1997 — by mostly staying out of the debate. This is firstly, in order to avoid supporting people who create their own bitstreams and use home-grown tools; this argument can be easily dismissed, though discrimination for supporting the hardware based on certain software tools is not going to be easy, and bad for public-relation, so it is not that simple. Secondly, due to fear of competition with their own software tools and loss of control over the use of their hardware products. Lastly, and most importantly, the "openness" will also require revealing proprietary information, including portions of the architecture, which is the edge vendors have over one another and which they cannot afford to lose. But it may simply be that there is no business opportunity there, as the most relevant consumers of FPGAs are large companies who prefer to get the whole package, including support, accountability, and regular updates. Business motives dictate that appeasing the occasional researcher and open-source advocate is not worth the risk to the bottom-line. That said,

in 1998, we saw Xilinx release the JBits API [40], which allowed users to directly manipulate bitstreams. It supported only a few device families and was not very convenient to use, but it marked a step in the direction of openness that would enable the creation of independent tools. JBits was quite extensively used by academic researchers and was updated up to the Virtex-II family, though it seems to have been abandoned since.

3.3 Readback

Readback is the process of retrieving a snapshot of the FPGA's current state while it is still in operation. Upon request, the FPGA sends the snapshot that includes configuration, look-up tables, and memory contents to the host PC, or other device, via the configuration port. This image is different from the original bitstream by missing the header, footer, initialization commands, and no-ops; of course, the dynamic data in LUTs and BRAMs is also different from their initialized state. Readback is a powerful characterization tool in the verification and production testing of the FPGA by the vendors, and also allows the system developer to verify the correctness of the design as it is operating on the FPGA itself.

If enabled, however, an attacker can readback the design, add the missing static header and footer and use it in another device, re-program the FPGA with a modified version, or reverse engineer it. It also enables an active *readback difference attack* where the attacker is able to observe signal changes on an individual clock-cycle basis to bypass defense mechanisms. Consider the case where a functional core is waiting for an enable signal from an authentication process. If the adversary has control over the input clock, he can take a snapshot before the signal is set, clock the design, and then take another snapshot. Through a relatively easy iterative process of comparing the snapshots, the attacker can determine which bits are required to be changed in order to disable the enable signal(s). Then, the original bitstream can be modified to have the enable signal asserted permanently, subverting the defense. In contrast, readback can also be used as a defense mechanism by providing indications of tampering, such as in the case of ionizing radiation attack described in the following section.

Xilinx provides a bitstream bit for disabling readback, but it can be easily found. However, when bitstream encryption is used, multiple, majority-voted, disabling registers are activated within the FPGA to prevent readback [68] [120, User Guide 071]. Lattice devices also disable readback when bitstream encryption is used [64, Tech. Note 1109]. In theory, these disabling bits can be located via invasive attacks, but there is no evidence that this has been accomplished or even attempted. Altera's devices do not have readback capabilities and are therefore not vulnerable to this type of attack.

3.4 Side-channels

Side-channel attacks rely on device-external measurable manifestations of internal processes to deduce secret data or modes of operation by exploiting the implementation rather than the algorithmic construction. The challenge for designers interested in preventing this analysis is the isolation of internal operations of integrated circuits from their environment as they interact with other devices, consume and emanate energy in the form

of electromagnetic and heat radiation. Described below are three types of side-channel attacks and their relevance to FPGAs.

3.4.1 Power analysis attacks

Integrated circuits consume power in two ways. *Dynamic power* consumption is due to CMOS gates changing state while parasitic and load capacitances are charged or discharged according to the logic transition, $0 \rightarrow 1$ or $1 \rightarrow 0$, respectively; this also includes the brief low impedance period caused by both the p- and n-mos transistors simultaneously conducting during a transition. A simple dynamic power consumption model for a CMOS gate is the following:

$$P = C_{\text{load}} \cdot V_{\text{supply}}^2 \cdot f \cdot A$$

where C_{load} is the load capacitance of the gate, which includes wire, parasitic and output capacitance, that need to be charged or discharged with every transition; V_{supply} is the supply voltage to the gate; f is the operating frequency; and A is the probability of a $0 \rightarrow 1$ or $1 \rightarrow 0$ transition. Standaert *et al.* [98] describe a simple experiment that confirms this model on an FPGA. To obtain power trace samples, most of the literature describe measuring the voltage across a low-value resistor placed between either the circuit's power or ground, and the respective external power supply terminals; the exception is Bucci *et al.* [18] who suggest an active sampling circuit to enhance the quality of samples. Shang *et al.* [92] provide a thorough analysis of the dynamic power consumption of 150 nm Xilinx Virtex-II FPGAs, by looking at the power contribution of resources along a signal path. Their results show that about 60% of dynamic power dissipation is due to interconnect routing (the effective capacitance of driven wires), 16% to logic, and 14% to clocking resources.

Static power consumption is the power consumed when the circuit's gates are not changing state. Shang *et al.* estimated that for a 150 nm FPGA, 5–20% of the total consumption is static and is due to gate leakage. Gate leakage has dramatically increased for 90 nm and 65 nm transistors built for high frequency operation, often accounting for a larger portion of total consumption than dynamic power. This leakage is due the decreasing dimensions of the transistors, and the reduced threshold voltage, which cause relatively more current to flow between the source and drain terminals and through the gate oxide. This leakage is also very sensitive to temperature variations, and therefore, not uniform across the die, or in time, and may correspond to switching activity of the circuit. Kim *et al.* [56] provide an excellent introduction to the issues associated with static power consumption. The small dimensions have an effect on dynamic consumption as well since V_{supply} is reduced, the capacitances are smaller, and the interconnects are shorter, resulting in less consumption; albeit, in general, there are more transistors switching as there are more transistors per device. To date, power analysis reports have all ignored static power, and this may no longer be possible in the future and perhaps even assist with obtaining further data-dependent information.

Analysis of the current consumption patterns of an integrated circuit may reveal information about the specific data it is processing, and in most cases the attacker wishes to obtain the key being used in a cryptographic operation. In 1999, Kocher *et al.* [58]

introduced two types of power analysis, *simple* (SPA) and *differential* (DPA). SPA lets the attacker directly search power traces for patterns such as algorithmic sequences, conditional branches, multiplication, exponentiation, and other signatures that allow the inference of key material. DPA compares acquired traces with a statistical power consumption model that is tailored to the target device and specific implementation. This model is based on prior knowledge or analysis of the device which is then enhanced by many recorded samples of controlled operations, for example, by processing known plaintexts with known keys. If the attacker can infer key material based on single bit changes during an encryption, the exact details of the implementation may not be needed. While attacking a device, the model enables the attacker to iteratively guess candidate key bits and obtain statistical correlation between model and measurement. The statistical analysis is required to increase the signal-to-noise ratio such that a candidate guess is distinct from the samples; if noise is present, more samples are required to reach that distinction, but most noise sources and patterns can be modeled such that they can be sufficiently removed. In some cases, even a small number of known bits can make it possible to mount a brute force search for the remainder.

Most power analysis research to date has been done on microprocessors, such as smartcard ICs, for which a model is relatively easy to construct and power traces are simple to obtain due to their sequential and slow operation. Mangard *et al.* [71] provide a comprehensive introduction to power analysis techniques for smartcards. Power analysis of FPGAs has started receiving increased interest since 2003 with Örs *et al.* [83] and Standaert *et al.* [97] being the first to examine the possibility of successful attacks. Örs *et al.* described a power analysis platform for examining Xilinx's 220 nm Virtex device, with a successful SPA attack on an elliptic curve implementation operating in isolation. The research of Standaert *et al.* on a 220 nm Virtex FPGA has shown that SPA is not practical for most paralleled cryptographic implementations when many concurrent operations are running while sharing a single ground. DPA, however, was deemed to be possible, and within a year Standaert and co-authors demonstrated a potentially successful attack based on statistical correlation techniques against an implementation of the Advanced Encryption Standard (AES) [98] and Data Encryption Standard (DES) [99]. The investigation showed that the pipelining of the cipher does not protect against DPA since operations are separated into registered states and are thus better observed in the power traces. However, an unrolled implementation, where each round is implemented on its own for faster throughput (at the expense of more resources being needed), was shown to measurably increase the efforts of a would-be attacker. This is because all encryption/decryption rounds are run concurrently and, with the key unknown, the contribution to the power trace is effectively random noise that cannot be predicted and easily removed during analysis. In practical scenarios the cryptographic operation would be but a small subset of all the concurrent operations on the FPGA, all adding noise to the power trace. However, if these other operations can be determined, they could be masked out; the random nature of the cryptographic process prevents this, and may be useful as a defense. In 2006, Standaert *et al.* [100] analyzed the power signature of isolated pipelined structures on a 180 nm Spartan-II FPGA, improved their previous results from [98], and confirmed some of the results of Shang *et al.*. They also concluded that pre-charging buses with random values to mask transitions, at the expense of resources and throughput, amounts to added difficulty to an attacker, but should not be relied on as a single solution against power analysis.

Simply put, power analysis attacks could be made harder if operations that depend on secret data have the same power signature as ones that do not; achieving this, however, is incredibly challenging and the research community is constantly evaluating and critiquing new and old defense techniques. Standaert *et al.* [101] provide a survey of currently suggested defenses against power analysis attacks, namely, *time randomization*, *noise addition*, *masking*, and *dynamic and differential logic*, with the conclusion that no solution, on its own, can eliminate the susceptibility to power analysis attacks. Messerges [74] also surveys the weaknesses of power analysis countermeasures. One notable countermeasure is Tiri and Verbauwhede [110] who propose an FPGA-specific countermeasure, *wave dynamic differential logic* synthesis. Using differential logic and pre-charging of the gates, this method increases the resistance to DPA by making power dissipation independent of logic transitions with the disadvantage of increasing circuit size and lowering the operating frequency. This would make a power analysis attack very difficult but, in practice, uncontrolled manufacturing inaccuracies inhibit the creation of interconnects that match each other perfectly, and therefore, there will always exist some measurable variability (we will see how these variations are put to good use in Section 5.1 on PUFs). Thus, in a real device, some information will leak, although it may take more samples and more advanced techniques to isolate.

Although the results we have to date are vital to our continued understanding of power analysis, it is evident that more research is due in order to better evaluate the resistance of FPGAs to these attacks. In order to put these attacks in some perspective, below is a list of some of the challenges facing an attacker attempting an attack against modern FPGAs in practice.

- *Familiarity with implementation details.* Most of the attacks described in the literature on FPGAs used some knowledge of specific implementation details to increase the likelihood of success. This will almost universally be untrue when attacking a real system due to the nature of the FPGA's flexibility. It would be interesting to conduct an experiment where a completely unknown cipher implementation is attacked using power analysis. However, as opposed to an ASIC, the attacker has the advantage of having the generic FPGA available to construct an accurate power consumption model.

- *Isolation of target function.* In order for the attacker to obtain a correlation with a model, the noise contributed by concurrent processes must be removed. If these operations are unknown, this will present challenges to the attacker, although noise can be statistically removed given the right model. As a designer, a somewhat costly defense would be to implement an identical cryptographic function operating in parallel to inject, what is effectively, random noise.

- *Obtaining high signal-to-noise ratio samples.* With today's FPGAs operating at over 500 MHz, the required measurement equipment is not trivial, and more advanced techniques than the traditional small resistor are required. The attacker must also isolate the signal from the FPGA from the surrounding devices that contribute noise through the shared ground and power supply. Countermeasures may be a detection circuit for clock and temperature tampering, not allowing the attacker to tamper with the clock's frequency.

- *Probe BGA packages on dense multilayer circuit boards.* All high-end FPGAs — with low-end ones quickly following suit — have a flip-chip Ball Grid Array (BGA) packaging, the largest having nearly 2 000 balls, that physically prevent easy access to pins while the device is still soldered onto the board. BGA packages necessitate that traces be routed inside internal layers, using blind/buried vias, which increases the entry cost for an attack. Even if the unencrypted bitstream is copied to another system for analysis, that system still needs to be developed. Relatively cheap mechanical and electrical mechanisms can be added to the printed circuit design to make an attack increasingly more expensive; for example, sensitive signals between devices can be routed in internal printed circuit layers, perhaps sandwiched between sensor mesh layers.

Finally, the attacker will need to deal with devices manufactured at 90 and 65 nm technologies, the vulnerability of which to power analysis is still to be investigated. Smartcards have a simple and standardized interface and can be isolated, making it simple to launch attacks using "kits" and readily available equipment. In contrast, each FPGA-based system interfaces with the FPGA differently and in a much more complicated way. This is a major difference between smartcards and FPGAs, where the former has a simple and standardized interface. We may be able to conclude that would-be attackers must overcome considerable challenges before being able to collect power traces, and then even more trying to analyze them.

3.4.2 Electromagnetic emanation analysis

This side-channel attack relies on circuits producing electromagnetic fields due to the movement of charge during the execution of internal operations. These fields can be picked up outside of the device using carefully tuned antennas, even without removing its packaging. *Compromising emanations* were known to military organizations since at least the 1960s, and have been used in electronic warfare since; Kuhn [60] provides the history and evolution of such attacks along with practical experiments and results from eavesdropping on computer displays.

Applying electromagnetic analysis (EMA) attacks on integrated circuits has only started to receive attention from the research community since the late 1990s. In the rump session of Eurocrypt 2000 Quisquater and Samyde introduced the terms *simple* and *differential* electromagnetic attacks, SEMA and DEMA respectively, as the EM analysis equivalents to power consumption analysis. A paper then followed describing their techniques and initial results analyzing microcontrollers [88]. At about the same time, Gandolfi *et al.* [35] demonstrated EM analysis on three cryptographic implementations in microcontrollers. Their results show that if set-up correctly, EMA attacks can be more efficient and produce better signal-to-noise ratios than their power analysis counterparts. In a comprehensive analysis Agrawal *et al.* [3, 4] analyze smartcards and observe that there are two kinds of emanations, *direct*, which are caused by current flowing along interconnect, and *unintended*, caused by electrical and magnetic coupling between wires and components. The authors were able to exploit these emanations to obtain better results than their application of power analysis. There is an inherent advantage to electromagnetic attacks over power analysis in that they can be localized to a particular portion of the chip where the activity of interest takes place and can be mounted in the device's original setting.

