INVITED PAPER

FPGA Security: Motivations, Features, and Applications

This paper discusses all aspects of FPGA security and trust.

By Stephen M. Trimberger, *Fellow IEEE*, and Jason J. Moore

ABSTRACT | Since their inception, field-programmable gate arrays (FPGAs) have grown in capacity and complexity so that now FPGAs include millions of gates of logic, megabytes of memory, high-speed transceivers, analog interfaces, and whole multicore processors. Applications running in the FPGA include communications infrastructure, digital cinema, sensitive database access, critical industrial control, and high-performance signal processing. As the value of the applications and the data they handle have grown, so has the need to protect those applications and data. Motivated by specific threats, this paper describes FPGA security primitives from multiple FPGA vendors and gives examples of those primitives in use in applications.

KEYWORDS | Anti-tamper (AT); authentication; encryption; field-programmable gate arrays (FPGAs); information assurance; physically uncloneable function (PUF); system on chip (SoC); trust

I. INTRODUCTION

A. FPGAs and Programming Technology

A field-programmable gate array (FPGA) is a semiconductor device that can be programmed after manufacture to perform a specific application design, typically specified as a digital logic system [43]. A taxonomy of FPGAs commonly starts with the program storage technology (Fig. 1).

SRAM-programmed FPGAs store their configuration data in internal volatile memory cells distributed throughout the device. These are generally not SRAM cells, but are more similar to static latch cells [43]. Xilinx's 7-Series and Altera's Stratix-5 are examples of popular SRAM-based FPGAs. A recognized disadvantage of SRAM programming

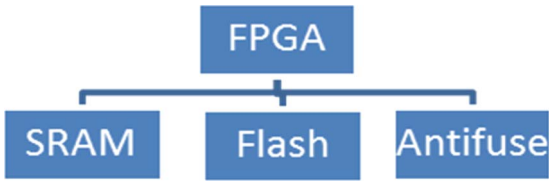

Fig. 1. ***FPGA taxonomy.***

stems from its volatility. When power is removed, the programming is lost, so an SRAM FPGA requires an external nonvolatile memory for permanent storage of the application program. When power is applied, the SRAM FPGA loads its programming bitstream from that external storage. Besides requiring a second device, the transmission of the program from the nonvolatile external memory to the SRAM FPGA may expose the programming to a potential adversary. The volatility of data may also be used as a positive security feature, enabling the SRAM FPGA to clear all programming if it is tampered [48].

In contrast, flash memory programmable logic devices, such as traditional complex programmable logic devices (CPLDs), the Microsemi Corporation (Aliso Viejo, CA, USA) SmartFusion2, and Lattice Semiconductor (Hillsboro, OR, USA) ispXPGA [1], [21], are nonvolatile and use internal flash memory to hold the programming. While the internal flash memory eliminates the need for an external nonvolatile storage device and the consequent exposure of the programming to potential adversaries, systems employing flash FPGAs commonly require in-system programming (ISP) of the FPGA. ISP exposes the programming of the FPGA to the same security concerns as SRAM FPGAs. The availability of reprogrammable flash provides FPGA manufacturers with the ability to build applications that "remember" information through power cycles—useful in cryptographic applications such as tamper logging and key revocation. Flash devices can also be erased upon command to eliminate the design when needed.

Antifuse FPGAs, such as the Microsemi Axcelerator, use a one-time programmable structure to form

Manuscript received September 14, 2013; revised May 16, 2014; accepted June 11, 2014. Date of publication July 8, 2014; date of current version July 18, 2014.
S. M. Trimberger is with Xilinx, San Jose, CA 95124 USA (e-mail: steve.trimberger@xilinx.com).
J. J. Moore is with Xilinx, Albuquerque, NM 87109 USA.

Digital Object Identifier: 10.1109/JPROC.2014.2331672

nonvolatile links between internal nodes [2], [25], [26]. An antifuse, commonly built as a programmable via between metal layers, is disconnected at manufacture. A high-voltage pulse programs the fuse, causing it to form a low-resistance connection between the internal nodes. Antifuses are nonvolatile, but one-time programmable. Once programmed, an antifuse FPGA cannot be changed or reprogrammed. Because there is no need for external configuration storage, the confidentiality and authentication of configuration data is more easily maintained. ISP is not possible. However, because the program cannot be erased from the device, additional system-level security concerns may remain.

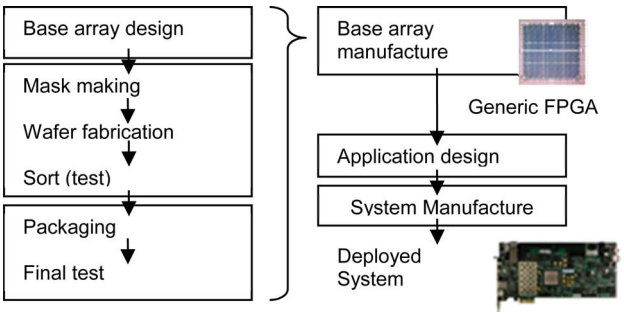

Fig. 2. *FPGA lifecycle flows. (Left) Generic integrated circuit flow for the base array. (Right) Application design and deployment flow.*

B. Why SRAM?

By far, the most common FPGAs, even in security-conscious applications, are those programmed with SRAM. If SRAM programming exposes sensitive data to adversaries, why would anyone use them? The popularity of SRAM programming technology derives from the simplicity of its manufacture: SRAM FPGAs require only transistors and wires to realize the interconnect, configuration memory cells and switches of the generic device. Therefore, SRAM FPGAs take advantage of new process nodes earlier than other FPGAs [47], which may be two process generations ahead of other technologies. This process advantage results in higher performance, greater logic density, and improved power efficiency for SRAM FPGAs. SRAM programming also simplifies manufacturing test, where the SRAM FPGA is typically programmed many times to perform self-tests. In addition, SRAM FPGA applications can be easily updated in the field in much the same way software is updated.

When used with strong bitstream security features, including those described in this paper, the security of SRAM FPGAs is on par with the security of nonvolatile internal storage of the bitstream. Therefore, despite the greater perceived security of antifuse and flash FPGAs, SRAM FPGAs are deployed in many security-conscious applications.

C. The FPGA Design Lifecycle

The FPGA lifecycle includes two design flows: the base array design and the application design (Fig. 2). Security must be maintained through both [44]. The base array design is a standard integrated circuit development flow controlled by the FPGA manufacturer. The base array is designed using commercial design tools and libraries, manufactured at a foundry and tested. It is then typically sent to another facility for packaging and final test. The resulting base array is shipped to a customer or authorized distributor. The base array design is subject to all the supply chain trust and security concerns as any other integrated circuit, including questions about tampering with tools, supply-chain control, and reverse engineering. Large FPGA manufacturers maintain a close watch on their supply chain, tracking every manufactured device through to final customer delivery or destruction. As the security issues associated with the design and manufacture of the base array are no different than those of other semiconductor devices, this paper does not focus on the base array design and manufacture, but instead focuses on the security concerns that arise from the need to protect the application design.

The application design also has a design phase, typically performed with FPGA vendors' tools, often augmented with commercial EDA tools. The application developer integrates design information or intellectual property (IP) from a number of sources into an FPGA application: original and reused hardware description language (HDL) code, libraries from the FPGA vendor and other parties and software for soft and hard microprocessors. The FPGA vendor's tools compile the application design into a bitstream, the programming of the FPGA base array to realize the application function. As with any design process, the design itself can be carried out in a secure location. Protection of IP during the design phase is no different for FPGAs than it is for ASICs or microprocessors. Therefore, this paper does not address design-phase security.

A nonvolatile FPGA, such as a flash or antifuse FPGA, may be programmed before it is shipped. An SRAM FPGA is typically shipped with a separate nonvolatile memory containing the programming, and when power is applied, the FPGA loads its programming from the nonvolatile memory.

D. This Paper

This paper begins by focusing on those FPGA aspects that impact security, both positively and negatively. It summarizes the common threat vectors and then introduces some early FPGA security strategies. The remainder of the paper focuses on modern FPGA security as it relates to two of the primary security domains: information assurance (IA) and anti-tamper (AT). In each domain, the presentation describes the techniques that are currently deployed, introducing them broadly, then using specific threats to motivate additional detail. The various FPGA

vendors have chosen solutions to these threats that are similar, yet they differ in detail. In this paper, we attempt to describe the major security solutions deployed by large FPGA vendors, outlining major distinctions while omitting minor differences. Since FPGA security is continually changing, newer FPGAs may well deploy different mechanisms.

Following the discussion of security features is a section showing applications using those features to achieve security goals. This paper concludes with a short discussion of the future of FPGA security capabilities.

II. FPGA SECURITY INTRODUCTION

A. Unique Aspects of FPGA Security

FPGA programming bitstreams are qualitatively much like microprocessor software. They are susceptible to all the same security concerns that surround software, including unauthorized copy, theft of IP embodied in the FPGA application program, and tampering to introduce malware [9], [46]. FPGA programming is present in the system in the field, whether programmed directly in antifuses, flash memory cells, or in an external nonvolatile memory. If an adversary can recover the programming by reading the internal memory, intercepting the programming bitstream, or reverse-engineering programmed fuses from a decapped device, then the application can be duplicated and reverse engineered. SRAM FPGAs, in particular, have been criticized over this concern [2], although Flash-based FPGAs have the same susceptibility if in system reprogrammability is required.

On the other hand, the application developer does not reveal the application design to FPGA vendors or their suppliers. Because the FPGA base array is manufactured without knowledge of the end application, there is no chance of IP theft or tampering of an application design during manufacture and test of the FPGA base array. Since all FPGA devices are manufactured identically and sold into a variety of applications, an adversary cannot discover any application-dependent information by attacking the FPGA vendor's supply chain.

Further, since the programming is not done with metallization as is the case with ASIC devices, traditional reverse engineering, where the mask layers are recognized from a decapped device, does not work. Such reverse engineering may yield the application-independent base array, but not the application implemented on it.