Carlier *et al.* [19] have reported the first EM analysis of an AES implementation on a 130 nm Altera Cyclone FPGA. Their *square electromagnetic attack* is based on the square attack [23] which is more efficient than brute force for six rounds or less of AES. This *chosen plaintext* attack fixes all but one byte of the input and observes the propagation of this byte's bits throughout the round functions. The authors were successful in obtaining some key bits by placing an antenna close to the FPGA and using DEMA techniques; they were also able to distinguish relevant signals from the noise produced by parallel processes.

De Mulder *et al.* [26] have reported a successful attack against a special implementation of an elliptic curve algorithm on a 220 nm Xilinx Virtex 800 FPGA. They used SEMA to observe key-dependent conditional branching, and DEMA statistical techniques against an improved algorithmic implementation. A later publication by De Mulder *et al.* [27] has more technical details on the DEMA attack. It is interesting to note that localization considerations were taken into account and that the FPGA was operating at a very low frequency of 300 kHz. As with the power analysis reports, these implementations ran in isolation, making the attack environment ideal for the attacker.

Most of the suggested techniques to prevent the success of EMA attacks must be designed into the device by the manufacturers, for example, lower power consumption resulting in reduced emissions, encompassing meshes, and other architectural solutions [88]. In the absence of these mechanisms, the system developer can distribute the sensitive processing elements across the FPGA's fabric to avoid localization.

In summary, EM analysis is certainly a side-channel that can be exploited by attackers, though many of the difficulties outlined for power analysis also apply here and require further research. As Argrawal *et al.* [5] and others demonstrate, the combination of power, electromagnetic, and other side-channel analysis may be the best way to improve results. An interesting aspect that may affect both attacks, and is yet to be explored, is the distribution of ground and power pins in the package. We discussed down-facing die in flip-chip packaging, but the arrangement of power and ground pins has also been radically changed to gain better signal integrity. Traditionally, ground pins were at the center of the package with power pins in batches around this center cluster. Package and chip designers now tend to spread power pins across the grid array, and closer to signal pins, for less inductive and shorter ground returns. It would be interesting to see if this makes acquiring good samples, perhaps locally to where the target function is, better or worse.

A final note on metrics. Both power and electromagnetic analysts use the number of samples required to obtain key bits as a success metric. The difference between 10 000 samples and 100 000, for example, is not very meaningful if no indication is given regarding the cost difference between them (*i.e.*, how much the attacker must spend); for the system designer, however, the difference between a handful and 10 000 samples is meaningful on its own in two ways. Firstly, evaluating how results improve in experiments where the equipment and target device is the same, with the only difference being the signal processing or measurement techniques, and secondly, when it is used to determine the intervals between the re-keying of the cryptographic process, or how many invocations of the cipher with the same key to allow before halting.

3.4.3 Timing analysis

Each data-processing operation takes a certain time to complete and if that operation depends on secret material such as a key, some information may leak. Conditional branching, memory access, and algorithmic operations, for example, are often key-dependent in cryptographic function implementations; analyzing their timing signatures can provide sufficient knowledge of key bits. A common example of timing attack is the checking of passwords, where they are verified a character at a time, and once an incorrect character has been entered no further comparisons are made, resulting in a different processing time. This way, the attacker can know which characters are correct without completing the password entry. Kocher [57], who introduced this class of attacks in 1996, and Dhem *et al.* [28] have shown how practical these attacks are against microcontroller implementations of cryptographic algorithms.

Observing timing variations through the power traces might not be as effective with FPGAs because, unlike microcontrollers, processes run concurrently. However, timing can be observed through memory accesses and other interfaces with external devices. The designer can prevent information leaking through timing variations by making sure that sensitive operations take constant time, *i.e.*, same number of clock cycles; by adding timing randomization to operations; or, by loading data into internal memory blocks before processing them. In general, any operation that is observable via the device's pins should be checked for timing-related data leakage.

3.5 Ionizing radiation

Single Event Upsets (SEU) are "radiation-induced errors in microelectronic circuits caused when charged particles lose energy by ionizing the medium through which they pass, leaving behind a wake of electron-hole pairs" [78]. SEUs in CMOS devices are generated by atmospheric, ambient, ionizing radiation consisting of neutrons, protons and heavy ions and also from alpha particles emitted from materials used for integrated circuit packaging [51, 80] [120, White Paper 208]. An SEU may cause a transient pulse called a *single transient effect* resulting in delay faults [34] and may also cause a memory bit to flip state; multi-bit upsets due to a single event are also possible with decreasing probabilities. These flips are called *soft errors* because they can be corrected by overwriting or power-cycling.

In FPGAs, the result of a flip in a *used* configuration cell is a change to the functionality of the device. Lesea *et al.* [69] provide experimental results from a long running study to validate extrapolated Mean Time Between Failure (MTBF) estimates from accelerated particle beams. This was done by placing hundreds of unconfigured FPGAs, manufactured with different technology geometries, exposed to ambient radiation at four different altitudes. The true MTBF was measured over periods longer than a year, and was shown to be higher than many previous predictions for small feature size transistors. However, the mean time between *functional* failure can be at least ten times that rate. This is because most of the FPGA's fabric is routing, and even if there is a 100% utilization of resources, only up to 10% of the whole FPGA is actually used. Of course, flipped bits in user logic, such as ROM content, may also result in faulty operation. As an adversarial tool, one can use ionizing radiation to modify the configuration of the device to disable

protection mechanisms or alter memory content. This attack could be made successful by exhaustively irradiating the device until the desired results are observed; if one is able to focus the radiation and accurately apply it, once the location of the relevant registers is known, the cost of attacking other systems will be lower. However, given the amount of transistors in a device and the cost associated with precise irradiation, this attack might not be practical except to well funded outfits. We should be aware that SRAM cells for memory and FPGA configuration are of different construction than ones found in memory and ASICs. While SRAM cells are built for speed and have minimal loading, FPGA configuration memory is allowed to be slow and thus, may have larger capacitive loads, making this type of SRAM cell harder to flip [48, 69].

Measures for detection (and correction) of SEUs were introduced by FPGA vendors for high-reliability applications [8, App. Note 357] [120, App. Note 714]. These functions continuously scan the configuration cells and compare their CRC or Hamming syndrome to the original's, alerting on discrepancy. Triple Modular Redundancy (TMR) is another solution, where all logic is triplicated and majority voters inserted to determine logic faults due to radiation. This is used mostly in space applications where the mean time between function failure is so low that the cost is justified. Along with TMR, these applications also "scrub" the content of the FPGA every so often to restore the correct state of the cell that flipped due to SEUs. Although not originally intended as security-enhancing measures, these solutions may be used to detect radiation attacks, which may also be applied to tamper proof modules which are not built to resist them. Aside from the internal checks, readback could be used to send the read-out bitstream to an external radiation hardened device to verify it using stored copy of the original. Alternatively, if the bitstream is authenticated, it could be read-back and used to reprogram the FPGA.

3.6 Invasive and semi-invasive attacks

Invasive attacks physically probe and alter the target device in order to extract secret information from it. The process involves de-packaging of the device and removing the passivation layer that protects the metal interconnects from oxidation. This can be done using chemicals, or for more precision, with a laser cutter that creates a hole for the insertion of probes. This requires a microprobing station that allows the attacker to accurately control the probes, position the die, and observe it with a microscope. A more sophisticated and expensive tool required for small feature size integrated circuits and higher precision is the Focused Ion Beams (FIB) workstation. Using accelerated particle beams that interact with gases close to the die, the FIB can create incisions at the nanometer scale, deposit metal connection, and take high resolution images of the die. This can enable an attacker to either read data from buses or disable certain structures. Skorobogatov [94] details the process of invasive attacks on microcontrollers while Soden *et al.* [96] provide an excellent survey of techniques used for failure analysis by device manufactures, which are the same ones that may be used by attackers for analysis and attack.

The shrinking feature sizes[7], integrated circuit complexity, and the destructive nature of invasive attacks make them very costly. Further, the advent of flip-chip packaging used

[7] In 2006, FPGAs manufactured in 65 nm and made of 12 metal layers were introduced.

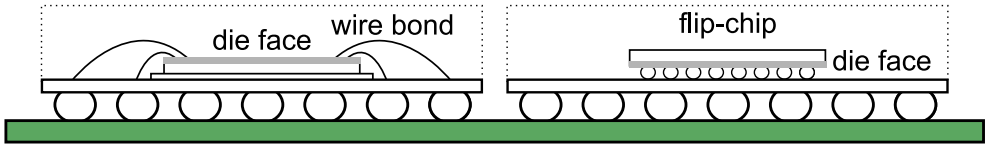

Figure 4: A wire-bond package is shown on the left where the die faces up with wires bonded connecting it to the the solder balls through a substrate; in the flip-chip package on the right the doun-facing die has "bumps" used to connect it to the circuit board substrate using solder balls.

for many of today's FPGAs, and shown in Figure 4, prevent easy access to the face of the die. Flip-chip packages mount the die facing down, close to the package's pins to reduce inductance and allow greater package densities, or smaller pitch. The older wire-bond packages had the die facing up and wires attached to die in order to connect to the package's pins, and thus were easier to probe. Currently, there are no published reports on a successful invasive attack against volatile FPGAs.

Semi-invasive attacks were introduced by Skorobogatov [94] and are a class of attacks that cover the gap between the non-invasive (covered next) and invasive types. These attacks require the removal of the device's packaging, leaving the passivation layer intact, while the rest of the analysis is done using techniques such as imaging and thermal analysis. Methods under this category are cheaper than invasive attacks since they typically do not require expensive equipment or extensive knowledge of the chip; Skorobogatov has applied these attacks on devices fabricated with 250 nm and larger manufacturing technologies. As with invasive attacks, no reports are available on successful applications of these techniques on recent FPGAs.

One relevant semi-invasive techniques is *data remanence*: the effect resulting in the retention of evidence of previously stored state in storage media or RAM cells after they have lost power. Ionic contamination, hot-carrier effects, and electromigration can "impress" the stored state over time. Gutmann [41, 42] covers remanence of magnetic media and RAM at length. For RAM, under certain conditions, such as high voltages or lower temperatures, the previous state is available for a period of time after power is removed. Skorobogatov [95] tested eight (manufactured at 250 nm technology and larger) feature-size SRAM devices, all showing remanence of up to a few seconds at sufficiently low temperatures. No public data is available for remanence analysis of FPGA memory cells and since these have different characteristics, prior analysis of other types of cells may not apply.

Both invasive and semi-invasive techniques remain an interesting research topic that will undoubtedly yield new insights for dealing with the shrinking feature sizes of modern FPGAs and integrated circuits in general.

3.7 Brute force, crippling, and fault injection

In cryptography, brute force means attempting all possible key values to search for a valid output. It can also mean exhaustion of all possible logic inputs to a device in order, for example, to make a finite state machine reach an undefined state or discover the combina-

tion to enter the device's "test mode". Another form of brute force attack is the gradual variation of the voltage input and other environmental conditions, rather than variation of logic states. Brute force is sometimes associated with *black-box attacks* that attempt to exhaust all input combinations and record the outputs in order to reverse engineer the device's complete operation, or create a new design that mimics it. Considering the stored state, complexity, and size of current FPGAs, this type of attack is not likely to be practical or economic for reverse engineering the FPGA's whole functionality [119]. That said, if a subset of the functionality is targeted that can be interfaced with directly through the IOs, brute forcing can be fruitful, perhaps in combination with other attacks. For critical functions, therefore, randomization may be useful. Christiansen [20] suggests adding *decoy circuits* to the design to increase the effort of an attacker. The cost is high, though, seven times LUT usage and twice the power in addition to requiring more IOs and design time. Certainly, this would not be appealing to cost-conscious designers who try to fit the design into the smallest FPGA possible.

Crippling attacks either subvert a system to perform malicious functions or completely bring it off-line, similar to *denial-of-service attacks* on networked servers and devices. The absence of robust integrity preserving mechanisms for bitstreams, such as authentication, enables anyone to program an FPGA if they have access to it. In the case where bitstream encryption is used (see Section 4.2) confidentiality is provided, but may not be sufficient, as an adversary can still re-program the device with an invalid bitstream and bring the system off-line. An extreme scenario is described by Hadžić *et al.* [43] where an FPGA is permanently damaged due to induced contention by using invalid bitstreams. Their attack may not work in practice, however, because most systems cannot supply the necessary current for sufficiently long to damage the device, unless they are connected to a non-current-limited power supply. In any case, bitstream authentication, discussed in Section 5.2, will solve all these issues, except the one where the FPGA is left unconfigured, because it can only store a single configuration at a time. Of course, if the attacker has physical access to the system, nothing could be done to prevent denial-of-service, aside from physical measures.

Fault injection or *glitch attacks* attacks attempt to force a device to execute an incorrect operation, or cause it to be left in an unintended or undefined state that can be exploited, or that leaks some secret information. Techniques include altering the input clock, or creating momentary over- or under-shoots to the supplied voltage. For example, if a conditional branch is skipped by the CPU due to a clock glitch, security mechanisms could be subverted; a voltage surge or trough can cause registers to keep their state for a similar outcome. For example, if a power glitch is applied at the right time, the number of rounds of an encryption algorithm may be reduced; Anderson and Kuhn [11] demonstrate how glitches and fault injections were used to attack microcontrollers. The best defense against these attacks is making sure all states are defined and at the implementation level, verifying that glitches cannot affect the order of operations. Other defenses are clock supervisory circuits to detect glitches, and detection of voltage tampering from within the device. Some of these solutions are available, and are discussed in Section 6.3.

3.8 Relay and replay attacks

Relay attacks allow an adversary to impersonate a participant during an authentication protocol by extending the intended, or assumed, transmission range for which the system was designed. Relay attacks[8] have been known since at least 1976 [22, p75] and are simple to execute as the adversary does not need to know the details of the protocol or break the underlying cryptography. A good example is a relay attack on proximity door-access cards that was demonstrated by Hancke [45]. To gain access to a locked door, the adversary simply relays the challenges from the door to an authorized card, some distance away, and sends the responses back. The only restriction on the attacker is that the signals arrive at the door and remote card within the allotted time, which Hancke showed to be sufficiently liberal. Another example, is the joint work of the author with Steven Murdoch [31] on relaying smartcard payment transactions to place unauthorized transactions on a victim's card. It is evident, then, that despite the knowledge of existence of of such attacks, we regularly see systems susceptible to them being deployed.