B. Environment and the Cost of Security

FPGA security is complicated by the environment in which the FPGA is expected to perform. The design of FPGA security features assumes no physical barrier and no communication network: the FPGA may be in the hands of an adversary with no trusted party available. This environmental assumption distinguishes FPGA security from internet security, where servers may physically reside in a trusted environment and those servers can verify identity through name servers with which they are in communication. In FPGA security design, it is assumed that the adversary has physical access to the device and may mount any electrical, physical, side channel, or replay attack. The rationale is straightforward: if the adversary does not have such access, then the containing system could ensure the security of the FPGA by controlling all access to the FPGA. In this case, built-in FPGA security would be redundant.

Although military systems may employ physical security, the cost of "guns, gates, and guards" is impractical in commercial systems. The adversary is assumed to have an economic motive, such as theft of IP. Therefore, the security applied in the commercial domain is an economic concern where the cost of security measures is balanced against the value of the information being protected. FPGA security is designed to make the cost of breaking the security greater than the adversary's expected economic gain. This decision is ultimately in the hands of the application developer, not the FPGA manufacturer.

As FPGAs have become larger and more capable, the value of the IP of the application designs has grown, motivating significant investment in built-in security functions. Further, the value of the data handled by the FPGA has also increased significantly, including such information as decrypted digital cinema and personal-data databases. As a result, today we find FPGAs deployed in a security-hostile environment, protecting data of great commercial value.

III. THREATS

An adversary may attack the IP of the application design itself, the data stored in the application or the system of which the FPGA is a part. Each type of data has different value. Each attack requires different security features to defend. The attacks of major concern to FPGA vendors can be divided into categories.

A. Cloning/Overbuilding

In cloning, an adversary copies the FPGA programming, then uses it in an identical device, selling it as his own. In overbuilding, an adversary such as a contract manufacturer builds additional systems, inserting the legitimate bitstream into those systems and selling them without the designer's approval. Cloning may apply to an entire design or may apply to a subset of the design, for example, purchased cores that may be restricted by the seller. In both cases, the adversary does not require detailed knowledge of the design.

B. Reverse Engineering

An adversary may reverse engineer the bitstream to recover the circuit design that it implements. This may be done to understand and duplicate the functionality of that

application, but may also be used as part of an attack on other aspects of the system. Reverse engineering may be used to tamper with the application to insert malware. Historically, reverse engineering an FPGA bitstream, like decompiling software, has been considered possible, though tedious and nontrivial. Reverse engineering of FPGA bitstreams is further complicated because FPGA vendors do not have a standardized bitstream. As a result, every new FPGA device requires a new bitstream reverse-engineering effort.

A more insidious problem is dealing with the size of the application. Although reverse engineering may divulge the netlist of the application, transforming a multimillion gate netlist into an understandable design that can be modified is problematic. The complexity of the application increases its value, making theft attractive, but the consequent size makes theft difficult.

Regardless, researchers have periodically reported the ability to reverse engineer unencrypted bitstreams. It would seem imprudent to rely on the tedium of bitstream reverse engineering to protect valuable IP.

C. Tampering

In tampering, an adversary modifies an application design. Tampering may be employed to add logic that leaks information from an application or tampering may disable parts of the application, potentially defeating other security measures. For the former, tampering must control the application to set values in the bitstream, so reverse engineering may also be required. However, for the latter, merely scrambling parts of the bitstream may be sufficient.

D. Spoofing

In spoofing, an adversary replaces the FPGA bitstream with his own. That bitstream may or may not include components derived from cloning or reverse engineering. A spoofed application may compromise the system in which it operates.

E. Denial of Service, Destruction of the FPGA, and Substitution

Since it is assumed that the FPGA is in the hands of an adversary, denial of service and malicious destruction of the FPGA device are somewhat irrelevant. Rather than mount a clever attack on the design to prevent the system from operating, an adversary could simply smash the FPGA with a hammer. Conversely, if a system requires an FPGA containing a unique key, an adversary may choose to circumvent security measures by replacing the FPGA in a system with another identically manufactured device from the FPGA vendor without the key or with his own key. In many cases, this substitution is simpler than attempting to break the FPGA device security. Since these physical attacks are so simple, FPGAs typically do not defend against these types of threats.

IV. HISTORICAL FPGA SECURITY

Early FPGAs contained very little logic, and by inference that logic had low value. Therefore, when they were introduced, FPGAs provided only rudimentary protection against threats.

FPGA manufacturers did not release the coding of their bitstreams, though they did release a considerable amount of information about the bitstream in tools and documentation [50]. They considered the task of reverse engineering the bitstream to be more expensive than the task of re-creating the design by black-box observation of its operation.

A. Readback

From their inception, FPGAs of all types included a readback mechanism, whereby the program and data in the device can be read out for test purposes. To prevent unauthorized copy, early FPGAs followed the features of programmable logic devices (PLDs) and included a programming bit to disable the readback mechanism. This method worked well for antifuse and flash-based FPGAs, where the program could be loaded at a secure location, but SRAM FPGAs still needed to load the bitstream in the field, while potentially in the hands of an adversary. Preventing readback gave little protection if the bitstream could be intercepted as it was loaded into the FPGA. For this reason, antifuse FPGAs, that did not expose the programming in the system, gained an early reputation for being a more secure FPGA technology.

It is important to note that the readback function has, and continues to be, a valuable feature for both the FPGA manufacturer and the user. Whether the manufacturer uses it for device test, or the user employs readback for in-system data integrity checks, it is a feature, much like JTAG, that is useful but needs to be adequately protected to avoid vulnerabilities.

Readback continues to be a concern, and as late as 2012, Skorobogatov and Woods [38] discovered a keyed back-door/test mechanism that enabled the readback feature of a Microsemi antifuse FPGA that was assumed to be protected by the FuseLock protection mechanism [27].

B. Early Bitstream Protections for SRAM FPGAs

Before bitstream encryption, two methods were used to protect SRAM bitstreams. The first method was to load the FPGA at a secure location and use a battery to hold the configuration bitstream for the entire lifetime of the fielded system [3]. Since programmable logic devices had privacy settings to prevent readback of the program, and since the bitstream was never exposed outside the device, this method assured that the bitstream running inside the FPGA is both secure and unmodified. This is precisely the same level of security achieved by antifuse and other nonvolatile FPGAs. The drawback of this method is, of course, the requirement that the system be powered continually. As FPGAs grew larger and more complex, this

solution became impractical due to increased standby power requirements.

The second solution was to use an external memory with a unique identifier, and customize the FPGA program to require that identifier, essentially tying the FPGA application bitstream to a unique board-level identifier. An improvement to the simple test of the board-level identifier uses an external keyed device that is queried with a random number generated by the FPGA [6]. This solution defeats simple cloning because the bitstream only functions correctly in the system with the correct external identifier device. However, the bitstream for each system is unique, which complicates the manufacturing process. Further, the design can be copied by an adversary who reverse engineers the bitstream, identifies the check logic, and rebuilds the application with the check removed.

This solution increases the difficulty and cost of copying the design but still relies on the difficulty of reverse engineering the bitstream as the basis of security. This solution was considered strong enough for many commercial applications, and there is no evidence that anyone mounted a successful attack on a device protected with it. However, reliance on the tedium and complexity of bitstream reverse engineering seemed risky [49].

C. Modern FPGA Security

As FPGAs grew in capacity, the applications grew in value, driving the need for stronger security. Over the years, FPGA vendors have implemented circuitry, software, IP cores, and usage models to address security threats. Since the FPGA application design is embodied in a design file, aspects of information security, notably encryption and authentication, were applied to FPGA bitstreams. But that was not enough. Given that FPGAs were deployed into a hostile environment, measures were taken also to improve protocols and implementations to secure designs in the field. These include not only cryptography on the configuration files but also development of fault-tolerant design methodologies for the base array and for applications. Today, FPGA security is strong enough that they are deployed in security-sensitive applications in commercial and government systems [24].

V. INFORMATION ASSURANCE

The basic tenets of information assurance (IA) are: confidentiality, integrity, availability, authentication, and nonrepudiation. As mentioned earlier, since access to the FPGA is assumed, availability is not a requirement addressed by FPGA security features. Nonrepudiation will be addressed in the context of authentication. Therefore, we focus on confidentiality, integrity, and authentication.

A. Confidentiality

Large FPGA designs can contain IP of significant value, and bitstream encryption prevents a competitor from simply copying that IP. Encryption can also provide trust assurance by limiting access to the FPGA only to designs constructed with the proper key.

1) Overview of Bitstream Encryption: Xilinx (San Jose, CA, USA) introduced bitstream encryption in 2001 in Virtex-II devices [40], [41] to address the problem of unauthorized copy of the bitstream as it is loaded into the FPGA from external memory. Since that time, other FPGA vendors have added encrypted-bitstream capability.

Preventing unauthorized copy does not strictly require encryption, since the task from a cryptographic point of view is to determine if the bitstream is authorized to operate in the FPGA. This fundamentally requires authentication, not confidentiality: a device could verify a message authentication code on the bitstream. However, the adversary's workaround is simple: reverse engineer the bitstream, recompile, and load it into a new FPGA with the authentication removed. Therefore, reverse engineering must also be prevented, so confidentiality of the bitstream becomes a requirement for preventing cloning.

Virtex-II FPGAs used triple-Data Encryption Standard (DES) encryption and subsequent Xilinx FPGAs use 256-b Advanced Encryption Standard (AES). Recent SRAM devices from Altera Corporation (San Jose, CA, USA) [4] and Flash devices from Microsemi Corporation (Aliso Viejo, CA, USA) [28] also use 256 b AES. Lattice Semiconductor (Hillsboro, OR, USA) devices use 128 b AES [19]. Although features have changed over the years, and details vary among vendors, the basics of FPGA bitstream encryption for all SRAM and Flash FPGAs are similar. The major components and use flow are described here with respect to the Xilinx, Inc. (San Jose, CA, USA) 7-series FPGA.

An application developer prepares a secured FPGA application with the same tools and processes used for any other application. At the end of the design process, when the bitstream is generated, Xilinx proprietary software encrypts the bitstream. The Xilinx software can supply a randomly generated key and initialization vector or the application developer may supply those values. The Xilinx software produces the encrypted bitstream and a key-insertion file.