In the context of FPGAs, consider a router that employs an authentication protocol with a chip that is placed near it on the PCB (similar to the schemes that will be described in Section 4.3). The design of this router is being cloned, and the unmodified bitstream is loaded onto the counterfeit system's FPGA. Since the proper execution of the bitstream's functions require an authentication protocol to be completed with a device with which it shares a secret, it is assumed that since that device is no longer present on the board, it will not work. However, if the attacker is able to relay the challenge to a remote site where that device is present, the authentication will succeed, and the protection subverted.

It is unlikely in today's conditions that cloners would resort to such elaborate relay attacks, but this might change when more anti-cloning mechanisms are used. In general, every security system should consider the possibility of an attacker being able to relay challenges and responses for a range beyond the one assumed, and proper timing constraints should be applied. A cryptographic solution can be in the form of *distance bounding*, where the FPGA can determine the distance of another device by a mutual exchange of multiple single-bit challenge-response pairs and placing an upper bound on the distance by assuming the signals traveled at the speed of light. The author and Steven Murdoch [31] have implemented a wired version of the Hancke-Kuhn distance bounding protocol [46] on an FPGA as a defense against relay attacks, being able to bound the distance between participants to below a few meters. This may be a low-cost defense against relay attacks when devices supporting the protocol, such as FPGAs, are communicating, or even used as a cheap anti-tampering defense.

Replay attacks allow the attacker to resend recorded protocol transaction data at a later time. The purpose, for example, can be to repeat a money transfer transaction or impersonation of a participant in an authentication protocol; Syverson [105] provides a taxonomy of replay attacks. Cloning of FPGA bitstreams is the simplest replay attack. Consider, however, the case where a critical security flaw has been found in a fielded design and a field-reconfiguration is performed with an updated bitstream. If the attacker was able to record the bitstream containing the vulnerable design, and even if it is encrypted, he would still be able to program the FPGA with it. This is possible because the FPGA cannot verify the *freshness* of the bitstream; provide freshness of its own to the crypto-

[8] Also called *man-in-the-middle* or *wormhole* attacks, depending on context.

graphic process; and, has no record of past events and thus, will accept old and revoked bitstreams. One solution can be the addition of a non-volatile counter that is added to the encryption or authentication process to provide freshness; this value is called a *nonce*, which can be random, pseudo-random or even a predictable string, but must strictly be used only once. Another, more complicated solution that does not require additional FPGA features and at the expense of user logic, is for the design to send an authenticated message to some authority, which attests that the operating version up-to-date.

3.9 Social engineering

Social engineering or *pretexting* is the practice of manipulating people into revealing secrets through any form of human-to-human interaction; phones are mostly used because of the relative low risk and the ease in which one can pretend to be someone else. When a fraudster can hand a $100 note to a cleaning staff member for some inside information or cleverly manipulate the CEO's secretary to email the information to him from within the company's network, the type of FPGA, cryptographic algorithm or any other technology-based defense becomes irrelevant. The attacker can also penetrate offices under the guise of a technician and install a sniffing device on the port or cable used to program FPGAs, not only getting the designs, but also the keys used to encrypt them. In general, social engineering attacks are significantly cheaper than defeating active protection mechanisms. Simply restricting access for sensitive areas of the company's network or files to particular individuals with proper credentials can help, along with a robust audit trail of access to information. Education, rather than technology, is the more effective way to minimize the damage from social engineering due to the inherent human element; Mitnick [75] details his exploits in this area and also suggests ways for companies to mitigate losses due to social-engineering. One must also consider the corrupt, disgruntled and/or underpaid employee who is seeking fame, fortune, vengeance, or all three. These can be hired by competitors and walk away with secrets in their handbag or they may install a *Trojan horse*, *logic bomb*, or a *back-door* in corporate files or software code. Phishing and intrusion to databases is also a concern (think about modification of the HDL of cores offered free to customers). Then, of course, there are the more traditional crimes of laptop theft, break-ins, remote access at conferences and so on.

Social engineering class of attacks applies to any type of information and is mentioned here for completeness as system developers must be aware of the social threats in addition to the technical ones. Measures to thwart these attacks and minimize their effect should be an integral part of every defense strategy and certainly part of the threat and cost analysis.

4 Defenses

In the previous section we discussed the vulnerabilities and threats to which FPGAs are prone, now we consider further means by which designers can protect their products.

4.1 Defense categories

The efficacy of a defense mechanism is evaluated by the cost of circumventing it; skill, tools, and time required for breaking the defense are monetized to give the analyst a metric that indicates the system's estimated level of security. Before delving into the technical discussion, let us first define defense categories that would assist us in the discussion that follows:

- *Social* deterrents are provided by legal systems and also rely on peoples' good social conduct and aversion from being prosecuted and incarcerated. Designs are, sometimes by default, protected by trademarks, copyrights, trade secrets, patents, contracts, licensing agreements and, of course, by the catch-all concept of "intellectual property"[9] (IP). However, social deterrents are only effective where appropriate laws exist and are enforced. Attitudes towards design-ownership rights vary significantly from country to country, making this type of deterrent not wholly effective where it would matters the most, *i.e.* in countries where most counterfeit goods are manufactured and in which these rights tend not to be respected.

- *Active* deterrents are mechanical and data-processing mechanisms that attempt to actively prevent the theft and misuse of designs; for example, encryption, authentication, tamper proofing, *etc.* Active protection is highly effective if implemented correctly, and is also locale-free (assuming we ignore export restrictions laws preventing distribution of cryptographic "devices" to certain countries). Further, combined with the social deterrents, active deterrents assist in convincing a court that the designer has taken the appropriate measures to protect the design and that the perpetrator had to actively circumvent them.

- *Reactive* deterrents provide detection or evidence of intrusion and fraud that may help in applying the social tools that are available. Digital forensics rely on these mechanisms, such as closed-circuit TV, watermarking, fingerprinting, and steganography, to initiate further investigation or improve the security of a system after an attack or intrusion. *Audit trails* are a reactive mechanism, and are an important facet of security in the absence of, or in addition to, active ones. Reactive measures do not actively prevent fraud or theft, but their presence may deter would-be attackers, and it is sometimes beneficial to advertise them.

4.2 Bitstream encryption

Encryption is a reversible function that provides confidentiality to data and depends on the secrecy of a key such that even if the encryption algorithm is known, the reversal of the process to arrive to the correct original input is not possible. Encrypting the bitstream at the end of the design flow and decrypting it within the FPGA defends against cloning, reverse engineering, and in some cases, provides limited tampering protection. The basic encryption of configuration data for programmable devices was first suggested in 1992 in a patent by Austin [14]. The first implementation is due to Actel's 60RS device family,

[9]This term is now so loaded that it has become somewhat meaningless; in this paper, other, more descriptive terms, will be used instead.

though it soon became an example-case for bad key distribution practices since they permanently embedded the same vendor-defined key inside all devices (this protected against reverse engineering rather than cloning) [52]. This meant that a single successful attack would allow decryption of all bitstreams, so the incentive to obtain the key was high — and since the key had to be present in the software, the attacker only needed to reverse engineer the code, rather than resort to invasive attacks. In 2000 Xilinx implemented a bitstream encryption mechanism in their Virtex-II family which allowed a user-defined key, though by now this functionality is common in most high-end FPGAs, and works as follows.

After the bitstream has been produced, the software requests a key from the user and encrypts the configuration payload of the bitstream. The user then "programs" this same key into the FPGA, which has a dedicated hard-wired decryptor in its configuration core. The bitstream has header information that instructs the FPGA to pass the data through the decryptor before it is sent to the configuration memory cells, as usual. An attacker who obtained the encrypted bitstream cannot use it because he does not have the correct key; thus, he can neither reverse engineer it or use it in another device. Altera's Stratix II and III FPGAs also allow the designer to force all bitstreams through the decryptor, disallowing unencrypted bitstreams. This prevents the "execution" of any bitstream not encrypted with the correct key, but does not prevent denial-of-service attack by attempting to program the FPGA with an invalid one.

4.2.1 Key storage

Keys must be present inside of the device at the time of bitstream decryption, and there are two key-storage techniques used today: volatile and non-volatile. Using volatile storage, keys are kept in low-current SRAM memory cells and are powered by a battery attached to a dedicated pin when the FPGA is not on. The advantage of this storage method are that it allows zeroization of the keys by mechanical means in case of a breach when the device is not powered. Any attacker must maintain the power at all times, making semi-invasive and invasive attacks especially complicated. These attributes appeal mostly to security-conscious designers as it conforms to the US government's FIPS 140-2 [79] and provides a high level of security. The disadvantages are the requirement of an additional component, the battery, and other concerns of reliability, service costs in case of failure, and PCB real-estate (especially so if a battery holder is needed to facilitate replacement). As discussed in Section 2.1, all of these make it less appealing to the cost-conscious designers who may not necessarily need the security level that this method provides, and prefer to have a lower-cost solution.

Using non-volatile storage, keys are permanently embedded in the device using fuses, laser programming, Flash, or EEPROM. Fuses have long been used with redundant circuits to improve yield, for "tuning", or disabling of circuits. However, integrating non-volatile memory with the latest CMOS technology is a challenge as they introduce a non-standard manufacturing step that impacts cost, yield, and reliability. This may be the reason why FPGA vendors have only recently started offering non-volatile key storage. Embedded keys have the advantage of not requiring external devices and incorporate the cost of this functionality into the price of the FPGA, which has a much greater appeal to cost-conscious designers over the battery solution. Embedded keys can also assist against

run-on-fraud as the keys may be programmed into the device at a trusted facility before being shipped to a third party for system assembly and testing.

Xilinx provides volatile key storage with Triple-DES and AES-256 encryption for Virtex-II/PRO and Virtex-4/5 device families, respectively [120, App. Note 766]. If encryption is used, readback and partial configuration are disabled by multiple internal registers, independently of the bitstream bit-settings [68] [120, User Guide 191]. However, the facility of an Internal Configuration Access Port (ICAP) is still enabled, allowing internal readout of the configuration content and sending it out through a user IO. Therefore, when encryption is used, designers should be mindful of a Trojan horse attack, perhaps by a malicious core vendor. Altera's Stratix II FPGAs have non-volatile key storage for AES-128 encryption, but require external components on the printed circuit for producing the voltages necessary for programming the keys if this is done in-circuit [8, Handbook SII51003, App. Note 341 v2.1]. With Stratix III, Altera provides JTAG-programmable battery-backed volatile, and non-volatile, key storage for bitstream encryption using AES-256, catering to both types of security consumers [8, Handbook SIII5V1 and White Paper 01010]. The scheme requires two keys, one encrypting the other to produce a "real key" which is used to encrypt the bitstream. In the FPGA, the real key is obfuscated using a proprietary scrambling function and then stored in distributed cells to make it harder for the invasive or semi-invasive attacker to obtain it. When the encrypted bitstream is sent to the FPGA, an inverse scrambling function is used to produce the real key for decryption. It is not immediately clear what is gained in terms of security from the two-key scheme as it is described. Lattice provides non-volatile key storage using AES-128 encryption in the ECPM2/M family of FPGAs [64, Tech. Note 1109].

4.2.2 Key management

NIST's *Recommendation for key management* [79, FIPS 800-57], remarks that "key management is often an afterthought in the cryptographic development process. As a result, cryptographic subsystems too often fail to support the key management functionality and protocols that are necessary to provide adequate security... key management planning should begin during the initial conceptual/development stages of the cryptographic development lifecycle". Although it is one of the most important aspects of security engineering, one rarely sees it discussed as part of a defense strategy for rights protection and design distribution.

Key management is the process of key generation and distribution (transport) for and to communicating parties, either prior or after initiating this communication. As the security of the system should rely on the secrecy of keys, their management infrastructure is as important as the choice of cryptographic algorithm, protocol, and their implementation. The logistics of keeping keys secret while stored, transported, and updated, along with adequate access controls amounts to a non-trivial cost that must be incorporated into the overall cost of a defense strategy; at the least, procedures for key management should be defined. It is also quite possible that the total cost of a defense strategy could exceed the loss due to theft, when these are evaluated.

As a basic example, let us consider key management for bitstream encryption. As a first step, the value of keys needs to be established. If a single key is used to encrypt all instances of bitstreams, the costs are low, but the result may be a catastrophe, with a

single compromise undermining the security of all fielded systems (as we have seen with the 60RS device). On the other hand, if each bitstream is encrypted with a unique key, the logistical costs are higher, but a compromise is localized to a single system. A solution can lie between these extremes and decided based on the level of trust the system developer has with other principals and also by the risk of loss due to compromised keys. Placing a value on keys also determines how often they are to be replaced, how widely distributed, and guarded. The next step is to determine the key and encrypt the bitstream. The system developer must first identify a set of trusted engineers to handle keys, and also decide if he trusts the vendor's software not to leak the keys[10]. Then, "trusted" computers in a secure environment should do the encryption[11] and the programming of the keys onto the FPGA. The database of keys should be physically protected and audited, and be restricted to engineers with the right credentials. All efforts should be made to detect the compromise of keys, and mechanisms put in place to revoke them. More elaborate schemes for distribution will be discussed in Section 5.3, ones that will require even greater overhead for key management.

4.2.3 Problems with encryption

Despite encryption, and due to a lack of cryptographic-level data integrity the attacker would still be able to construct a bitstream that is not identical to the original, yet is still valid. Block-ciphers can operate in several *modes*, which describe the various ways in which messages greater than their respective block size n (in bits) are processed [79, Pub. 800-38A] (some modes allow block ciphers to operate as stream ciphers). Each of these modes has advantages and weaknesses depending on the application. For example, the Electronic Code-Book (ECB) mode encrypts each block independent of any other, resulting in identical plaintexts being encrypted to the same ciphertexts. This is considered a weakness for messages with repetitive blocks — as FPGA bitstream are — because the attacker can observe that certain blocks are the same, or even interchange them. Where messages are less or equal to n, and that contain random elements such as nonces, this is less of an problem. The Cipher Block Chaining (CBC) mode is used to prevent the repetitions of ECB by starting with the encryption of a random Initialization Vector (IV) and having each block depend on its predecessor's resulting ciphertext. CBC is *self recovering* in that it can recover from bit transmission errors within two blocks with an average of $(n/2) + m$ incorrect bits: $n/2$ in the plaintext block corresponding to the block with the corrupt bits and m bits in the subsequent plaintext block, while all other remain unchanged. Thus, and since encryption does not necessarily detect tampering, an adversary can toggle m bits in one block of an encrypted bitstream and affect the design without knowing the key. Of course, the block preceding the one with the toggled bits will be corrupt but may not contain data that is relevant to the attacked portion of the design, for example, RAM content. This attack can be fruitful, for example, if the goal is to find a collection of bits that when toggled, disable a defense mechanism. Cipher modes, therefore, need to be chosen carefully per the security goals of the application.