2) Key Loading: At a secure facility, the application developer uses the key-insertion file to load the decryption key into the FPGA through the JTAG scan chain, as shown in Fig. 3. On-chip, the key is stored in either dedicated nonvolatile or volatile memory. FPGAs supply an independent battery-backed array for volatile storage or one-time-programmable eFuses for nonvolatile storage or both. Typically, the key is loaded into the FPGA in plaintext form, which is why this must be done at a trusted facility. Alternative strategies for key loading and key storage are discussed in Section VI-A2.

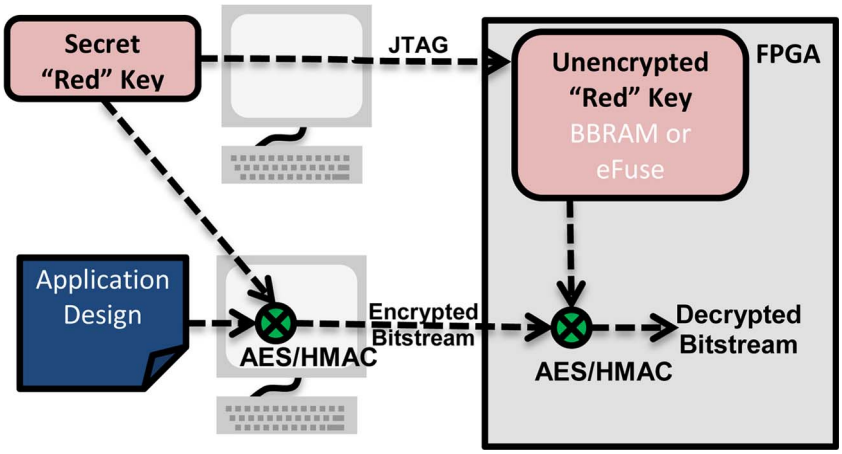

Fig. 3. *Encryption architecture for Xilinx 7-series FPGAs.*

3) Bitstream Loading: Later, in the field when the FPGA board boots, the FPGA loads its bitstream from an external memory. The FPGA begins loading an unencrypted bitstream. If the bitstream includes an encrypted-bitstream indicator, the FPGA starts the decryptor and decrypts the remainder of the bitstream as it loads. If the encrypted-bitstream indicator is not present, the FPGA bypasses the decryptor. This feature allows an FPGA in the field to be booted with either an encrypted bitstream or an unencrypted bitstream for test purposes without compromising the security of the bitstream confidentiality. In addition, most FPGAs now offer the ability to force the device to always configure with an encrypted bitstream.

B. Data Integrity

Bitstream data integrity, the ability to ensure a design has not been accidentally modified, was a feature of very early FPGAs. In those early devices, an improperly programmed FPGA might enable two large internal drivers in contention, generating excessive heat and current, damaging the chip. To prevent this, data integrity checks were added to FPGA bitstreams to detect corruption of the bitstream during loading. Cyclic redundancy check (CRC), a common data integrity check in data transmission protocols, was deployed in many FPGAs. While CRC is effective in detecting accidental data corruption, it is ineffective against intentional data modification.

1) Tampering With Encrypted Bitstreams: Xilinx FPGAs use 256 b AES encryption [11] in cipher block chaining (CBC) mode of operation [33] to produce a stream cipher. In CBC encryption, each block of data is first XORed with the ciphertext of the previous encryption before being encrypted. In decryption, the decrypted plaintext of each block is XORed with the ciphertext of the previous block (Fig. 4). CBC causes blocks with identical plaintext (for example, all zero) to encrypt to different ciphertext, thereby eliminating a dictionary attack on the data. Altera devices use AES in counter mode (CTR) [30]. In CTR mode, an encryptor encrypts the output of a counter to generate a pseudorandom stream of bits. That pseudorandom stream is XORed with the plaintext to generate ciphertext. On decryption, an encryptor generates the same pseudorandom stream to recover the plaintext.

CBC and CTR are non-error-extension modes of operation, meaning that corruption of the encrypted data causes only a localized corruption of the corresponding plaintext. Therefore, both CBC and CTR permit a "bit-flipping" attack on the plaintext. The attack is shown in Fig. 4 with respect to CBC. If an adversary inverts a bit in the first encrypted block, as shown by the shaded area, the first block will decrypt to unintelligible nonsense. However, the corresponding plaintext bit in the next decrypted block is inverted. Bit-flipping CTR mode is more straightforward, since a bit flip anywhere in the ciphertext inverts the corresponding bit in the decrypted plaintext without disrupting any other data.

Using this bit-flipping technique, an adversary can selectively invert any number of bits in the decrypted bitstream. If the location and state of the target bit are known, an adversary can set it. For example, if the logic to enable bitstream readback is disabled with a "0" at a specific location, an adversary could reenable bitstream readback without knowing the contents of the bitstream by inverting that one bit. For this reason, disabling of readback of an encrypted FPGA bitstream is not controlled by bits in the encrypted bitstream itself, but is instead controlled by the configuration logic of the FPGA. When the FPGA loads an encrypted bitstream, readback is disabled regardless of the bitstream contents. However, other attacks may attempt to modify the FPGA in a simple way: enable the internal configuration access port (ICAP), enable input/output (I/O) blocks, or change clock speed in an attempt to gain access to internal data.

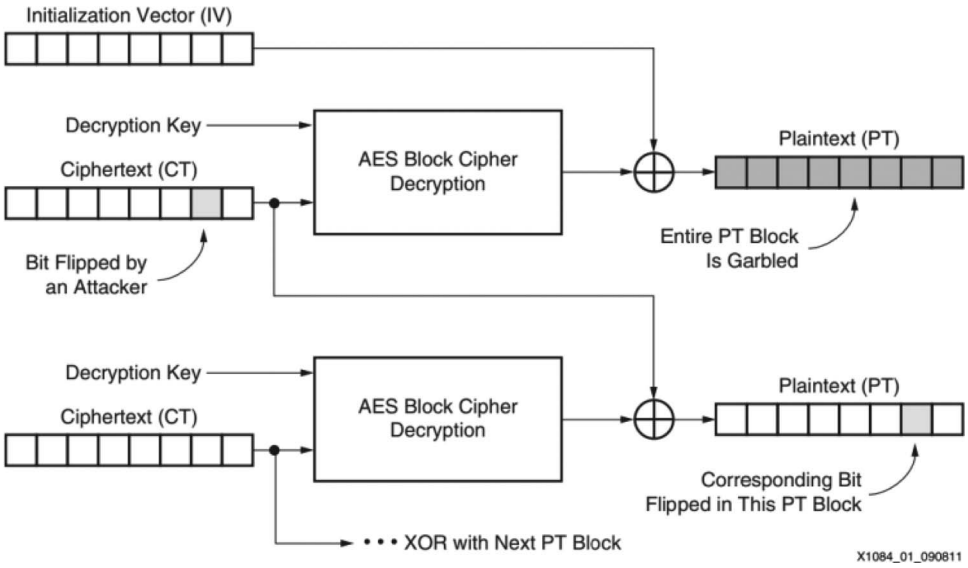

Fig. 4. ***Bit-flipping attack on CBC mode.***

Attacks on the bitstream are also possible without knowing the specific bit to attack. In Fig. 4, the first block of data is scrambled. An adversary does not know the plaintext that results from modifying the ciphertext. If the number of bits to control is small enough, an attacker with patience may attempt a brute-force attack on part of the bitstream. Scrambling the bits may program the FPGA to perform a function it was not supposed to, such as leak-sensitive information.

Checksums and CRCs on the FPGA bitstream detect errors in transmission, corrupted bitstreams, and unintentionally flipped bits. However, it is computationally straightforward to compute a revised CRC after tampering with the bitstream or to determine a set of bit flips that produce the same CRC value. Further, a CRC is typically only 16 or 32 b, so brute-force attacks on the CRC are tractable. Finally, in some FPGA architectures, CRCs can be disabled altogether. Simple data integrity checks are not sufficient to ensure that a bitstream has not been intentionally tampered.

C. Authentication

Communication of the bitstream to the FPGA is a one-way transfer. Therefore, two-way entity authentication cannot be performed. Instead, FPGAs rely on one-way message authentication, which assures the recipient of a message that the message is exactly the message the sender intended [8]. Strong authentication requires a message authentication code (MAC), a cryptographic hash function computed over the entire message. The hash function must be impossible to compute without knowing the plaintext of the message. The difficulty of recomputation of the MAC eliminates all forms of CRC as the hash function, since each bit of the CRC is a known XOR of a set of bits of the message.

Because authentication verifies that the application has not been accidentally or intentionally altered, it assures trust in the running application. That trust enables an application developer to guarantee protection of cryptographic services and the handling of sensitive data. These sensitive data may be customer data of high value, such as personal data in a database or copyrighted video. The cryptographic services may include key management functions, encryption/decryption algorithms, or keys for further partial reconfiguration of the FPGA. Data authentication provides a strong root of trust, allowing an initial FPGA configuration to act as a trusted boot loader for trusted subsequent configuration of the FPGA.

Xilinx integrated strong data authentication in Virtex-6 devices and 7-series to address the concerns of targeted tampering with encrypted bitstreams and the inherent cryptographic weaknesses of CRC. Microsemi also has a dedicated data integrity check for all of the nonvolatile configuration memory segments of some Flash devices [28]. Authentication is described here as it is implemented in Xilinx devices.

1) Data Authentication in Xilinx Virtex Devices: Virtex-6 and subsequent Xilinx FPGAs authenticate using the secure hash algorithm (SHA-256) to compute a 256-b keyed hashed MAC (HMAC) [12], [13], [42]. SHA-256 is a one-way hashing algorithm with a compact hardware implementation. The keyed HMAC requires a secret authentication key included in the hash. The MAC result cannot be computed without knowing the key, thereby authenticating the identity of the sender as well as verifying that the message has not been altered. The 256-b hash size ensures that any tampering with the bitstream will be detected with a high probability. HMAC with

SHA-256 makes tampering with the bitstream as computationally difficult as guessing the encryption key, which is also 256 b.

2) Integration of Authentication With Bitstream Encryption: Virtex devices use generic composition of the SHA-256 keyed HMAC authentication with AES-256 encryption [34], [37]. Generic composition allowed the two parts to be separated, which permitted them to be developed independently and separately pipelined.

Virtex-6 and 7-series authentication and encryption are composed using authentication then encryption (AtE). The HMAC is computed on the plaintext, unencrypted bitstream. The configuration data and the MAC result value are then encrypted. On the FPGA, the data are first decrypted and the MAC result is recomputed on the decrypted data and compared with the transmitted value in the bitstream. If the two MAC values disagree, the FPGA configuration fails and the FPGA does not become active. The authentication check catches errors in transmission and attempts to configure the FPGA with the incorrect key value as well as intentional tampering.

3) The Authentication Key: HMAC requires a secret authentication key in addition to the decryption key [12]. When generating an authenticated encrypted bitstream, both keys are specified to the bitstream generation software. To save nonvolatile storage space, only the decryption key is stored in the FPGA array. Because of the AtE composition, the encrypted authentication key can be transmitted with the bitstream. The bitstream encryption provides the privacy to keep the authentication key secret.

4) Authentication Using Public Key Cryptography in FPGAs/SoCs: The recent introduction of programmable systems on chip (SoCs) from FPGA manufacturers, including the Xilinx Zynq and Microsemi SmartFusion2 devices, have brought public key cryptography to the programmable logic market. Both of these devices use asymmetric cryptography to provide authentication during the secure boot process. The public key is stored on-chip in nonvolatile memory and its integrity checked before use. Public-key authentication of configuration files such as a first stage boot loader (FSBL) detects random-data attacks such as those commonly used for side-channel attacks. It can also serve to provide nonrepudiation of protected applications.

D. Bitstream Structure

Fig. 5 compares bitstream structures of representative Xilinx FPGA families, each with different security capabilities [42]. Fig. 5(a) shows the bitstream format of an unencrypted bitstream for Virtex devices. The unencrypted bitstream structure starts with a synchronization word (SYNC) followed by a sequence of instructions. Header commands set registers and control a variety of functions, including declaring the device type and setting up the startup sequence. The available commands and registers are described in the Configuration User Guides for each device family [50], [51]. The Write Frame Data Register Immediate (Write FDRI) command begins streaming the configuration data to the FPGA's configuration memory. An unencrypted bitstream can contain any number of Write FDRI commands, each writing a different, possibly discontinuous, portion of the FPGA configuration memory. Footer commands allow setting of register values after loading configuration data. CRC verifies data integrity and STARTUP begins the FPGA startup sequence. DESYNC prepares the configuration logic to accept postconfiguration reconfiguration commands.

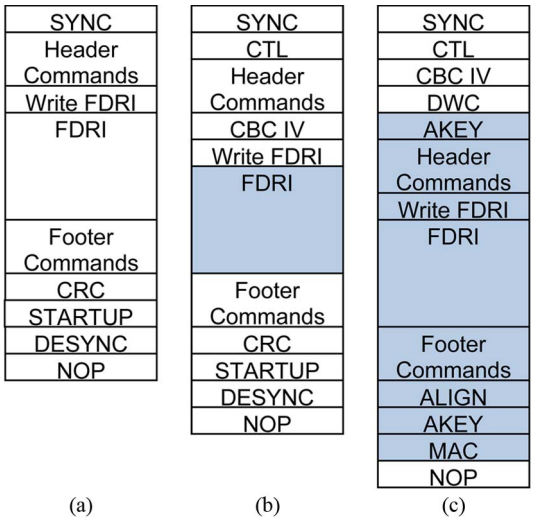

Fig. 5. *Xilinx bitstream structure. (a) Unencrypted. (b) Virtex-5. Shaded area is encrypted. (c) Virtex-6/7-series. Shaded area is authenticated and encrypted.*

Virtex-II through Virtex-5 FPGAs allowed encryption of the FPGA configuration data, but not authentication. As a representative of those encrypted-only bitstreams, Fig. 5(b) shows a Virtex-5 encrypted bitstream structure. The CTL instruction informs the FPGA that this is an encrypted bitstream. If the CTL command is missing, the FPGA assumes the bitstream is unencrypted. CBC IV is the initialization vector for the AES CBC register. The CBC IV does not need to be secret, and it is evident in the bitstream structure that it is set with an unencrypted header command. The Write FDRI command passes encrypted configuration data through the decryptor. The Write FDRI command includes a length field, also transmitted unencrypted, so the decryptor decrypts the proper amount of data. Only the configuration data are encrypted, although the CRC is computed on all data that precede it in the bitstream.

Fig. 5(c) shows an authenticated encrypted bitstream from Virtex-6 and 7-series devices. Authentication and encryption are always used together. There is no way to specify a bitstream that is only encrypted or only

authenticated for these devices. As in earlier devices, the CTL instruction informs the FPGA that the bitstream has security enabled. The CBC initialization vector initializes the decryptor as before. Decrypt word count (DWC) indicates to the FPGA the amount of secure data to follow. DWC includes not only the configuration data but header and footer commands as well. Header commands and footer commands are encrypted and covered by authentication. DWC is transmitted in the clear and could be modified by an adversary, but since the length of the data is included in the MAC computation, a modification of DWC will invalidate the computed MAC.

The authentication key is transmitted to the FPGA at the start of configuration and again at the end of configuration, since the key is used twice in the HMAC computation. ALIGN is a variable number of no-operation instructions, inserted to ensure the authenticated encrypted data are an even multiple of 512 b, simplifying the MAC computation. At the end of the bitstream, the required MAC is transmitted to the FPGA where it is compared with the MAC computed by the FPGA.

Confidentiality, data integrity, and data authentication of the configuration data are all required to protect FPGA configuration data that are exposed to potential adversaries. To date, only a few devices available from Xilinx and Microsemi provide all three protections on their configuration files.

VI. ANTI-TAMPER

Physical security of the FPGA is just as critical as the application of confidentiality, integrity, and authentication to the device configuration. While there are focus areas of AT that overlap with IA, there are also aspects of AT that are unique. FPGA manufacturers are faced with a number of challenges while focusing on improving the physical security of the device. As commercial products, some of the challenges include, but are not limited to:

- FPGAs are readily available for adversaries to experiment on;
- compliance with U.S. and worldwide export and import restrictions—manufacturers must be able to sell their product worldwide, and do so while meeting all import/export laws;
- FPGAs are cost sensitive, requiring a careful balance between protecting customers' IP and enabling FPGA use in all types of systems.

There has been significant investment by FPGA manufacturers to enhance the physical security of their devices, driven primarily by the continual growth in performance, density, and capabilities. This puts FPGAs at the heart of most electronic systems today, where customers' IP must be protected.

This section describes some of the primary security features and protocols of the Xilinx 7-series FPGA. These are explored by looking at the configuration lifecycle of the device. AT protections are employed preconfiguration, during configuration, and postconfiguration.

A. Preconfiguration

1) Defense Against Trojan Insertion: FPGAs allow the ability to configure either encrypted or unencrypted. This is useful for application developers who may not want to use encryption during integration and test, but then enable encryption when the system is fielded. This ability to configure either encrypted or unencrypted, subjects the device to a class of Trojan insertion attacks.

If an FPGA contains a decrypted bitstream, an adversary may attempt to load a partial configuration into a subset of the device that spies on the resident application. It could connect to internal signals or memories. By connecting to internal components, the Trojan could be used to deduce the secured application in the FPGA. Conversely, an adversary may operate the same attack by preloading a Trojan design and interrupting the secure loading of the protected application.

Consequently, Xilinx FPGAs do not permit mixing encrypted and unencrypted bitstreams, or partial bitstreams, in any order. A new configuration of the device requires fully clearing the existing device, either by cycling power or executing the JTAG JPROGRAM command. Both methods initiate internal device housekeeping, which clears all configuration and internal memory.

Similar concerns exist today with SoCs being introduced by the FPGA manufactures. Xilinx, Altera, and Microsemi are now offering processor-centric SoC devices that typically have separate and independent regions for the processor subsystem and the programmable logic. The independence provides users flexibility and the ability to significantly reduce power by turning off the programmable logic. This capability presents vulnerability. If an adversary can preload a Trojan, either into the processor memory or the programmable logic before allowing the device to boot normally, then the Trojan will have access to the entire internal application running on the device. As with Xilinx FPGAs, Xilinx SoCs have been designed to address this security concern. The Xilinx Zynq device boots both the processor and the programmable fabric from the same root of trust, either fully secured, or fully open.

The Trojan insertion vulnerability also exists postconfiguration. Xilinx and Altera FPGA families permit partial reconfiguration, the ability to change the configuration of a section of the FPGA while the rest operates normally. This feature has proven to be very valuable for innovative applications. However, it also is susceptible to a Trojan insertion attack after the initial configuration. The application design must authenticate postconfiguration bitstreams to exclude Trojans.

2) Protecting Keys: The secrecy of a cryptographic key is fundamental to security; protecting the key is the first

priority of the FPGA manufacturer. As stated earlier, this can be a challenge for commercial vendors who are developing a device that is used in nearly all types of applications, and not specifically designed for a specific domain, application, or cost point. Also important to note is the fact that while we address the protection of keys here in the preconfiguration section of this paper, protection of keys is essential before, during, and after configuration.

a) Key storage—Technology: Most FPGA manufacturers provide both volatile and nonvolatile key storage. In the case of SRAM FPGAs, volatile key storage is implemented as battery-backed RAM (BBRAM) and nonvolatile key storage is implemented as eFuses. Each has advantages and disadvantages.

BBRAM key storage requires no process changes, making it easily implemented in state-of-the-art process technology. The volatile key storage also allows for key agility and key zeroization, critical components of a strong cryptographic system. In Xilinx FPGAs, when primary power is applied, the BBRAM is powered by that power supply, which not only reduces the drain on the battery but also permits replacing the battery in fielded system. Altera specifies that a battery must be attached before the key is loaded, implying that the source for the BBRAM memory is only the external battery [4]. Xilinx also provides an internal interface that can be used by an application to command a zeroization of the key space. Zeroization is intended for use when the FPGA detects tampering with an operating application.

BBRAM is not without its disadvantages. A momentary loss of contact or low battery voltage could cause the key to be lost. While modern coin-cell batteries hold enough energy to hold encryption keys for the design lifetime of 20 years, and new betavoltaic batteries with great reliability are being introduced to the market, many battery vendors do not specify thermal wearout or other failure modes, for the length of time required by most FPGA users.