[10] This concern can be alleviated by encrypting the bitstream's payload with a custom written program, assuming that the encryption algorithm is available.

[11] For example, ones that have never been on a network and are "fresh". For a complete discussion see Anderson [10].

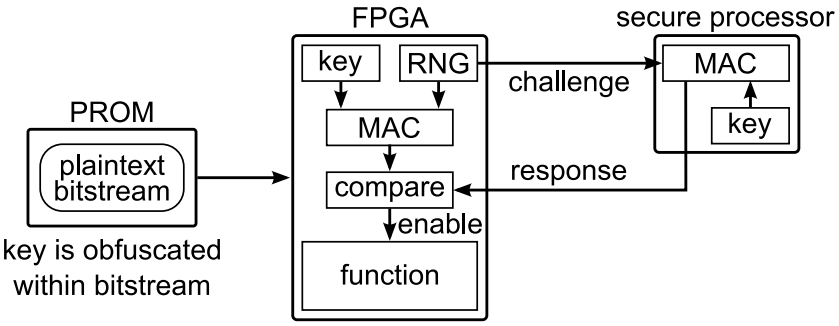

Figure 5: A challenge-response scheme used as a cloning deterrent where a shared key is stored in an external non-volatile processor, and also obfuscated in the bitstream. When this bitstream is loaded onto the FPGA, a random challenge is produced internally and is sent to the processor, which responds with a MAC of this challenge. The FPGA performs the same operation and the MACs are compared; an enable signal is produced if there is a match.

The Virtex family of devices uses CBC [120, User Guide 071] while Altera and Lattice do not specify their block cipher's mode of operation [8, Handbook SII51003] [64, Tech. Note 1109]. FPGAs can detect spurious bit flips that are due to noisy transmission by using linear checks, and the attacker may need to overcome that in order to succeed in the attack above. This will be discussed in Section 5.2 on bitstream authentication.

4.3 Design theft deterrents

FPGA vendors offer a few cloning deterrents that rely on the secrecy of the bitstream's encoding to obfuscate secrets. Indeed, they are not cryptographically secure but may increase the cost of cloning sufficiently to be useful for cost-conscious designers. These solutions are suitable for the low-end devices that do not have the bitstream encryption capability.

One of the first FPGA bitstream theft deterrent proposals is due to Kessner [54] who, in 2000, suggested a CPLD send a keyed LFSR stream to the FPGA for comparison to an identical computation locally to verify that it is mounted on the right circuit board; now out-dated, it is still an interesting read for a perspective on the state-of-the-art back then. More recently, both Altera and Xilinx proposed very similar challenge-response schemes [8, White Paper M2DSGN] [120, App. Note 780] as cloning deterrents and they are outlined in Figure 5. The FPGA shares a key with a non-volatile (NV) processor placed beside it. The FPGA sends a challenge, produced by a true random number generator, to the NV device and both perform a key-based computation on it (encryption or keyed-MAC, for example). The NV device sends the outcome back to the FPGA where a comparison is made; if there is a match, an enable signal is asserted to the rest of the logic. The result is that the design is verifying that the FPGA it is operating on is mounted on the expected printed circuit board. While implementing these schemes we must be careful to make sure the random number generator is not easily influenced by temperature and voltage variations, as well as considering readback difference, relay, and replay attacks.

In the Spartan-3A device family, Xilinx offers a "Device DNA", which is a non-volatile,

factory-set, user-logic accessible, 55-bit unique serial number [120, User Guide 332, p241]. Xilinx suggests using this number as a "key" for design verification schemes using an external device in a similar, or even less secure, ways to the ones described above. Since the serial number is not a secret — it can be easily read out by anyone — it is suggested that the attacker's challenge would be to discover the algorithm used for processing this number.

These schemes rely on the continued secrecy of the bitstream's encoding for security, which can provide an incremental challenge to would-be attackers when active measures are not available. We discussed the issues of bitstream reversal in Section 4.3; unfortunately, we cannot quantify how much protection these schemes buy us.

4.4 Watermarking and fingerprinting

A *digital watermark* is a reactive mechanism that allows the owner of a file to place a hard to remove proof of ownership, which is interwoven and transparent to an observer; it typically ties the file or work to a particular author, source, or tool. *Fingerprinting* is a watermark which is used to identify specific end-users for which the design or work is intended. Watermarks are traditionally encoded into human-inaudible frequencies of audio files and in the least significant bits of images, where they are below the sensitivity of human vision. The content, therefore, is altered but goes un-noticed.

Khang *et al.* [50] provide the fundamentals of watermarking techniques for integrated circuits and FPGA design marking; Abdel-Hamid *et al.* [1] provide a good survey for general watermarking techniques, and their merits. Ideally, watermarks should be easy to insert into the development flow; not affect the correct functionality, which is different from the traditional use; be low on resources; be unforgable; and, be sufficiently robust as proof of theft or fraud.

Watermarks can be inserted at three different stages of the FPGA design flow: HDL, netlist, or bitstream. An HDL watermark can be used if the designer is marking his own design, but he needs to be careful that the mark is preserved throughout the flow; if the watermarked HDL was supposed to protect another principal's work it would not be effective because it can be easily removed. At the netlist level, the mark can be removed with some difficulty, so it should be delivered encrypted, also making sure that the mark is not optimized away or removed at later stages. Bitstream-level insertion is available only for the system designer because otherwise it would require post-processing by the core vendor, which would be complicated given that he does not know how the core was integrated.

Lach *et al.* [62] were the first to address FPGA bitstream watermarking and fingerprinting for design protection. They suggested doing so by embedding content or functions into unused LUTs (*additive-based* marking). To avoid an *elimination attack* by colluding recipients of the same design with differing marks, they also suggest varying their placement for every instance. A simple attack against this scheme would be to remove LUT content from unencrypted bitstreams while confirming that the design still works correctly, iteratively removing identifying marks; *masking attacks* are also possible by inserting random content into all LUTs that are not used. Lach *et al.* [63] later improve their technique by splitting the watermark into smaller pieces such that the above attack

is made harder. Building on the *constraint-based* ideas of Khang *et al.* [50], Jain *et al.* [49] propose place-and-routing a portion of a completed design to incorporate design-specific timing constraints as watermarks. This will result in a unique bitstream that can be reproduced given the original HDL and constraints to prove ownership. This may work for a complete design rather the more desirable marking of individual cores, because a core vendor cannot enforce constraints through a netlist such that they always produce the same bitstream portion when multiple cores are combined together by the system designer. Van Le and Desmedt [65] describe possible attacks on the schemes proposed in [50, 62, 63]; however, the ones relating to LUT-based marks will require reverse engineering the bitstream, at least partially.

Ziener *et al.* [24] propose using the content of LUTs extracted from the design's netlist as a watermark. Then, LUT content is extracted from a suspected copied bitstream and it is statistically determined to some degree of certainty, if an unlicensed core is present; since the LUTs are part of the functional design, removing them also alters its correct operation, so it resists the above elimination and masking attacks. Ziener and Teich [122] propose using LUT-based signatures that once externally triggered for verification can be observed through power analysis.

The reactive nature of watermarks do not actually prevent theft, but may serve as initial evidence in court or trigger further investigation. It would make a hard case for a fraudster to claim that he independently arrived at an identical copy of artwork when photographs, audio, or art are concerned, though it is plausible that engineers independently arrive at very similar code, ones that may falsely trigger the statistical threshold for fraud in watermarking schemes; so the scheme must be robust enough such that the probability of this happening is very low. Then, of course, reactive schemes are only useful where ownership rights are enforced, further reducing their usefulness. As most of the counterfeit hardware comes from countries where that is not the case, these mechanisms are less appealing compared to active ones. Lastly, some watermarking techniques rely on the continued secrecy of bitstream encoding, opening them to various manipulation attacks that may prevent them from being used as conclusive evidence in court; if the fraudster encrypts the original plaintext watermarked bitstream, the detection and proof problems are made worse, if possible at all. For all the reasons above, watermarking should be implemented as a tool that is conveniently applied and verified, as part of an overall *active* defense mechanism. This will serve as a deterrent, help protect the watermark from being tampered with, but also provide evidence that the infringer had to actively attack the system in order to remove or alter it.

4.5 More bitstream encryption

This section covers some protection mechanisms that have been suggested in the past for bitstream protection, or as cloning deterrents.

Kean [52] suggests that embedded keys (programmed by the FPGA vendors, foundry, or system designers) be used such that the FPGA itself can encrypt and decrypt the bitstream without the key ever leaving the FPGA. As outlined in Figure 6, the first stage happens in a trusted facility where the plaintext bitstream is encrypted by the FPGA using the embedded key, and then stored on a PROM. While the system is deployed in the field, the bitstream is decrypted with the same internal key using an internal decryption

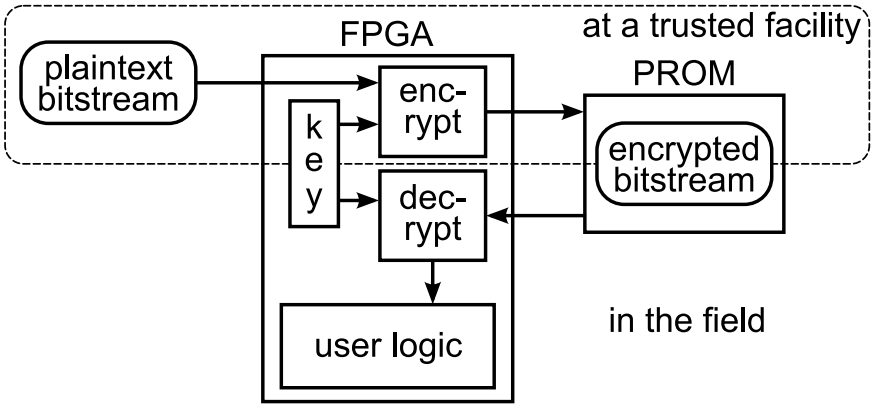

Figure 6: In a trusted facility the plaintext bitstream is encrypted using the embedded key and is stored in a PROM, while in the field the bitstream is decrypted on power-up. The advantage of this scheme is that the key need not leave the device.

core. The advantages of this scheme are that the manufacturing facility does need to be trusted and that the key does not need to ever leave the FPGA. The disadvantages are that unless the system developer is allowed to program the keys, other principals become trusted parties; both encryptors and decryptors must be used, which is fine for DES implementations, but costly for AES[12]; keys are permanently embedded which is a disadvantage to some; and, the logistical cost of key and design distribution are high. Kean also made the observation that it may be sufficient to only slightly increase the price of cloning and suggests embedded keys in the photo masks (artwork) be used in the manufacturing process. Each FPGA family member will have embedded in it one of a set of keys, but with no package marking indicating which. Since the device encrypts the design as with the previous scheme, the system designer does not need to care which key from that set is used on his FPGA. However, a cloner, having obtained a bitstream, will have to buy a number of FPGAs which is a multiple of the key set size for every one he can use, on average. The result is that cloning becomes more expensive, but at the expense of complicating the manufacturing process.

Bossuet *et al.* [17] propose a scheme where an embedded key is accessible to the user logic and uses partial reconfiguration to encrypt and decrypt the bitstream. First, a bitstream with a user-defined cipher is loaded onto the FPGA and used to encrypt another bitstream, which is then stored in a PROM. On the same PROM a decryption bitstream is also stored which is used to decrypt the encrypted bitstream in the field. The critical flaw of this scheme is that if the key is accessible to the user logic, anyone can read it out and decrypt the bitstreams, so some hard-wired access control mechanisms need to be put in place.

[12]Kean suggests the CBC mode, but other modes that only require the encryption function for both operations may be used.

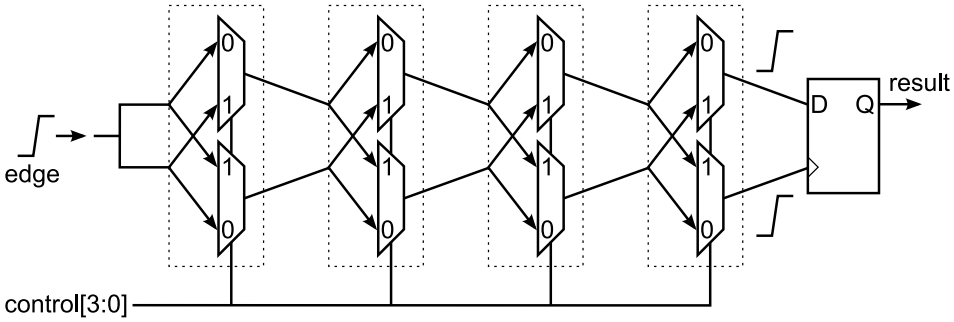

Figure 7: An "arbiter PUF" by Lim *et al.* [70] where a signal edge is propagated through arbiters controlled by a challenge vector to produce a single bit result depending on minute variation in the length of the connections. PUFs rely on uncontrollable manufacturing variability to extract entropy from the device's unique physical properties.

5 Ongoing research topics

FPGAs are popular amongst researchers for prototyping and demonstrating new ideas and concepts. This section covers ongoing research that relates to the security of FPGA design and use.