BBRAM is inherently more physically secure than nonvolatile key storage technology. To steal the key, an adversary would need to decap the FPGA and mill away many levels of metal, then scan the bits with a scanning electron microscope (SEM). This attack must be performed while keeping clean power to the key memory. This is the type of attack required to extract the entire configuration directly from the FPGA SRAM cells as well, so no bitstream encryption method is qualitatively stronger. This attack is considered to be beyond the capabilities of all but the most sophisticated of adversaries.

An eFuse provides a simple, one-time-programmable nonvolatile memory. Because they are nonvolatile, eFuses eliminate the maintenance issues associated with a battery. A common eFuse structure is a narrow wire that is programmed by electromigration from high programming current. eFuses are simple to build and program, requiring no additional process complexity or high voltage. However, eFuses and their programming circuitry are rather large, so eFuses are practical only for small amounts of memory, such as a decryption key. The physical change caused by eFuse programming is visible under a microscope, so eFuses are comparatively easy to reverse engineer from a decapped part. Of course, they cannot be reprogrammed or erased. However, to zeroize an eFuse key, one could burn all eFuse cells in the key.

b) Key loading: The JTAG test port is a common interface for loading keys into programmable logic devices [4], [22], [35]. Loading a key into a Xilinx device begins by first executing a JTAG command to enter key access mode, which clears the existing key and all configuration data and memory in the FPGA. A second JTAG command writes the new key and reads it back to verify it. Of course, on power-up, FPGAs key access is disabled.

Details of key loading vary considerably among manufacturers. Loading of the key may be done in plaintext ("red key load") or ciphertext ("black key load") or otherwise obscured. In Xilinx devices, the key is transmitted to the FPGA in the plaintext, so it must be loaded in a secure location. The key access control sequence ensures that the key is cleared before any command is executed that could read it back. Other vendors have chosen alternative solutions. Altera Stratix devices include a key obfuscation mechanism so the key may be presented to the FPGA in an encrypted form. Moradi [32] reported that two 128 b keys are used by Altera Stratix-II devices. The bitstream key is transmitted and stored in encrypted form, encrypted by a second key, which is presented without any obfuscation. Although the key used to decrypt user data is not transmitted or stored in the FPGA, and hence cannot be extracted, it can be computed by a straightforward algorithm from the readable key that accompanies it. In Altera Stratix-V devices, the user key is sent through a one-way function before being stored on the device [4]. In both of these scenarios, the loading of the key is obscured, and while not cryptographically sound, may provide a level of security acceptable at a given price point.

Microsemi is the first FPGA manufacturer to offer a true "black-key load": the key is encrypted by a secret key before loading. In selected SmartFusion2 devices, the device and user exchange public keys and perform an elliptic curve Diffie–Hellman (ECDH) exchange to generate a key that can be used for the authenticated/encrypted loading of a user key [28]. The generated key is used as a key encryption key (KEK) to encrypt the user key on the transmit side, and to decrypt the user key within the SmartFusion2 device.

c) Key storage—red or black?: Much like key loading, the device key can be stored in plaintext, ciphertext, or obfuscated form. Xilinx stores 7-series keys in plaintext form. An adversary who decaps the part and can identify the key storage cells can attempt to extract the actual key bits. An obfuscated key defeats this attack until the obfuscation method is discovered. Altera implemented a key obfuscation algorithm in Stratix devices, so that probing the device could not divulge the key directly. When the

obfuscation algorithm was revealed, obfuscation was no longer a barrier to invasive key extraction [32].

Selected SmartFusion2 devices from Microsemi make use of Intrinsic-ID's (Eindhoven, The Netherlands) Quiddikey technology [17]. This technology does not store an encryption key on-chip. Instead, it generates the key when needed through the use of an activation code generated during an enrollment phase, and the output of Intrinsic-ID's SRAM-based physically uncloneable function (PUF) [17], [29].

d) Eliminating keys: When an unauthorized event occurs, the application may need to eliminate sensitive keys within the device. For systems that employ BBRAM key storage, there are multiple options. First, passive erasure can be accomplished by simply electrically disconnecting the battery from the supply. Second, for Xilinx devices, an external device could send the appropriate JTAG command to enter key access mode. As mentioned earlier, this actively clears the device key and the configuration of the device. Finally, most FPGA vendors have the ability to erase the key from within the device under control of the application [4], [29], [38]. Xilinx and Microsemi offer the ability to fully zeroize the device key, actively erase the key, and then verify that it indeed has been erased, either through readback or dedicated hardware.

3) Antispoofing: When they are manufactured, FPGAs can accept either an unencrypted bitstream or an encrypted bitstream. All programmable logic vendors that provide encrypted bitstreams have the ability to modify the FPGA to require an encrypted configuration. This modification involves programming a nonvolatile eFuse register that disables unencrypted configuration. An adversary cannot substitute an alternative bitstream in the device or change the key. Instead, that adversary must replace the FPGA with another equivalent device. This solution provides no value with a BBRAM volatile key, as the adversary only needs to remove the battery to clear the key, then load a new key into the device to gain access. Of course, an adversary can circumvent the antispoofing by replacing the protected FPGA with a new, unprogrammed one. Nonetheless, antispoofing the device is a cost-effective component of overall system security.

4) Test Circuitry: Because it provides access to and control of internal nodes, test circuitry has long been a primary point of security vulnerability in integrated circuits, and must be disabled for a secure application to indeed be secure. While protection of test circuitry is discussed in this preconfiguration section, it must be considered during configuration and postconfiguration as well.

Test interfaces can be disabled in many ways. Proprietary test interfaces are typically handled differently than industry-standard interfaces such as JTAG. Xilinx disables readback by setting internal security bits when an encrypted bitstream is loaded. In the Zynq SoC, eFuses may be used to disable test interfaces permanently [36]. Test disable is also provided in Altera devices where a tamper-protection bit disables the test modes of the FPGA [4]. When permanently disabling test circuitry, users must be aware of the consequences for additional failure analysis: if the test access port has been disabled, there is very little anyone can do to debug the device.

Microsemi and Xilinx provide mechanisms to permanently disable the JTAG interface as well as monitor it internally for tamper conditions [29], [35]. Altera has the ability to reduce the number of JTAG commands executed to only those mandatory by the standard (e.g., Extest, Intest, IDCODE, etc.). The execution of nonmandatory JTAG instructions can be enabled by issuing the UNLOCK JTAG instruction, which is only allowed to execute when sent from within the device [4].

B. During Configuration

1) Side-Channel Attacks on Keys: In recent literature, Xilinx, Altera, and Microsemi FPGAs have been shown to be vulnerable to differential power analysis (DPA) attacks on their keys [30]–[32], [38]. Although noninvasive, these published attacks employ a custom board with a significant reduction in bypass capacitance in order to enhance the power signal. This brings up the question of the difficulty of moving an FPGA from one board to another while keeping the key intact. eFuse, antifuse, and Flash storage should be unaffected, but battery backed RAM keys are lost if, during the transfer, power is lost to the keys or if the device temperature exceeds operating limits.

Security is always a moving target. Attacks continue to improve, and since a custom board is not required, in principle, to mount a DPA attack, one would expect that future side-channel attacks on FPGAs will target devices in their native environment. Defenses improve as well. As side-channel attacks became better understood, FPGA vendors added countermeasures, though they are not always explicit about precisely what they have done. Microsemi has licensed CRI technology, but has not released which aspects of that technology they have used. Other vendors are silent on the question of precise circuit details to address DPA.

C. Postconfiguration

FPGAs rely on the application as an active participant in protecting the device after configuration, a capability somewhat novel to FPGAs [45]. FPGAs provide security-related features, but leave the policy decision of handing the features to the user of the FPGA to implement in the application.

1) Readback Disable: Traditional FPGA operation allows the unencrypted bitstream and data to be read out using the bitstream readback command. Therefore, when an FPGA loads an encrypted bitstream, it disables the

readback mechanism, regardless of bitstream settings. This automatic, mandatory setting prevents the simple attack of using the FPGA to decrypt the bitstream, then reading it out. Readback continues to be a valuable feature for both the FPGA manufacturer and the application developer. Proper measures must be taken so that it does not jeopardize the application security.

Skorobogatov and Woods [38] used a side-channel attack to extract a key that unlocked readback in an FPGA that was advertised to have no such capability. While sensationalized as a back door, and questioned for who inserted it, and for whom, in all practicality it was no more than an interface used for device test.

2) Restricted Access to Base Silicon Cryptographic Logic: On-chip cryptographic functions, such as the decryptor, are well-tested, high-speed logic designs of a standard function. It would seem efficient to allow an operating FPGA application to use cryptographic functions after configuration. However, user access to the decryptor, or other cryptographic functions, permits data flow paths that complicate the analysis of the security of the base silicon. If the user has access to the cryptographic functions, and the device is programmed to permit unencrypted bitstreams, then the adversary has access to the cryptographic functions as well. The manufacturer must perform a security analysis to verify that no key data could leak into the application domain.

Second, there are U.S. export and various national import laws worldwide that add risk to the manufacturer if cryptographic functions are used for more than just the configuration of the device. Third, most cryptographic functions, such as AES decyrptors are simply not very large and can be implemented in the user application without consuming much of the FPGA logic. Finally, users have a wide range of needs for cryptographic services. This becomes a cost/benefit tradeoff for the manufacturer. Xilinx and Altera do not allow access to the cryptographic functions on the FPGAs. Microsemi allows access to the cryptographic functions on selected models of the SmartFusion2 devices [28].

3) Restricted Access to Base Silicon Features: Concern over tampered bitstreams in early Virtex devices led Xilinx to prohibit reconfiguration of encrypted bitstreams. This restriction applied to the internal configuration access port (ICAP) as well as the external configuration port. The concern was that a bitstream might be tampered to enable access to ICAP, which could then be used to read back the decrypted configuration. Virtex-II through Virtex-5 devices required encrypted bitstreams to pass a CRC to begin operating, thus ensuring the integrity of the bitstream data. However, as described earlier, CRC does not give a strong defense against bitstream tampering.

Since the addition of authentication in Virtex-6 and 7-series, a secured bitstream must pass the authentication check, defeating any bitstream tampering. Since an authenticated bitstream could not have been modified by an adversary, it can be trusted. This trust applies to the application in general, but specifically enables trusted self-reconfiguration with the ICAP. Since the application design is trusted, ICAP operation is permitted with authenticated encrypted bitstreams. An authenticated bitstream may use the ICAP to launch a partial configuration while the device continues to operate, allowing the design of a trusted reconfigurable platform [53].

ICAP is a Xilinx-specific example of a base silicon feature that, if used maliciously, could provide a vulnerability without the appropriate protections. In all cases, the manufacturer must provide safeguards, while the application developer has final responsibility. It is, of course, possible to construct an insecure application despite the encryption and authentication. For example, if an application developer connected the ICAP interface directly to the external pins, an adversary could interrogate the ICAP to read back the unencrypted application bitstream. FPGA security enables the construction of secure applications; it does not guarantee them.

4) The Value of ICAP and Checking Designs in the Field: ICAP permits logic inside the FPGA to read and write its own bitstream, providing a wide range of powerful use cases. These include:

- internal readback of the device configuration for in-system integrity checks;
- configuration clearing and zeroization;
- algorithm agility for those applications that need to change algorithms without a complete reconfiguration of the device;
- self-test;
- use of user-specific decryption and authentication algorithms with custom protections against attacks such as DPA or other side-channel attacks;
- configuration repair: random single-event upsets (SEU) [23], [52] or intentional tampering may cause configuration bits inside the FPGA to change. Jones [19] describes the SEU controller, an application in which the FPGA logic reads its own bitstream internally through ICAP, checks the stored bitstream with previously computed ECC data, and corrects configuration errors. The SEU controller is intended to detect and correct errors in a high-reliability environment, but it can be used to detect tampering with the FPGA in the field if individual bits are flipped. More recent FPGAs include the SEU detection and scrubbing feature in dedicated hardware [35].

D. Invasive Attacks

Because of the environment of the fielded FPGA, the difficulty of protecting FPGA keys and configuration data persists regardless of the technology used to store them.

When an adversary can physically open the device and scan the contents, no storage technology is wholly secure. However, using our model of cost-based security, some storage methods are more expensive to break, sometimes despite having no qualitative advantage.

The strongest way to prevent theft and tampering with a bitstream is to keep it out of an adversary's hands. The Xilinx Spartan 3AN is a multichip package containing an FPGA die and a flash memory die. Since nothing is transmitted from an external source, the trivial bitstream interception method does not work. However, after decapping the package, the signals between die can be probed to pirate the bitstream.

MicroSemi's SmartFusion devices have internal nonvolatile Flash memory storage as well. These devices are still subject to physical, invasive attack, though that attack is more difficult for several reasons. The storage in these devices is distributed around the device, and there is no localized point at which one could intercept the configuration data, so the attack must scan the entire device. Programmed Flash and antifuse cells are not observably changed from unprogrammed cells, so the detection of programming is more difficult. It may require SEM or thermal analysis. SEM images of programmed and unprogrammed antifuse cells show no apparent differences [2].

Invasive physical attacks on antifuse devices and Flash devices are qualitatively no more difficult than methods for extracting eFuse bits. However, these attacks are considered significantly more expensive because millions of bits of data must be extracted, rather than merely a 256-b key. Further, the resulting extracted programming is not formatted for programming another FPGA, so it must be formatted properly by the adversary in order to clone the design. The proper format is not published, so there is no cryptographically strong protection, but it is considered difficult and tedious.

Despite the concerns, there has not been a report of a successful invasive attack on any FPGA regardless of the internal storage: SRAM, BBRAM, eFuse, Antifuse, or Flash.

E. Environmental Attacks

The circuits inside FPGAs that implement the security functions are no less susceptible to attack than those in other semiconductor integrated circuits. Published attacks on security functions in other devices include out-of-range temperature and power adjustment, overclocking, and other environmental attacks. Defense against these attacks is very difficult because, by definition, semiconductor foundries do not guarantee operation outside their guaranteed environmental range. FIPS140-2, level 4, requires environmental failure protection on cryptographic modules [10] and FPGA vendors provide limited protection from environmental attacks.

The traditional response to environmental attacks has been more robust circuitry, including dedicated voltage regulation for security functions, large hamming distances in security-critical state machines, and redundant storage of critical state values, such as those disabling readback in a secure system.

Xilinx provides an embedded analog-to-digital converter (ADC) that can be used to monitor voltage and temperature both outside and inside the FPGA. Users can configure the circuitry to specific voltage and temperature ranges based on the environment the system will operate in. If the voltage or temperature exceeds this user-specified range, an internal alarm signal will be generated notifying the application running on the device. User-specific actions can then be taken, for example, clearing sensitive cryptographic variables in registers or RAMs, zeroizing the key or clearing the configuration of the device itself and shutting down.

With few exceptions, FPGA manufacturers do not publish details of their security circuitry. Microsemi FuseLock and FlashLock include internal fuses or flash cells that prevent inappropriate access. According to Microsemi, "special security keys are hidden throughout the fabric of the device, preventing internal probing and overwriting. They are located such that they cannot be accessed or bypassed without destroying the rest of the device" [28]. Xilinx readback disabling circuitry has "hardened triple-redundant logic" and key loading FSMs have "large hamming distances between states" [35].

1) Device Identifier: A unique identifier is a powerful way to restrict access to an FPGA, defeating cloning and spoofing. An application can be coded to operate only on the one device that matches a specific identifier or on a subset of devices with a range of values.

Modern FPGAs contain a device identification register. Xilinx provides device DNA, a 57-b serial number programmed in eFuses during manufacture and used for tracking devices. Device DNA is accessible from outside the FPGA via JTAG. In addition, devices include a user-programmable 32-b eFuse field that can be used as an identifier as well. This user eFuse field is only available to logic within the FPGA.

F. PUFs and FPGAs

Other alternatives exist for device identifier. PUFs [14], [39] provide a device-specific unique identifier derived from random process variations. A PUF generator produces a different signature for each manufactured device. PUFs have been demonstrated in FPGA fabric ("soft PUF") as well as in dedicated logic ("hard PUF"). Microsemi's SmartFusion2 includes a hard PUF. Other FPGA vendors have IP providers who provide soft PUF functions in fabric. Therefore, application developers can build PUFs for device identification today with existing FPGAs.

There are several drawbacks for the use of PUFs that have precluded their use as decryption keys in FPGAs. First, the PUF only resides inside the device. It must be read out of the device to encrypt the bitstream data file.

Alternatively, Kean recommended an encryption method in which the FPGA encrypts its own data file using its internal key and emits the encrypted data for external storage [20]. More importantly, the PUF is unique to each unit, so the bitstream must be encrypted uniquely for each device. This problem may be addressed by including a key transformation word, the exclusive-OR of the computed PUF for the device with the actual key used to decrypt the data. Still, at system build time, the FPGA must be powered on and the transformation word derived. Perhaps most importantly, PUFs are not stable: a few bits may change over the lifetime of the device. This is not particularly important for a device identifier, but disastrous for a decryption key. One method to compensate for this is the addition of helper data to the PUF-encrypted bitstream. Helper data are fundamentally error correcting code information for correcting errant bits in the PUF. It is unclear how much information about the key is leaked in helper data. Finally, long-term PUF reliability data over process, voltage, and temperature is sketchy at advanced process nodes, leading to concern over lost keys during the lifetime of the fielded device.

VII. APPLICATIONS

A. IFF Flow for Nonsecured Devices

Baetoniu [6] described "identification friend or foe" (IFF), a way to tie an FPGA bitstream to a specific system. IFF uses an external storage device, a secure serial electrically erasable programmable read-only memory (EEPROM), such as the Dallas Semiconductor/Maxim DS2432 (Fig. 6). The secure EEPROM includes a cryptographic hash function. At system build time, the application developer programs a secret key into the EEPROM and also programs the secret key into the FPGA application.

After the FPGA boots, it uses its random number generator to interrogate the EEPROM. The EEPROM computes the hash of the random string with its stored key. The FPGA does the same. If the two hashes match, the FPGA continues to operate. If the hashes do not match, the FPGA enacts countermeasures such as ceasing operation or disabling premium functionality. The check may be repeated as often as desired during operation.

IFF ties the FPGA bitstream to a properly programmed secure EEPROM. Although it can be applied to an FPGA without bitstream encryption, doing so leaves the system vulnerable. An adversary may reverse engineer the bitstream and disable the check on the hash function. This mechanism is even vulnerable with an encrypted, but not authenticated, bitstream, because an adversary may attempt to disable the hash function check by a bit-flipping attack or random perturbation of the plaintext to disable the hash check.

B. Metered IP

As third-party IP cores become more common, one would like a mechanism to charge per copy for those cores. The core vendor would be paid for each use, just as if it had been a physical device. Guajardo *et al.* [15] described a method for doing this and the company Intrinsic-ID developed into a product under the brand name Quiddicard [17].

The method has an enrollment phase and an authentication phase. In the enrollment phase, the FPGA is programmed with a PUF which generates an identifier unique to the FPGA. An activation code is generated from the PUF value and stored off-chip. The activation code generation is a proprietary algorithm, but may be an encryption of the PUF value using a private key of a public/private key pair. In the authentication phase, the same PUF is constructed in the FPGA and the design is authorized with the activation code (Fig. 7).

To turn this activation process into an IP metering mechanism, the generation of the activation code may be done by a trusted third party, possibly a trusted piece of billing hardware at a manufacturing site that reports the IP usage as it generates the activation code. This mechanism has been extended to include multiple keys to permit access to multiple pieces of IP in the FPGA application [18].