5.1 Physical unclonable functions

Physical Unclonable Functions (PUF[13]) are used to identify items or circuits based on their physical properties. Pappu *et al.* [84, 85] introduced *physical one-way functions* where the scatter pattern from a laser beam passing through, or reflected from, a block of epoxy is converted into a block-specific bit string. Since this string is inherent to that item, it may be used for its identification. Silicon PUFs (SPUF) were introduced by Gassend *et al.* [36], Lee *et al.* [66], and Lim *et al.* [70] and rely on the uncontrollable manufacturing variability of integrated circuits for uniqueness. PUFs are very attractive for authenticating devices and design-rights protection: creating a model for faking PUFs is incredibly hard given that is it based on physical properties of the individual die; derived keys "exist" only when needed, and are not permanently stored; invasive tampering changes the properties of the PUF such that the correct key can no longer be reproduced; there is no need to program a key, and it does not even need to leave the device; random challenge-response pairs can be created with some PUFs such that a unique string is generated for the purpose of authentication; PUFs are scalable in that more of them can be generated according to security needs; and finally, for FPGAs, several of the proposed structures can already be realized in existing devices, as we will see next.

The *arbiter PUF* is shown in Figure 7. Identically designed delay lines are routed through arbiters that are controlled by a challenge vector. A signal edge is simultaneously asserted for both routes, which propagates according to the multiplexers' setting until it reaches the sampling register, with the result determined by which signal arrived first. The two routes are designed and laid out to have identical delays, but because the manufacturing

[13] Also called Physical Random Functions but both are abbreviated to PUF.

process introduces very small uncontrolled variations, in practice, the routes will not have the same propagation delay on the fabricated integrated circuit. Suh and Devadas [103] suggest the *ring oscillator PUF*, where many identical free running ring oscillators are used to produce unique bit strings based on slight delay variations between the oscillator's inverters. Guajardo *et al.* [39] propose using the initial states of large uninitialized SRAM blocks in Altera devices as a source of entropy for key generation that is used for design protection. Tuyls *et al.* [112] have developed a coating applied to the die such that its unique capacitive distribution can be measured to extract a key and detect invasive tampering.

When used for identification and to produce keys for cryptographic applications, PUFs should be *unique* for each die and *reproducible* for a given die irrespective of temperature and voltage variation (within operating ranges). These properties are quite challenging to achieve and pose the most difficulty in the generation of consistent, yet randomly distributed, strings. The research on delay-based PUFs [36, 70, 103] has shown that these circuits can be implemented on FPGAs, and by using error correcting codes, the above requirements can be satisfied to a reasonable extent. However, as of late 2007, the author has no knowledge of a commercial use of silicon PUFs, or ones that are used in FPGAs. The reason might be the resource overhead required to make them work, which is an aspect often neglected in research publications during analysis and evaluation of their applicability. The error correcting codes or "Fuzzy Extractors" [39, 112] may require significant resources, much more than the simple structures that generate the "raw" bit strings. An elegant solution may use the embedded temperature and core voltage monitors available in recent FPGAs to minimize the need for error correction by operating the PUF only under specific environmental conditions.

In summary, robust PUFs will solve a real industry problem, especially for FPGA-design protection; one would expect this area of research will continue to be very active in the next few years.

5.2 Bitstream authentication

Bitstream authentication was already mentioned as a solution to several problems associated with FPGA security. Authentication provides two things: *entity identification*, which allows the receiver of a message to know for certain who the message was from; and, *message integrity* that guarantees to the receiver that the message has not been modified in transit. Authentication is sometimes considered to be more important than encryption, as the damage an adversary can cause by impersonation can be much greater than him having the ability to merely read secret communications. Bitstream encryption is a good step forward, but we have seen that it is not meant to guarantee the data's integrity, only its confidentiality. Encryption, therefore, protects the bitstream from cloning and reverse engineering in transit *independently of the FPGA* while authentication guarantees the *correct* and *intended* operation of the bitstream while it is running on the FPGA.

FPGA vendors have traditionally relied on linear checks, such as *cyclic redundancy* (CRC), to protect FPGAs from bitstreams corrupted by transmission errors on a noisy channel. The result of a corrupt bitstream is wrong functionality and also circuit shorts (or, contention) that may damage the device if the current is sufficiently large and lasts for a long period of time. Linear codes are good for detecting occasional, unintentional bit

errors, but are poor at handling maliciously inserted bit manipulations since they can be forged with relative ease [102]. Linear checks, of course, lack a critical requirement of authentication: entity identification.

Authentication is most often achieved by computing a Message Authentication Code (MAC). MACs are generated by one-way, collision-free, functions that take an arbitrary length message and produce a fixed length string of bits such that it is computationally infeasible for an attacker to find the original message, or find two different messages that result in the same string. The entity identification is achieved by incorporating a shared key, between sender and receiver, to the process.

$$\begin{aligned} \text{Encryption}: &\quad \text{plaintext} + \text{key} \xrightarrow{E_{key}} \text{ciphertext} \\ \text{Authentication}: &\quad \text{plaintext} + \text{key} \xrightarrow{H_{key}} \text{plaintext} \,||\, MAC \end{aligned}$$

The MAC string is appended to the message, which may or may not be encrypted, and is verified at the receiver by computing the MAC of the message again. If those MACs match, the receiver knows that the message is authentic and that whoever produced it is in possession of the shared key. MACs require symmetric keys to be established prior to communication but signature-based authentication can also be done using Public Key Cryptography (PKC) where each participant has a private and public key. By signing the message with the private key, a signature is generated that can be verified with the public key. PKC has all the necessary properties for solving bitstream security, except that current implementations are too costly in hardware for reaching the configuration throughput required in order for them to be integrated as a hard core by the vendors.

For computing MACs we can use hash functions or block ciphers in certain modes. Block-cipher mode CBC-MAC can be used to produce a MAC by simply taking the last enciphered block as the MAC. CBC-MAC is insecure for variable message lengths, and NIST has now standardized a new mode called Cipher-based MAC (CMAC) [79, Pub. 800-38B] to overcome this shortcoming. In either case, the same key should not be used to both encrypt and MAC the same plaintext. Recognizing that using the same key for both operations is a desirable property, new modes and ciphers have been proposed to do that securely; these are collectively called Authenticated Encryption (AE) and Black [15] provides a good introduction to them.

For bitstream authentication, Parelkar [86] suggested and evaluated the use of various AE and hash algorithms, concluding that CCM was best fit with respect to performance and size; Parelkar and Gaj [87] suggested the use of the AEX mode. Non-standardized, or patented algorithms are not appealing for FPGA vendors to adopt since their security has not been fully scrutinized, and for fear of future legal complications. The demand for authentication may not be sufficiently high for FPGA vendors to commit large portions of silicon area to this function, so digital signature schemes are probably not an option at this point. An important reason why authenticated encryption may not be ideal is because it uses a single key for both operations, but this might be a disadvantage when these need to be separated due to key management considerations, access control, and authenticating multiple entities. Given the growth in size, it is likely that in the future, a multi-user environment will reside within a single FPGA; then, it would be advantageous to have these operations separated for role- and identity-based authentication (similar

key management issues outlined in Section 4.2.2 apply here as well). The author [30] gives an application example where authentication would be useful on its own, without encryption, to provide code audit functionality for voting machines, and also facilitate an authenticated "heart-beat". Also, an argument is made that two parallel block-cipher cores, one in Counter [79, Pub. 800-38A] and the other in CMAC mode, would be a suitable solution considering the technical and economical constraints of the FPGA vendors.

5.3 FPGA digital rights management

Most of the topics covered in this survey are related in one way or another to "intellectual property" protection and secure distribution of designs. More precisely, restrict FPGA bitstreams and cores from being used in unauthorized devices or by unauthorized principals, and enabling a pay-per-use model, basically, Digital Rights Management (DRM) for FPGAs. The term "DRM" now carries a bad connotation due to the abuse of this tool by some companies and industries, though for our application, it may actually benefit both core *and* system designers. For example, an unknown start-up would benefit from paying per-use for protected cores, rather than pay large sums up-front for unlimited use (before they know how successful their product will become), while the cores vendor benefits from system developers buying his cores, ones that would not be able to afford it otherwise. Up to now, the "design-reuse" industry has dealt with distribution by mostly relying on social means, such as "trusted partners" and reputation considerations we will discuss in Section 6.1. An industry-wide panel discussion in early 2007 [117, 118] provides some telling insight into the industry's perception of using a software encryption flow for cores protection. The concluding remarks are that the current trust-based system is working well for large corporations — less so for start-ups — and a better solution is desirable for the long-run, but is not necessarily urgent. These discussions primarily pertain to ASIC cores, with the unique FPGA design flow and usage model typically glossed over and added as an attachment. Thus, considering the ever growing size and use of FPGAs and their usage model, DRM for FPGAs may be more pressing and should be examined more carefully. We defined an ideal model for core distribution in Section 2.1, but achieving it all at once would be near impossible, and perhaps even impractical. Thus, a better understanding of the problem is required, and consideration given to issues more pressing to cores vendors such that those are dealt with initially.

Although some tool vendors allow the input of a post-synthesis encrypted netlist, without industry-wide compatibility, this may not work well. Core vendors, therefore, devise their own protection mechanisms based on their risk perception. Altera, for example, allows the compilation of non-Altera cores, hosted on its website, for simulation and resource utilization purposes but disallows the generation of a bitstream though software enforcement [8, App. Note 343]. For its own cores, time-restricted bitstreams ("untethered"), or ones that require constant connection to the software through the programming cable ("tethered"), are generated for on-chip evaluation [8, App. Note 320]. Xilinx also provides evaluation facilities of a handful of cores for their embedded processors, which expire after "6–8 hours when running at the nominal clock frequency specified for the core" [121], while other HDL cores are available through various licenses that allow a time-limited evaluation Both Altera and Xilinx's schemes require signing, or agreeing with, licensing agreements.

And, most likely, rely on the bitstream's encoding, not cryptography, for enforcing the timing restrictions while the cores are operating inside the FPGA. In June 2006, Synplicity developed the "Open IP Encryption Initiative" for secure core exchange [25] and offered it to the "Virtual Socket Interface Alliance" (VSIA) [116] to become an industry standard. But in June 2007 VSIA announced it is ceasing operation [6] leaving the status of Synplicity's initiative unclear[14].

Secure distribution of digital designs and content is incredibly challenging, and we know it has yet to be solved in either software or hardware. In the case of FPGA design, we must allow the system designer to simulate, verify, and integrate several cores into his own design, while still being assured that the final core he receives is identical to the one simulated[15]. Key management and distribution is a problem; specifically who is entrusted with them, and who can access the designs themselves. If we use cryptography, export-restrictions may be an issue; can we still utilize these solutions knowing that we cannot use them where they are needed the most? At the low-level, how can we prevent the system designer from reading back the design, or streaming it out using the ICAP, yielding an unencrypted non-DRM'd version of the bitstream? Finally, and most importantly, we must justify our proposals to the FPGA vendors in terms of their gain from adopting it. As we discussed in Section 2, FPGA vendors need to be convinced that the solution is worth their development time and commitment silicon real-estate, otherwise these proposals will remain paper designs. Thus, a cursory economic assessment needs to be made, while making sure that incentives for adoption are well aligned for all principals and that the overall system is not too cumbersome.

5.3.1 Secure processor-code distribution

Several FPGA families have hard-core embedded processors, while soft-core ones (*e.g.* Nios and MicroBlaze) can be instantiated using the regular HDL flow. These are used to execute instruction sets corresponding to the architecture, as would otherwise be possible if the processor was an external device. The code is written in a high level language such as C, compiled, and then becomes part of the bitstream as part of a RAM, for example; this compiled code may be updated without full reconfiguration of the rest of the configuration fabric. Simpson and Schaumont [93] address the scenario where the system developer creates a design that uses either a hard or soft processor, but is able to accept updates to the compiled code[16] from third parties. Their protocol is able to authenticate and decrypt code based on keys and challenge-response pairs derived from an embedded PUF. This requires the FPGA vendor to collect many challenge-response pairs of vectors from the PUF, and enroll those, along with the FPGA's unique identifier, with a trusted party. Then, cores vendors enroll their compiled code, along with a unique identifier, with the trusted party. Through several exchanges between principals and the mutually trusted party, the compiled code is encrypted such that it can only be used in

[14]Since this is the case, elaboration of the security of the scheme is avoided, but it does have some ill-advised properties, such as the distribution of vendors' private keys with every instance of the software.

[15]The extent to which system designers may care about this last point varies, but the trust aspects associated with it are important.

[16]The authors use the terms "software", or SW, interchangeably with "IP", which is a bit confusing; here it is referred to as "compiled code".

FPGAs that can reproduce the correct PUF response given a certain challenge vector. The scope to which this scheme applies is limited to the secure distribution of compiled code for embedded processors and does not apply to cores that use the FPGA's fabric because then a feedback mechanism would be required such that bitstream-level "stitching" with the system developer's design is possible. Guajardo *et al.* [39] suggest enhancements to the Simpson-Schaumont protocol, and also describe an implemented PUF, as discussed in Section 5.1, which was originally simulated by an AES core.

5.4 Physical isolation of cores

For protection of individual cores within a multi-core design Huffmire *et al.* [47] propose "moats" to isolate and "drawbridges" macros to connect the cores such that no sensitive information leaks from one core to another. The cores are separated by unrouted columns and rows, and may only communicate through macros with pre-defined connections. Commonly known in the security industry as "red-black separation", elements of a design that handle classified material (red) need to be completely separated from the unclassified/encrypted (black) portion of the design. Huffmire *et al.* describe a set of tools that operate directly on the bitstream, to verify the separation after the insertion of arbitration modules ("moats") between the cores that are supposed to prevent leaks through a shared memory bus. In order to create the moats, certain routing resources are prohibited from being used during the place and route process, although it is unclear how this was done.

Similarly, and likely preceding the research of Huffmire *et al.*, McLeane and Moore [72] reported a collaboration between Xilinx and the U.S. National Security Agency that yielded a set of tools and macros for isolation in the Virtex-4 family of FPGAs. The analysis showed that it is sufficient to provide a single CLB column "fence" to provide adequate isolation, while connecting them using pre-defined "bus macros". The unclassified portion of the NSA-Xilinx report reveals that the Virtex-4 FPGA has gone through rigorous analysis by the NSA and was deemed secure for red-black applications. The short unclassified report also indicated a development of an internal "security monitor", but does not provide implementation details.