This mechanism relies on confidentiality and authentication of the application design, so that an adversary

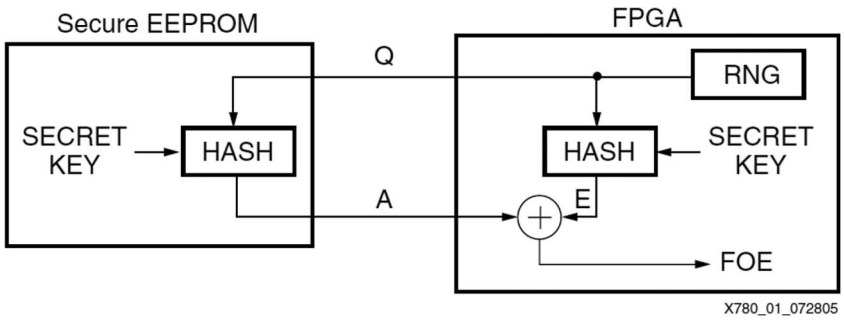

Fig. 6. *IFF design.*

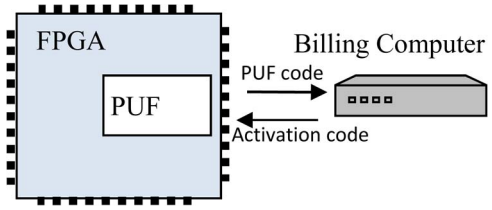

Fig. 7. *Metered IP system architecture.*

cannot reverse engineer the device to remove the activation code checking. There is nothing fundamental about using a PUF for identification. Device DNA or some other unique or nearly unique fixed device identifier can serve.

C. Just in Time Secure Configuration

Utilizing partial reconfiguration and authenticated encrypted bitstreams, it is possible to design a system where critical technology (CT) is only configured into the device when it is needed, thereby adding an additional layer of security to the system. Peterson [35] proposed a method by which a user application is partitioned between CT and non-CT. The non-CT is resident in the FPGA at all times and the CT logic is partially reconfigured into the FPGA only when needed. Otherwise, it is stored externally, encrypted and authenticated.

The CT, which exists as a partial configuration, can be decrypted by the device using the device key or by the application using a user-specified algorithm implemented in the FPGA fabric, and potentially a PUF to generate the key. The boot configuration of the FPGA sends the CT partial bitstreams to the ICAP so that the decryption process is completely contained with the FPGA. Encryption is required to ensure the privacy of keys included in the CT partial bitstreams or the boot configuration bitstream. Authentication is required so that the bitstreams cannot be tampered in a way that compromises the CT partial bitstreams. The IP described by Zeineddini and Wesselkamper [53] for secure and high-reliability applications utilizing partial reconfiguration also checks for tampering. It uses the integrated ADC to monitor power and temperature, and checks the JTAG port to detect tamper conditions. If necessary, the IP zeroizes the CT and its key.

D. Fault-Tolerant Design

FPGA manufacturers supporting confidentiality, integrity, and authentication of the configuration provide a strong foundation that users can build high-reliability system upon. Cryptographic processing and security services, like any high-reliability function, must be fault tolerant. Xilinx's isolation design flow (IDF) [7], developed in conjunction with government entities [24], was the first in the programmable logic industry. Altera has since developed similar technology, called the design separation flow [5].

IDF provides fault containment at the FPGA module level, enabling single-chip fault tolerance by various techniques, including modular redundancy, watchdog alarms, segregation by safety level, and isolation of test logic for safe removal [7]. The applicability of this type of technology goes beyond cryptographic processing and security. The same technology can be used to aid in compliance for systems that must be designed to safety-critical standards such as IEC61508, ISO26262, and DO-254.

The basic concept is to separate critical and/or intentionally redundant functions physically on the FPGA. This can be accomplished through careful floorplanning and the use of unused logic as fences. Fig. 8 represents a design that has been floorplanned with IDF. Fig. 9 is the same design after place and route.

The fences are exhaustively analyzed by the FPGA manufacturer to show that a single failure would not compromise the isolation or redundancy built into the system. The goal is to minimize the size of the fence to reduce the inefficiencies that come with its use [16]. As an example, the width or height of a fence made of configurable logic blocks (CLBs) in a Xilinx 7-series FPGA is a single CLB.

In an ideal world, each module would be completely isolated from each other. In practice, this scenario is not feasible: some level of communication must exist between isolated regions. Xilinx developed the concept of "trusted routing," restricted routing that is specifically chosen by the place and route algorithms such that the isolation established by the use of "fences" is not compromised.

Finally, no high-reliability system is complete without the use of independent verification. To address concerns associated with software "bugs" or inappropriate use of the design methodology by the user, FPGA manufacturers must provide independent verification tools that can be applied to the design to validate the isolation of the modules. Xilinx

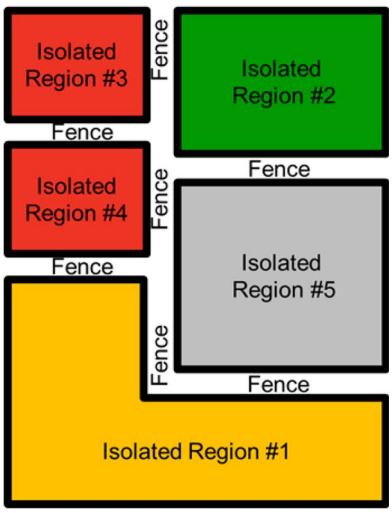

Fig. 8. *Notional floorplan of a design into five isolated regions.*

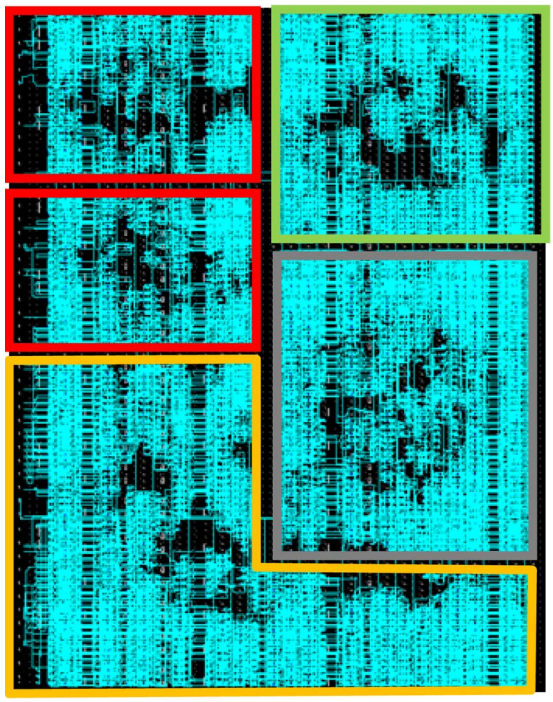

Fig. 9. *FPGA editor view of a design implemented using the IDF methodology.*

developed the isolation verification tool (IVT) for this purpose. IVT can be used early in the development flow to aid in isolation verification before a printed wiring board (PWB) is committed. It is also used once the design is complete in order to verify that the final design, placed and routed, has the isolation designed in that the user intended.

E. Single-Chip Cryptography

Single-chip crypto (SCC) combines data of different levels of secrecy or control in a single device. The device must not only protect programs during loading, but also it must defend against attacks from outside and attacks while operating, including leakage of protected information across internal boundaries. Therefore, single-chip cryptography aggregates much of the technology discussed in this paper.

SCC uses the authenticated encryption capability to load a boot loader. The boot loader, isolation region #1 in Fig. 8, manages further FPGA configuration, software for on-chip processors, and data handling. Because it was authenticated and encrypted, the boot loader is known to be unaltered by potential adversaries or accidental bit errors. In addition, sensitive data, such as session keys, are known to be kept secret. To ensure no internal leakage of information, SCC implements the fences of IDF as described in Section VII-D (Fig. 8) to separate sensitive data spatially in the FPGA. This separation assures the confidentially of sensitive information even in the presence of accidental or intentional attacks on the fences. The spectrum of isolation capabilities is sufficient to support applications such as the separation of red and black data processing, key management, and other high-reliability functions.

Bitstream scrubbing, using internal readback, continually monitors the configuration data, in particular the isolation fences, to ensure that changes to the configuration are detected and corrected quickly. SCC can even verify that the device DNA is correct, ensuring operation on the proper individual chip.

Starting with the root of trust, followed by the power and flexibility of both hardware and software, coupled with the application of isolation technologies and partial reconfiguration, a system that would typically have been developed through the use of multiple devices now could be integrated into just one with no loss of security.

VIII. THE FUTURE OF FPGA SECURITY

A. Field-Programmable SoC

SCC was originally conceptualized and developed in cooperation with government authorities for FPGAs [24], and the application provides additional value in new programmable SoCs such as Zynq. Zynq includes both a programmable logic subsystem (PL) that comprises hundreds of thousands of gates of logic, and a processor subsystem (PS) that includes a dual-core ARM (ARM Holdings, Cambridge, U.K.) Cortex A9 processor, caches, memories, and peripherals, connected to one another and to the PL using an Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI) bus. The Zynq device boots securely, using authenticated encryption capabilities like those described for FPGAs. Zynq also provides asymmetric and symmetric authentication, confidentiality, and integrity. Leveraging this root of trust, applications can implement cryptoprocessors or systems performing cryptographic functions in the combination of processor and FPGA with confidence that they have not been compromised.

In Zynq, the processor subsystem is known to be isolated physically from the programmable logic. Within the PL, isolated regions as in IDF ensure separation of sensitive data spatially. Within the PS, known software methods, such as hypervisors and ARM Trustzone technology isolate sensitive software processes from other processes. The trusted boot loader decrypts and authenticates all configuration data and software.

Partial reconfiguration is further enhanced. The entire PL can be reconfigured, or even powered down, controlled by the PS. Alternatively, portions of the PL can be partially reconfigured for applications that require algorithm agility. The same reliability checks performed on ICAP [52] can be applied to the processor configuration access port (PCAP) to ensure proper data integrity of software. Decryption and authentication of partial configuration files

can be performed by either the PS or the PL, allowing users the flexibility to choose their own authentication and decryption algorithms as well as perform functions such as authenticate before decryption to aid in defense against side-channel attacks. Of course, key management remains a critical consideration in these applications.

B. Conclusion

Security in FPGAs has been driven by the need to address new threats, by the growth in value of the IP of the applications, and by the growth in the expected sophistication of the adversary. All three drivers continue to operate. New areas of protection, such as confidentiality of the data handled by the FPGA, metering of third-party IP, and counterfeit protection motivate additional capabilities and combinations of capabilities in the FPGA. Modern FPGAs and new programmable SoC devices hold applications that comprise complete systems, processing very sensitive data and controlling valuable systems. The high value of the applications, the data they handle, and the systems they control motivate well-equipped adversaries to steal IP or to subvert the systems of which the FPGA is a part.