5.5 Evolvable hardware

Modeled after biological evolution, circuits are made to evolve by inducing "mutations" and subsequently assigned "fitness" scores iteratively until the desired functionality is achieved. Fitness is assigned to a circuit, or portions of it, based on the degree to which it meets the specifications in a process similar to natural selection. Reconfigurable devices are an ideal platform in which such circuits can be developed and indeed they are used extensively in this field [37, 90, 91, 107].

Evolved circuits can produce designs that are significantly different from conventionally designed ones; Thompson and Layzell [108] described "the most bizarre, mysterious" evolved circuit implemented on a Xilinx XC6216 FPGA. The circuit's function was to be able to distinguish between 1 kHz and 10 kHz input clocks without having a clock of its own. It worked, but remarkably, the exact way the circuit achieved this functionality

could not be fully explained. It is evident from the research that the functionality is tightly coupled to the physical attributes of the individual FPGA it ran on — the same circuit did not operate correctly on other devices of the same size and family. Donlin and Trimberger [29] suggest using the attributes of evolved circuits to extract unique keys from each FPGA for the purpose of bitstream protection, though there doesn't seem to be any active research on this particular application.

5.6 Cryptographic algorithms: implementing and breaking

Though only tangentially related to our main topic, here we discuss how FPGAs are used to implement and "break" security algorithms. There exists an extensive collection of papers starting from the mid 1990s with reports of incremental improvement in performance for nearly every cryptographic algorithm in existence. Implementers are taking advantage of technological advancements, and algorithmic tweaks to match, in order to improve on the current benchmark in terms of throughput and/or resource use. Wollinger *et al.* [119] provide a good summary while Synaptic Laboratories [104] maintain an on-line index of cryptography-related cores implementations for both FPGAs and ASICs. Anyone who has tried *fairly* comparing FPGA implementations from different sources, however, knows that it is a near impossible task due to the amount of variables and inconsistency in reporting. The situation would be much improved if primitives were used for reporting resource use (*e.g.* flip-flops, RAM blocks, look-up tables) instead of the somewhat arbitrarily component-bundling definitions that are inconsistent between devices (*e.g.*, slices, CLBs, logic elements). Even then, without the context of a target application as a concrete benchmark, comparing "fastest/smallest ever" implementations is somewhat irrelevant.

FPGAs are also used as part of machines that brute force cryptographic algorithms, as they strike a balance on the cost-performance scale between general purpose CPUs and ASICs. Software implementations are easily made but are slow compared to ASICs, which, in turn, are hard to design and expensive to fabricate. FPGAs are cheaper than ASICs, better suited for most cryptographic algorithms than software, are widely available, and are highly parallelizable. They are better at specific tasks compared to general purpose CPUs, especially for functions that can be divided into many small pieces operating in parallel.

Ever since DES was made into a standard, many people have hypothesized about the cost of a brute force attack. The first (public) dedicated machine built for this purpose was the Electronic Frontier foundation's (EFF) "Deep Crack". Built in 1998 for $250 000, using 1 536 ASICs, it broke the "RSA DES Challenge II-2" in 56 hours [33]. In 1999, Hamer and Chow [44] proposed cracking DES in 1 040 days using the "Transmogrifier-2a" at a cost of approximately $60 000; this machine consisted of 32 Altera FLEX 10K100 FPGAs running at 25 MHz. Clayton and Bond [21] have describe how they broke the DES implementation in the IBM 4758 hardware security module by exploiting a short-cut and using a low-cost Altera FPGA development board. Kumar *et al.* [61] have designed "COPACOBANA", a $10 000 backplane with 120 low-cost Xilinx Spartan-3 FPGAs, six each on twenty modules, operating in parallel. The researchers have used this platform as a DES key search engine and as of September 2007, it can find a key within 6.4 days, on average. Bono *et al.* [16] demonstrated the use of 16 Xilinx Spartan-3 FPGAs operating in parallel for extracting

keys from RFID-based tokens that uses a proprietary cipher, called DST40. As of late 2007, there is an on-going project [106] to cracking the A5 stream cipher in real time using commercial FPGA modules.

Moore's Law and the various device architectures are responsible for most of these performance improvements. The same generic HDL code synthesized for two device generations would nearly always show a significant performance increase with the latest one, and that should be expected. For example, now that high-end FPGAs moved to 6-input LUTs, DES s-boxes (which are the most costly part of the design) can be made with a single logic level, guaranteeing a boost in performance over 4-input LUT architectures; some FPGAs can perform 48 bit bit-wise logic operation at over 500 MHz, further making DES operations much faster than previously possible. A final remark on the topic is the "NSA@home" project [82], where recycled HDTV hardware containing 15 Virtex II PRO FPGAs are used to built a pre-image brute force attack machine against SHA-1 that purportedly can find a pre-image of an 8 character password hash from a 64 character set in about 24 hours. This is quite exceptional, and we will most likely see more of this recycling in the future as FPGAs become more prevalent and the difference between old and new ones would mainly be in specialization, rather than performance.

6 Trust, adversaries, and metrics

6.1 Trust in the flow

How can users of software and hardware be confident that these tools are indeed "honest", *i.e.*, not spying on them or covertly inserting malicious code or circuits into their products. Software and chip verification is an option, but a prohibitively costly one. The sheer enormity of EDA tools make this impractical, let alone the fact that vendors generally do not make their source code available. Even if source code was available for compilation, how can one be sure that the compiler is trustworthy? How about simulators? These may also be corrupt. In *Reflections on trusting trust*, Thompson [109] elegantly discusses these issues and concludes that "you can't trust code that you did not totally create yourself... No amount of source-level verification or scrutiny will protect you from using untrusted code". Verifying integrated circuits is even harder than software verification, especially so without access to the design files and a complete audit along the manufacturing process.

We saw that the design process of both FPGAs and the systems that use them require the participation of principals not under the control and supervision of the designer, and if we accept Thompson's assertions, we also realize that it is unavoidable to extend trust to them. Some principals, however, may not have matching practices or attitudes towards the protection of others' design rights and that is why, of course, these rights are violated. Some of this mismatch is reconciled by the motivation of companies to build and maintain a positive reputation. The reputation of being honest, technologically advanced, quality-driven, *etc.* is an asset that is slowly gained, but easily lost, and without which companies can no longer be successful. By striving to maintain a reputation, companies are aligning themselves with the interests of their customers to whom they provide products and services (this, of course, assumes a competitive and open market).

Many millions of dollars are invested in designing and manufacturing an FPGA, and it is

in the interest of both the FPGA vendors and the foundries that these do not go to waste. This necessitates many self-enforced checkpoints to assure that no flaws or deviations from the intended design are made. For example, the foundry must deliver high-yielding wafers that are faithful to the original design, and in turn, FPGA vendors need to provide their customers reliable FPGAs that are faithful to the data-sheets. All this means is that there are numerous people and procedures in place to make sure the product is as it should be, which translates into a reasonable assurance of quality that many customers can rely on, or trust[17]. Still, colluding malicious insiders may be able to circumvent these audits, especially in the design phase, and indeed, this is still a viable threat.

Other points where trust is involved are the continued secrecy of bitstream encoding; trusting the manufacturing facility in which the system is built and tested not to overrun, sell or modify the designs; core vendors supplying designs true to simulation models; IC-packaging facilities inserting "key-loggers" into packages; and many others. Some of these issues can be mitigated by cryptographic measures and protocols, others may not, though many of the consideration are not technological, but economic.

6.1.1 System-level trust boundaries

There are many ways to configure an FPGA: through a network, PC, microprocessor, non-volatile memory, *etc.* Any of these devices can perform security related tasks such as encryption and authentication. Then, however, these devices and their environment must be trusted and so does the communication path to the FPGA. At the system level, therefore, we can define two trust boundaries that will assist in evaluating and designing defense mechanisms:

- *FPGA trust.* Only the FPGA is trusted for security operations on data it is processing, including the bitstream. In this setting, any external source of information, whether local or remote, is untrusted until validated and no operation to do so is delegated to other devices. All security mechanisms reside within the FPGA itself with the result of having all configuration methods are supported.
- *Peripheral trust.* The trust boundary is extended to the system level to include other components. Here, security related operations may be performed by external devices which are within the trusted perimeter. Once in this perimeter, the bitstream can be transmitted in the clear, as the system is assumed to be in a protected environment and be adequately tamper-proof.

When evaluating risk from attacks and the effectiveness of defenses, it is important to first determine the appropriate trust boundary.

6.2 Adversaries

Adversaries, attackers, thieves, enemies, "bad people", *etc.* are trying to either steal goods or information without compensation to the rightful owner, or use systems in an

[17]ASIC design and manufacturing has similar problems, but with FPGAs having the advantage of being generic, such that tampering may be detected with higher probability simply because it has a much larger user-base.

unintended way. How we label these people depends on our perspective, and in any security evaluation we must identify them and have a good estimate of their capabilities in order to arrive to a good defense strategy. It is common to quote the Abraham *et al.* [2] adversary classification system from 1991 — clever outsiders, knowledgeable insiders, funded organizations — to assist the analysis of threats. This system, however, seems too coarse and perhaps outdated to account for the developments of that past 15 years. "Clever outsiders" are no longer operating in isolation, most likely being a part of an on-line information-sharing community; they now also have access to expensive tools through hourly rentals, ones that were previously considered to be only accessible to "funded organizations". Funded organizations is too broad of a definition to be very useful today: governments have effectively infinite amounts of money, resources, and can ignore any kind of law or copyright. Defending against these types of organization is incredibly challenging. But what about a criminal organization? They are well funded and employ very clever and capable crooks, but have constraints that allow a designer many more defense options. The conclusion is that there are unique adversaries for each system, ones that need to be identified and placed on a continuum according to their resources and the potential harm they may bring. Once those are identified, an adequate defense strategy needs to be established and implemented.

6.3 Security levels of hardware modules

Although we should design our defenses per our application's threats, it is sometimes useful, or mandatory if you are a security-conscious system designer, to evaluate it against a set criteria. The most often used criteria is the U.S. Government's *Security requirements for cryptographic modules*, FIPS 140-2 [79], which defines levels of security for hardware devices. In July 2007, a draft of an updated publication, FIPS 140-3 [79], has been open for public comment before becoming a publication. This draft outlines the following 5-level classification criteria, each new level building on top of the previous one:

- *Security Level 1*: No specific physical security mechanisms.
- *Security Level 2*: Tamper evident coating or seals; role-based authentication; operating system needs to provide access control facilities and an audit trail pertaining to Sensitive/Cryptographic Security Parameters (SSP/CSP).
- *Security Level 3*: High probability of detection and response to physical tampering attempts; zeroization of CSPs; identity-based authentication; physically separated, trusted, and audited channel for entry and read-out of CSPs; resistance to timing analysis; OS preventing user-level modification of SSPs; trusted channel protects leakage of CSPs between software modules.
- *Security Level 4*: Detection and response to *all* physical tampering attempts; *immediate* zeroization of CSPs upon tampering; two-factor authentication; protection against environmental tampering; protection against power analysis; encryption and authentication of CSPs when module is not in use; robust audit of all operator accesses to data.

- *Security Level 5*: Encryption and authentication of all SSPs while module is not in use; opacity to non-visual radiation; zeroization circuitry is protected against disablement; protection against electromagnetic analysis; environmental failure protection.

Let us now examine volatile FPGAs with respect to these requirements. The standard defines three types of modules: single-chip, multiple-chip embedded, and multiple-chip stand-alone. The most interesting is the evaluation of the FPGA as a single-chip module since, as we discussed in Section 6.1, if the trust is *peripheral* all security functions can be delegated to other devices. With the knowledge we have gained so far, we are well equipped to do a preliminary evaluation of how volatile FPGAs fare with regards to these requirements.

Authentication: Role-based authentication is already required in SL2 and we have discussed the need for this security enhancement in terms of bitstreams. With basic bitstream authentication there will be a single role, the "administrator", with additional keys enabling further role- and identity-based authentication functionality. The critical part is the initial configuration. Once its authenticity has been established, additional authentication of data and two-factor authentication can be done using the user logic. Tamper proofing and resistance: as mentioned before, a Xilinx and NSA collaboration brief [72] refers to a "security monitor" to enhance security properties from within the FPGA, which is certainly a step towards embedded tamper detection for the FPGA to function as a single-chip module. However, already at SL2 there is a requirement for an opaque enclosure, which will require the die and printed circuit carrier to be encased in potting material. To accommodate for the requirements of higher levels, the die must permanently cease to operate upon tampering, and mesh detectors added along with active, always-on, circuitry to monitor it and zeroize SSP/CSPs when the FPGA is not in use. Heat density and dissipation may be a problem with potting, as are environmental conditions triggering the tamper detection mechanisms during transport. Audit: this is not currently possible in FPGAs because there is no way to store information after power-down; it may require a stacked-die approach with a secure trusted interface (it is unclear whether this qualifies as a single-chip module under the definition of the publication, though), or having non-volatile memory inside the FPGA. Side-channels: timing analysis could be mitigated by the designer, while further study of power and electromagnetic analysis is still required. Non-visual radiation attacks: these were discussed in Section **??** and the available solutions against them are quite robust. The publication draft also discusses requirements for the software running in the module. Is the bitstream considered "software"? That's not an easy question to answer, but some of the requirements, such as built-in self tests and integrity checks, can certainly be satisfied through the digital design.

The conclusion is that volatile FPGAs, in their current form and on their own, will not pass Security Level 2 certification, but with the addition of external authentication, temper-evidence and resistance mechanisms and audit facilities, they can be certified for higher levels as a single-chip modules.

7 Conclusion

Since the late 1990s we have seen volatile FPGAs advance from simple logic devices to large, complex, "systems-on-chip" that may no longer be handled by a single engineer implementing simple logic functions. FPGA designs are big, complicated, and require a large investment by system developers who would like to profit from them. This survey has attempted to capture the state-of-the-art in design protection, and provide the foundations for future discussion, and introduce readers to possible attacks and defenses. Interesting and relevant research topics were described along with the social and economic aspects of the field. Most readers would only find a subset of the topics covered relevant for their work, but the purpose was to provide a full picture in order for those readers to be better informed when constructing a threat model and corresponding defense mechanism, as a system is specified and designed. Considering the fast pace of the industry and research activity, it is likely that this survey will need to be updated on a regular basis, but most of the concepts can be expected to remain the same.

In 2002, Tom Kean [52] summarized: "Lack of design security has long been the skeleton lurking in the closet of the SRAM FPGA industry". As we have seen, there are still many open problems and items to explore for enhancing the security of FPGAs and designs made for them.

Acknowledgments

My research is funded by Xilinx Inc., and I am grateful for their support. My gratitude is also extended to the people who invested their time reading and commenting on this manuscript as it was written: Ross Anderson, Richard Clayton, Tom Kean, Markus Kuhn, Steven Murdoch, Christof Paar, Patrick Schaumont, François-Xavier Standaert, and Robert Watson.

References

[1] A. T. Abdel-Hamid, S. Tahar, and E. M. Aboulhamid. IP watermarking techniques: survey and comparison. In *IEEE International Workshop on System-on-Chip for Real-Time Applications*, 2003. `http://ieeexplore.ieee.org/Xplore/defdeny.jsp?url=/iel5/8609/27279/01213006.pdf`

[2] D. G. Abraham, G. M. Dolan, G. P. Double, and J. V. Stevens. Transaction security system. *IBM Systems Journal*, 30(2):206–229, 1991. `http://www.research.ibm.com/journal/sj/302/ibmsj3002G.pdf`

[3] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM side-channel(s): Attacks and assessment methodologies. Technical Report 2001/037, IBM Watson Research Center, 2001. `http://www.research.ibm.com/intsec/emf-paper.ps`

[4] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM side-channel(s). In *Cryptographic Hardware and Embedded Systems Workshop*, volume 2523 of *LNCS*, pages 29–45, August 2002. `http://www.springerlink.com/content/mvtxbq9qa287g7c6/`

[5] D. Agrawal, J. R. Rao, and P. Rohatgi. Multi-channel attacks. In *Cryptographic Hardware and Embedded Systems Workshop*, volume 2779 of *LNCS*, pages 2–16, Septem-

ber 2003. http://researchweb.watson.ibm.com/people/a/agrawal/publications/CryptoBytes2003.pdf
[6] V. S. I. Alliance. SoC standards leader VSI Alliance announces plans to close operations, July 2007. http://www.vsia.org/news/vsia_plans_to_close.htm
[7] Alliance for Gray Market and Counterfeit Abatement. *Managing the risks of counterfeiting in the information technology industry*, August 2006. http://www.agmaglobal.org/press_events/press_docs/Counterfeit_WhitePaper_Final.pdf
[8] Altera Corp. http://www.altera.com
[9] *Court issues preliminary injunction against Clear Logic in Altera litigation*, Altera Corp., July 2002. http://www.altera.com/corporate/news_room/releases/releases_archive/2002/corporate/nr-clearlogic.html
[10] R. J. Anderson. *Security engineering: A guide to building dependable distributed systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001. ISBN 0471389226. http://www.cl.cam.ac.uk/~rja14/book.html
[11] R. J. Anderson and M. G. Kuhn. Low cost attacks on tamper resistant devices. In *International Workshop on Security Protocols*, pages 125–136, 1998. http://www.cl.cam.ac.uk/~mgk25/tamper2.pdf
[12] R. J. Anderson and M. G. Kuhn. Tamper resistance – a cautionary note. In *USENIX Workshop on Electronic Commerce Proceedings*, pages 1–11, November 1996. http://www.cl.cam.ac.uk/~mgk25/tamper.pdf
[13] R. J. Anderson, M. Bond, J. Clulow, and S. P. Skorobogatov. Cryptographic processors – a survey. Technical Report 641, University of Cambridge, Computer Laboratory, August 2005. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-641.pdf
[14] K. Austin. *Data security arrangements for semiconductor programmable devices*, United States Patent Office, 1995. http://patft1.uspto.gov/netacgi/nph-Parser?patentnumber=5388157
[15] J. Black. *"Authenticated encryption" in Encyclopedia of Cryptography and Security*, section A, pages 10–21. Authenticated encryption. Springer, 2005. http://www.cs.colorado.edu/~jrblack/papers/ae.pdf
[16] S. Bono, M. Green, A. Stubblefield, A. Juels, A. Rubin, and M. Szydlo. Security analysis of a cryptographically-enabled RFID device. In *USENIX Security Symposium*, pages 1–16, July-August 2005. http://www.usenix.org/events/sec05/tech/bono/bono.pdf
[17] L. Bossuet, G. Gogniat, and W. Burleson. Dynamically configurable security for SRAM FPGA bitstreams. April 2004. http://www.lilianbossuet.com/en/Doc/publications/Bossuet_RAW04.pdf
[18] M. Bucci, L. Giancane, R. Luzzi, G. Scotti, and A. Trifiletti. Enhancing power analysis attacks against cryptographic devices. In *Circuits and Systems Symposium*, May 2006. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1693232
[19] V. Carlier, H. Chabanne, E. Dottax, and H. Pelletier. Electromagnetic side channels of an FPGA implementation of AES. *Cryptology ePrint Archive*, (145), 2004. http://eprint.iacr.org/2004/145.pdf
[20] B. D. Christiansen. FPGA security through decoy circuits. Master's thesis, Air Force Institute of Technology, Ohio, USA, March 2006. http://stinet.dtic.mil/cgi-bin/GetTRDoc?AD=ADA454021&Location=U2&doc=GetTRDoc.pdf
[21] R. Clayton and M. Bond. Experience using a low-cost FPGA design to crack DES keys. In *Cryptographic Hardware and Embedded Systems Workshop*, volume 2523 of *LNCS*, pages 579–592, 2002. http://www.cl.cam.ac.uk/~rnc1/descrack/DEScracker.pdf
[22] J. H. Conway. *On numbers and games*. Academic Press, 1976. http://en.wikipedia.org/wiki/On_Numbers_and_Games

[23] J. Daemen and V. Rijmen. *AES proposal: Rijndael*, September 1999. http://www.gel.ulaval.ca/~klein/maitrise/aes/rijndael.pdf

[24] A. S. Daniel Ziener and T. Jürgen. Identifying FPGA IP-Cores based on lookup table content analysis. In *Field Programmable Logic and Applications*, pages 481–486, August 2006. http://www12.informatik.uni-erlangen.de/publications/pub2006/zienerfpl06.pdf

[25] A. Dauman. *An open IP encryption flow permits industry-wide interoperability*, Synplicity, Inc., June 2006. http://www.synplicity.com/literature/whitepapers/pdf/ip_encryption_wp.pdf

[26] E. De Mulder, P. Buysschaert, S. B. Örs, P. Delmotte, B. Preneel, G. Vandenbosch, and I. Verbauwhede. Electromagnetic analysis attack on an FPGA implementation of an elliptic curve cryptosystem. In *EUROCON: Proceedings of the International Conference on "Computer as a tool"*, pages 1879–1882, November 2005. http://www.sps.ele.tue.nl/members/m.j.bastiaans/spc/demulder.pdf

[27] E. De Mulder, S. B. Örs, B. Preneel, and I. Verbauwhede. Differential electromagnetic attack on an FPGA implementation of elliptic curve cryptosystems. In *World Automation Congress*, July 2006. https://www.cosic.esat.kuleuven.be/publications/article-737.pdf

[28] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A practical implementation of the timing attack. In *CARDIS*, pages 167–182, 1998. http://users.belgacom.net/dhem/papers/CG1998_1.pdf

[29] A. P. Donlin and S. M. Trimberger. *Evolved circuits for bitstream protection*, United States Patent Office, May 2005. http://patft1.uspto.gov/netacgi/nph-Parser?patentnumber=6894527

[30] S. Drimer. Authentication of FPGA bitstreams: why and how. In *Applied Reconfigurable Computing*, volume 4419 of *LNCS*, pages 73–84, March 2007. http://www.cl.cam.ac.uk/~sd410/papers/bsauth.pdf

[31] S. Drimer and S. J. Murdoch. Keep your enemies close: Distance bounding against smartcard relay attacks. In *USENIX Security Symposium*, August 2007. http://www.cl.cam.ac.uk/~sd410/papers/sc_relay.pdf

[32] J. Edwards. No room for second place: Xilinx and Altera slug it out for supremacy in the changing PLD market. *EDN Magazine*, June 2006. http://www.edn.com/index.asp?layout=article&articleid=CA6339519

[33] Electronic Frontier Foundation. *Cracking DES: Secrets of encryption research, wiretap politics and chip design*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1998. ISBN 1565925203. http://cryptome.org/cracking-des/cracking-des.htm

[34] B. Fechner. Dynamic delay-fault injection for reconfigurable hardware. In *Parallel and Distributed Processing IEEE Symposium*, page 282.1, April 2005. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1420244

[35] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems Workshop*, volume 2162 of *LNCS*, pages 251–261, May 2001. http://www.gemplus.com/smart/rd/publications/pdf/GMO01ema.pdf

[36] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Silicon physical random functions. In *ACM Conference on Computer and Communications Security*, pages 148–160, 2002. http://csg.lcs.mit.edu/pubs/memos/Memo-456/memo-456.pdf

[37] T. G. W. Gordon and P. J. Bentley. On evolvable hardware. In *Soft Computing in Industrial Electronics*, pages 279–323, 2002. http://www.cs.ucl.ac.uk/staff/t.gordon/scie.pdf

[38] W. S. Gosset. Atmel at40k/94k configuration format documentation (usenet comp.arch.fpga), August 2005. http://groups.google.com/group/comp.arch.fpga/msg/a90fca82aafe8e2b

[39] J. Guajardo, S. S. Kumar, G. J. Schrijen, and P. Tuyls. FPGA intrinsic PUFs and their use for IP protection. In *Cryptographic Hardware and Embedded Systems Workshop*, volume 4727 of *LNCS*, pages 63–80, September 2007. void

[40] S. A. Guccione, D. Levi, and P. Sundararajan. Jbits: A java-based interface for reconfigurable computing. In *Military and Aerospace Applications of Programmable Devices and Technologies*, 1999. http://www.io.com/~guccione/Papers/MAPPLD/JBitsMAPPLD.pdf

[41] P. Gutmann. Data remanence in semiconductor devices. *USENIX Security Symposium*, pages 39–54, August 2001. http://www.cypherpunks.to/~peter/usenix01.pdf

[42] P. Gutmann. Secure deletion of data from magnetic and solid-state memory. In *USENIX Workshop on Smartcard Technology*, pages 77–89, July 1996. http://www.cs.cornell.edu/people/clarkson/secdg/papers.sp06/secure_deletion.pdf

[43] I. Hadžić, S. Udani, and J. M. Smith. FPGA viruses. In *Field Programmable Logic and Applications*, volume 1673 of *LNCS*, pages 291–300, 1999. http://www.springerlink.com/content/9wnbm5eqgpjvlcug/BodyRef/PDF/558_10705539_Chapter_30.pdf

[44] I. Hamer and P. Chow. DES cracking on the Transmogrifier 2a. In *Cryptographic Hardware and Embedded Systems Workshop*, volume 1717 of *LNCS*, pages 13–24, 1999. http://www.springerlink.com/content/bqllltjjm24h671a/fulltext.pdf

[45] G. P. Hancke. A practical relay attack on ISO 14443 proximity cards, 2005. http://www.cl.cam.ac.uk/~gh275/relay.pdf

[46] G. P. Hancke and M. G. Kuhn. An RFID distance bounding protocol. In *Security and Privacy for Emerging Areas in Communications Networks*, pages 67–73, 2005. http://www.cl.cam.ac.uk/~gh275/distance.pdf

[47] T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, R. Kastner, T. Levin, T. Nguyen, and C. Irvine. Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems. In *IEEE Symposium on Security and Privacy*, pages 281–295, 2007. http://www.cs.ucsb.edu/~sherwood/pubs/IEEESP-moats.pdf

[48] iRoC Technologies. *Radiation results of the SER test of Actel, Xilinx and Altera FPGA instances*, 2004. http://www.actel.com/documents/OverviewRadResultsIROC.pdf

[49] A. K. Jain, L. Yuan, P. R. Pari, and G. Qu. Zero overhead watermarking technique for FPGA designs. In *ACM Great Lakes symposium on VLSI*, pages 147–152, 2003. http://www.ece.umd.edu/~gangqu/research/papers/c027.pdf

[50] 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 CAD of Integrated Circuits and Systems*, 20(10): 1236–1252, 2001. http://www.eecs.umich.edu/~imarkov/pubs/jour/j009.pdf

[51] T. Karnik, P. Hazucha, and J. Patel. Characterization of soft errors caused by single event upsets in CMOS processes. *IEEE Transactions on Dependable and Secure Computing*, 1 (2):128–143, 2004. http://ieeexplore.ieee.org/iel5/8858/29698/01350778.pdf

[52] T. Kean. Secure configuration of Field Programmable Gate Arrays. In *Field Programmable Logic and Applications*, pages 142–151, 2001. http://www.algotronix.com/content/security%20FPL%202001.pdf

[53] T. Kean. Cryptographic rights management of FPGA intellectual property cores. In *Field Programmable Gate Arrays Symposium*, pages 113–118, 2002. http://www.algotronix.com/content/security%20fpga2002.pdf

[54] D. Kessner. *Copy protection for SRAM based FPGA designs*, May 2000. http://web.archive.org/web/20031010002149/http://free-ip.com/copyprotection.html

[55] J. Khatib. *Open hardware design trend*, January 2004. http://www.opencores.org/articles.cgi/view/12

[56] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, 2003. http://www.ece.northwestern.edu/~rjoseph/ece510-fall2005/papers/static_power.pdf

[57] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Cryptology Conference on Advances in Cryptology*, volume 1109 of *LNCS*, pages 104–113, 1996. http://www.cryptography.com/resources/whitepapers/TimingAttacks.pdf

[58] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Cryptology Conference on Advances in Cryptology*, volume 1666 of *LNCS*, pages 388–397, 1999. http://www.cryptography.com/resources/whitepapers/DPA.pdf

[59] O. Kömmerling and M. G. Kuhn. Design principles for tamper-resistant smartcard processors. In *USENIX Workshop on Smartcard Technology*, pages 9–20, May 1999. http://www.cl.cam.ac.uk/~mgk25/sc99-tamper.pdf

[60] M. G. Kuhn. Compromising emanations: eavesdropping risks of computer displays. Technical Report 577, University of Cambridge, Computer Laboratory, December 2003. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-577.pdf

[61] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler. Breaking ciphers with COPACOBANA – a cost-optimized parallel code breaker. In *Cryptographic Hardware and Embedded Systems Workshop*, volume 4249 of *LNCS*, pages 101–118, October 2006. http://www.springerlink.com/content/b5gp2783243w1964/fulltext.pdf

[62] J. Lach, W. H. Mangione-Smith, and M. Potkonjak. FPGA fingerprinting techniques for protecting intellectual property. In *Custom Integrated Circuits Conference*, 1998. http://www.ieeexplore.ieee.org/iel4/5666/15173/00694986.pdf?arnumber=694986

[63] J. Lach, W. H. Mangione-Smith, and M. Potkonjak. Robust FPGA intellectual property protection through multiple small watermarks. In *ACM/IEEE Conference on Design Automation*, pages 831–836, 1999. http://portal.acm.org/citation.cfm?id=310080

[64] Lattice Semiconductor Corp. http://www.latticesemi.com

[65] T. V. Le and Y. Desmedt. Cryptanalysis of UCLA watermarking schemes for intellectual property protection. In *Workshop on Information Hiding*, volume 2578 of *LNCS*, pages 213–225, 2002. http://www.springerlink.com/content/0qp5l9u76cehw6gv/fulltext.pdf

[66] J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas. A technique to build a secret key in integrated circuits for identification and authentication application. In *Proceedings of the Symposium on VLSI Circuits*, pages 176–159, 2004. http://people.csail.mit.edu/suh/papers/vlsi04.pdf

[67] A. Lesea. jbits & reverse engineering (Usenet comp.arch.fpga), September 2005. http://groups.google.com/group/comp.arch.fpga/msg/821968d7dcb50277

[68] A. Lesea. Personal communication, January 2006

[69] A. Lesea, S. Drimer, J. Fabula, C. Carmichael, and P. Alfke. The Rosetta experiment: Atmospheric soft error rate testing in differing technology FPGAs. *IEEE Transactions on Device and Materials Reliability*, 5(3):317–328, September 2005. http://www.saardrimer.com/docs/IEEE_Rosetta.pdf

[70] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas. Extracting secret keys from integrated circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(10):1200–1205, October 2005. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1561249

[71] S. Mangard, E. Oswald, and T. Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Springer-Verlag, Secaucus, NJ, USA, 2007. ISBN 978-0-387-30857-9. http://www.dpabook.org/

[72] M. McLean and J. Moore. FPGA-based single chip cryptographic solution–securing FPGAs for red-black systems. *Military Embedded Systems*, March 2007. http://www.mil-embedded.com/PDFs/NSA.Mar07.pdf

[73] A. Megacz. A library and platform for FPGA bitstream manipulation. *Field-Programmable Custom Computing Machines Symposium*, pages 45–54, April 2007. http://www.megacz.com/research/megacz-fccm07.pdf

[74] T. S. Messerges. Power analysis attack countermeasures and their weaknesses. In *Communications, Electromagnetics, Propagation and Signal Processing Workshop*, 2000. http://www.iccip.csl.uiuc.edu/conf/ceps/2000/messerges.pdf

[75] K. D. Mitnick and W. L. Simon. *The art of deception: Controlling the human element of security*. John Wiley & Sons, Inc., New York, NY, USA, 2002. ISBN 0471237124. http://portal.acm.org/citation.cfm?id=861316

[76] K. Morris. All is not SRAM - a survey of flash, antifuse, and EE programmable logic. *FPGA and Programmable Logic Journal*, February 2004. http://www.fpgajournal.com/articles/sram.htm

[77] D. C. Musker. Protecting and exploiting intellectual property in electronics. *IBC Conferences*, June 1998. http://www.jenkins-ip.com/serv/serv_6.htm

[78] *Aerospace science and technology dictionary*, National Aeronautics and Space Administration, 2006. http://www.hq.nasa.gov/office/hqlibrary/aerospacedictionary/

[79] National Institute of Standards, U.S. Department of Commerce. http://www.nist.gov

[80] E. Normand. Single event upset at ground level. *IEEE Transactions on Nuclear Science*, pages 2742–2750, December 1996. http://www.boeing.com/assocproducts/radiationlab/publications/SEU_at_Ground_Level.pdf

[81] J.-B. Note and É. Rannaud. From the bitstream to the netlist. Technical report, Département d'informatique École Normale Supérieure, September 2007. http://islsm.org/~jb/debit/bitstream.pdf

[82] NSA@home, September 2007. http://nsa.unaligned.org/

[83] S. B. Örs, E. Oswald, and B. Preneel. Power-analysis attacks on an FPGA — first experimental results. In *Cryptographic Hardware and Embedded Systems Workshop*, volume 2779 of *LNCS*, pages 35–50, September 2003. http://www.iaik.tugraz.at/Research/sca-lab/publications/pdf/Ors2003Power-AnalysisAttackson.pdf

[84] R. S. Pappu. *Physical one-way functions*. PhD thesis, Massachusetts Institute of Technology, March 2001. http://pubs.media.mit.edu/pubs/papers/01.03.pappuphd.powf.pdf

[85] R. S. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. Physical one-way functions. *Science*, 297:2026–2030, 2002. http://web.media.mit.edu/~brecht/papers/02.PapEA.powf.pdf

[86] M. M. Parelkar. Authenticated encryption in hardware. Master's thesis, George Mason University, Fairfax, VA, USA, 2005. http://mason.gmu.edu/~mparelka/reports/Milind_Thesis_pdf.pdf

[87] M. M. Parelkar and K. Gaj. Implementation of EAX mode of operation for FPGA bitstream encryption and authentication. In *Field Programmable Technology*, pages 335–336, December 2005. http://mason.gmu.edu/~mparelka/pdfs/fpt05.pdf

[88] J.-J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and countermeasures for smart cards. In *E-SMART: Proceedings of the International Conference on Research in Smart Cards*, pages 200–210, 2001. http://www.springerlink.com/

content/chmydkq8x5tgdrce/fulltext.pdf
[89] G. Seamann. *FPGA bitstreams and open designs*, April 2000. http://web.archive.org/web/20050831135514/http://www.opencollector.org/news/Bitstream/
[90] L. Sekanina. Towards evolvable IP cores for FPGAs. In *NASA/DoD Conference on Evolvable Hardware*, pages 145–154, 2003. http://www.fit.vutbr.cz/~sekanina/publ/eh03/eh03b.pdf
[91] L. Sekanina and Š. Friedl. An evolvable combinational unit for FPGAs. *Computing and Informatics*, 23(5):461–486, 2004. http://www.fit.vutbr.cz/~sekanina/publ/cai/cai04.pdf
[92] L. Shang, A. S. Kaviani, and K. Bathala. Dynamic power consumption in Virtex-II FPGA family. In *Field Programmable Gate Arrays Symposium*, pages 157–164, 2002. http://post.queensu.ca/~shangl/papers/shang02feb.pdf
[93] E. Simpson and P. Schaumont. Offline hardware/software authentication for reconfigurable platforms. In *Cryptographic Hardware and Embedded Systems Workshop*, volume 4249 of *LNCS*, pages 311–323, October 2006. http://rijndael.ece.vt.edu/schaum/papers/2006ches.pdf
[94] S. P. Skorobogatov. Semi-invasive attacks – a new approach to hardware security analysis. Technical Report 630, University of Cambridge, Computer Laboratory, April 2005. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-630.pdf
[95] S. P. Skorobogatov. Low temperature data remanence in static RAM. Technical Report 536, University of Cambridge, Computer Laboratory, June 2002. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-536.pdf
[96] J. M. Soden, R. E. Anderson, and C. L. Henderson. IC failure analysis: Magic, mystery, and science. *IEEE Design & Test*, 14(3):59–69, July 1997. http://portal.acm.org/citation.cfm?id=622765
[97] F.-X. Standaert, L. van Oldeneel tot Oldenzeel, D. Samyde, and J.-J. Quisquater. Differential power analysis of FPGAs : How practical is the attack? In *Field Programmable Logic and Applications*, pages 701–709, September 2003. http://www.springerlink.com/content/lgvq5dy9x37v1d7x/
[98] F.-X. Standaert, S. B. Örs, and B. Preneel. Power analysis of an FPGA implementation of Rijndael: Is pipelining a DPA countermeasure? In *Cryptographic Hardware and Embedded Systems Workshop*, volume 3156 of *LNCS*, pages 30–44, August 2004. http://www.springerlink.com/content/00ylcvw3rh7nwded/
[99] F.-X. Standaert, S. B. Örs, J.-J. Quisquater, and B. Preneel. Power analysis attacks against FPGA implementations of the DES. In *Field Programmable Logic and Applications*, pages 84–94, August 2004. http://www.springerlink.com/content/j6ru6h2a0jcw9vc3/
[100] F.-X. Standaert, F. Mace, E. Peeters, and J.-J. Quisquater. Updates on the security of FPGAs against power analysis attacks. In *Reconfigurable Computing: Architectures and Applications*, volume 3985 of *LNCS*, pages 335–346, 2006. http://www.springerlink.com/content/d38271pw36628h1r
[101] F.-X. Standaert, E. Peeters, G. Rouvroy, and J.-J. Quisquater. An overview of power analysis attacks against field programmable gate arrays. *Proceedings of the IEEE*, 94(2): 383–394, Febuary 2006. http://ieeexplore.ieee.org/iel5/5/33381/01580507.pdf
[102] M. Stigge, H. Plötz, W. Müller, and J.-P. Redlich. Reversing CRC – theory and practice. Technical Report SAR-PR-2006-05, Humboldt University Berlin, May 2006. http://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2006-05/SAR-PR-2006-05.pdf
[103] G. E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. In *Design Automation Conference*, pages 9–14, 2007. http://

people.csail.mit.edu/devadas/pubs/puf-dac07.pdf
[104] *[Hardware Ciphers]*, Synaptic Laboratories Ltd., November 2006. http://www.hardware-ciphers.com
[105] P. Syverson. A taxonomy of replay attacks. In *Computer Security Foundations Workshop*, 1994. http://chacs.nrl.navy.mil/publications/CHACS/1994syverson-foundations.pdf
[106] The A5 cracking project, September 2007. http://wiki.thc.org/cracking_a5
[107] A. Thompson. *Hardware evolution page*, February 2006. http://www.cogs.susx.ac.uk/users/adrianth/ade.html
[108] A. Thompson and P. Layzell. Analysis of unconventional evolved electronics. *Communications of the ACM*, 42(4):71–79, 1999. http://portal.acm.org/citation.cfm?id=299174
[109] K. Thompson. Reflections on trusting trust. *Communications of ACM*, 27(8): 761–763, 1984. http://www.cs.washington.edu/education/courses/cse590s/02sp/Reflections.pdf
[110] K. Tiri and I. Verbauwhede. Synthesis of secure FPGA implementations. In *International Workshop on Logic and Synthesis*, pages 224–231, 2004. http://eprint.iacr.org/2004/068.pdf
[111] S. Trimberger. Trusted design in FPGAs. In *Design Automation Conference*, June 2007. http://videos.dac.com/44th/papers/1_2.pdf
[112] P. Tuyls, G.-J. Schrijen, B. Škorić, J. van Geloven, N. Verhaegh, and R. Wolters. Read-proof hardware from protective coatings. In *Cryptographic Hardware and Embedded Systems Workshop*, volume 4249 of *LNCS*, pages 369–383, October 2006. http://www.springerlink.com/content/8454587207415662/fulltext.pdf
[113] Ulogic FPGA netlist recovery, October 2007. http://www.ulogic.org
[114] *High performance microchip supply*, United Stated Department of Defense, February 2005. http://www.acq.osd.mil/dsb/reports/2005-02-HPMS_Report_Final.pdf
[115] *Altera Corporation vs. Clear Logic Incorporated (D.C. No. CV-99-21134)*, United States Court of Appeals for the Ninth Circuit, April 2005. http://www.svmedialaw.com/altera%20v%20clear%20logic.pdf
[116] Virtual Socket Interface Alliance. February 2007. http://www.vsi.org/
[117] R. Wilson. *Panel unscrambles intellectual property encryption issues*, January 2007. http://www.edn.com/article/CA6412249.html
[118] R. Wilson. *Silicon intellectual property panel puzzles selection process*, February 2007. http://www.edn.com/article/CA6412358.html
[119] T. Wollinger, J. Guajardo, and C. Paar. Security on FPGAs: State-of-the-art implementations and attacks. *Transactions on Embedded Computing Systems*, 3(3):534–574, March 2004. http://portal.acm.org/citation.cfm?id=1015052
[120] Xilinx Inc. http://www.xilinx.com
[121] Xilinx Inc. Processor peripheral IP evaluation, October 2007. http://www.xilinx.com/ipcenter/ipevaluation/proc_ip_evaluation.htm
[122] D. Ziener and J. Teich. FPGA core watermarking based on power signature analysis. In *IEEE International Conference on Field-Programmable Technology*, pages 205–212, December 2006. http://www12.informatik.uni-erlangen.de/publications/pub2006/zienerfpt06.pdf

About this document

This survey was written for two reasons. Firstly, I thought that a write-up that takes a holistic approach to examine the topic of FPGA security, and that also appeals to design engineers, is missing from the literature; in many ways, I have written it to the engineer I was after starting my first job out of university. Secondly, to serve as a basis for my research. I will keep adding to this survey with new developments as they come and would appreciate comments and welcome a discussion.

Version and date are indicated on the title page and the latest version can be found at:
`http://www.cl.cam.ac.uk/~sd410/papers/fpga_security.pdf`

Previous versions are available by adding `_ddd` before the `.pdf` in the file name, where `ddd` corresponds to the version number. Links and BibTeX entries to all references, including this one and other related material, are available at:
`http://www.cl.cam.ac.uk/~sd410/fpgasec`

Revision history:

v0.95	6 Dec 2007	Added "System manufacturer" to principal list; revised Section 3.2 on reverse engineering; shuffled "Attacks" section.
v0.94	20 Oct 2007	Added Section 5.3.1 on secure processor distribution; split Figure 1 into two; changed bibliography to alphabetical order.
v0.90	10 Oct 2007	On-line release.
v0.60	22 Nov 2006	Submitted to "ACM Computing Surveys"; rejected 25 August 2007 (yes, 10 months later!)