As adversaries become more sophisticated, so do the FPGA defenses. Future FPGA security features must continue to improve to meet all three drivers. As in the past, these features will include circuits on the base array, algorithms in silicon, and IP in the programmable part of the device. ■

REFERENCES

[1] Actel, "Implementation of security in Actel's ProASIC and ProASICPLUS Flash-based FPGAs," Appl. Note AC185, 2003.

[2] Actel, "Understanding Actel antifuse device security," 2004. [Online]. Available: www.actel.com/documents/AntifuseSecurityWP.pdf

[3] P. Alfke, "Configuration issues: Power-up, volatility, security, battery back-up," Xilinx, Appl. Note XAPP092, 1997. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp092.pdf

[4] Altera, "Using the design security features in Altera FPGAs," Appl. Note, AN-556, Jun. 19, 2013.

[5] Altera, "Quartus II design separation flow," 2013. [Online]. Available: http://www.altera.com/literature/hb/qts/qts_qii51019.pdf

[6] C. Baetoniu, "FPGA IFF copy protection using Dallas Semiconductor/Maxim DS2432 Secure EEPROMs," Xilinx, Appl. Note XAPP780 v. 1.1, 2010. [Online]. Available: http://www.zylinks.com/support/documentation/application_notes/xapp780.pdf

[7] J. D. Corbett, "The Xilinx isolation design flow for fault-tolerant systems," Xilinx WP412, 2012. [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp412_IDF_for_Fault_Tolerant_Sys.pdf

[8] S. Drimer, "Authentication of FPGA bitstreams, why and how," *Reconfigurable Computing: Architectures, Tools and Applications*, vol. 4419. Berlin, Germany: Springer-Verlag, 2007, pp. 73–84.

[9] S. Drimer, "Security for volatile FPGAs," Ph.D. dissertation, Comput. Sci. Dept., Cambridge Univ., Cambridge, U.K., 2009.

[10] National Institute of Standards and Technology (NIST), "Security requirements for cryptographic modules," FIPS 140-2, 2001.

[11] National Institute of Standards and Technology (NIST), "Announcing the advanced encryption standard," FIPS 197, 2001.

[12] National Institute of Standards and Technology (NIST), "The keyed-hash message authentication code (HMAC)," FIPS PUB 198, Mar. 6, 2002. [Online]. Available: http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf

[13] National Institute of Standards and Technology (NIST), "Secure hash standard," FIPS PUB 180-2 + Change Notice to include SHA-224, Aug. 1, 2002. [Online]. Available: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

[14] J. Guajardo, S. S. Kumar, G. J. Schrijen, and P. Tuyls, "Physical unclonable functions and public-key crypto for FPGA IP protection," *Proc. IEEE Int. Conf. Field-Programm. Logic Appl.*, 2007, pp. 189–195.

[15] J. Guajardo, S. S. Kumar, G. J. Schrijen, and P. Tuyls, "Brand and IP protection with physical unclonable functions," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2008, pp. 3186–3189.

[16] T. Huffmire *et al.*, "Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems," in *Proc. IEEE Symp. Security Privacy*, 2007, pp. 281–295.

[17] Intrinsic-ID, "Quiddikey-Flex," 2013. [Online]. Available: http://www.intrinsic-id.com/products/quiddikey-flex

[18] Intrinsic-ID, "Quiddicard protecting your IP gainst overproduction, counterfeiting and cloning," Aug. 30, 2013. [Online]. Available: www.intrinsic-id.com/products/quiddicard-

[19] L. Jones, "Single event upset (SEU) detection and correction using Virtex-4 devices," Xilinx, Appl. Note #714, 2007. [Online]. Available: http://www.xilinx.com/bvdocs/appnotes/xapp714.pdf

[20] T. Kean, "Secure configuration of field programmable gate arrays," in *Proc. IEEE Annu. Symp. Field-Programm. Custom Comput. Mach.*, 2001, pp. 259–260.

[21] Lattice, "FPGA design security issues: Using Lattice FPGAs to achieve high design security," White Paper, 2007.

[22] Lattice, "Advanced security encryption key programming guide for LatticeECP3, LatticeECP2MS, LatticeECP2S devices," Tech. Note TN1215, 2012.

[23] A. Lesea, S. Drimer, J. Fabula, C. Carmichael, and P. Alfke, "The Rosetta experiment: Atmospheric soft error rate testing in differing technology FPGAs," *IEEE Trans. Device Mater. Reliab.*, vol. 5, no. 3, pp. 317–328, Sep. 2005.

[24] M. McLean and J. Moore, "FPGA-based single chip cryptographic solution," *Military Embedded Systems*, 2007. [Online]. Available: http://www.mil-embedded.com/pdfs/NSA.Mar07.pdf

[25] Microsemi, "Igloo2 FPGAs revision 0," 2013. [Online]. Available: www.microsemi.com/document-portal/doc_download/132042-igloo2-fpga-datasheet

[26] Microsemi, "Axcelerator family FPGAs," 2012. [Online]. Available: http://www.microsemi.com/document-portal/doc_download/130669-axcelerator-family-fpgas-datasheet

[27] Microsemi, "Implementation of security in Microsemi Antifuse FPGAs," Appl. Note AC168, 2012.

[28] Microsemi, "Security architecture," 2013. [Online]. Available: http://www.microsemi.com/products/fpga-soc/technology-solutions/security/security-architecture

[29] Microsemi, "SmartFusion2 SoC FPGA reliability and security user's guide," 2013.

[30] A. Moradi, A. Barenghi, T. Kasper, and C. Paar, "On the vulnerability of FPGA bitstream encryption against power analysis attacks: Extracting keys from Xilinx Virtex-II FPGAs," in *Proc. ACM Conf. Comput. Commun. Security*, 2011, pp. 111–124.

[31] A. Moradi, M. Kasper, and C. Parr, "Black-box side-channel attacks highlight the importance of countermeasures—An analysis of the Xilinx Virtex 4 and Virtex-5 bitstream encryption mechanism," in *Proc. 12th Conf. Topics Cryptol.*, 2012, DOI: 10.1007/978-3-642-27954-6_1.

[32] A. Moradi, D. Oswald, C. Paar, and P. Swierczynski, "Side channel attacks on the bitstream encryption mechanism of Altera Stratix II," in *Proc. ACM/SIGDA Int. Symp. Field-Programm. Gate Arrays*, 2013, pp. 91–100.

[33] National Institute of Standards and Technology (NIST), "Recommendation for block cipher modes of operation," Special Publ. 800-38A, 2001.

[34] M. Parlekar, "Authenticated encryption in hardware," M.S. thesis, Electr. Comput. Eng. Dept., George Mason Univ., Fairfax, VA, USA, 2005.

[35] E. Peterson, "Developing tamper resistant designs with Xilinx Virtex-6 and 7 series FPGAs," Xilinx, Appl. Note XAPP1084, 2012.

[36] L. Sanders, "Secure boot of Zynq-7000 all-programmable SoC," Xilinx, Appl. Note XAPP 1175 (v1.0), 2013.

[37] B. Schneier, *Applied Cryptography Second Edition*. New York, NY, USA: Wiley, 1996.

[38] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," *Cryptographic Hardware and Embedded Systems—CHES 2012*, vol. 7428. Berlin, Germany: Springer-Verlag, 2012, pp. 23–40.

[39] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proc. Design Autom. Conf.*, 2007, pp. 9–14.

[40] A. Telikepalli, "Is your design secure?" Xilinx, 2003. [Online]. Available: http://www.xilinx.com/publications/archives/xcell/Xcell47.pdf

[41] S. Trimberger, "Method and apparatus for protecting proprietary configuration data for programmable logic devices," U.S. Patent 6 654 889, 2003.

[42] S. Trimberger, J. Moore, and W. Lu, "Authenticated encryption of FPGA bitstreams," in *Proc. 19th ACM/SIGDA Int. Symp. Field Programm. Gate Arrays*, 2011, pp. 83–86.

[43] S. Trimberger, *Field-Programmable Gate Array Technology*. Norwell, MA, USA: Kluwer, 1994.

[44] S. Trimberger, "Trusted design in FPGAs," in *Proc. Design Autom. Conf.*, 2007, pp. 5–8.

[45] S. Trimberger and J. Moore, "FPGA security: From features to capabilities to trusted systems," in *Proc. 51st Annu. Design Autom. Conf.*, 2014, DOI: 10.1145/2593069.2602555.

[46] S. Trimberger, "Security in SRAM FPGAs," *IEEE Design Test Comput.*, vol. 24, no. 6, p. 581, Nov./Dec. 2007.

[47] S. Trimberger, "Three ages of FPGAs," in *FPGA20. Highlights of the International Symposium on Field-Programmable Gate Arrays, ACM*, 2011, pp. 1–18.

[48] T. Tuan, T. Strader, and S. Trimberger, "Analysis of data remanence in a 90 nm FPGA," in *Proc. IEEE Custom Integr. Circuits Conf.*, 2007, pp. 93–96.

[49] T. Wollinger and C. Parr, "How secure are FPGAs in cryptographic applications," *Field Programmable Logic and Application*, vol. 2778, P. Y. K. Cheung, G. A. Constantinides, and J. T. de Sousa, Eds. Berlin, Germany: Springer-Verlag, 2003, pp. 91–100.

[50] Xilinx, "Virtex-4 FPGA configuration user guide, v1.11," UG071, 2009.

[51] Xilinx, "Virtex-6 FPGA Configuration User Guide," UG360, Jul. 30, 2010. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug360.pdf

[52] Xilinx, "Device reliability report, second quarter 2013," UG116, 2013.

[53] A. Zeineddini and J. Wesselkamper, "PRC/EPRC: Data integrity and security controller for partial reconfiguration," Appl. Note XAPP887, 2012.

ABOUT THE AUTHORS

Stephen M. Trimberger (Fellow, IEEE) received the B.S. degree in engineering and applied science from the California Institute of Technology, Pasadena, CA, USA, in 1977, the M.S. degree in information and computer science from the University of California at Irvine, Irvine, CA, USA, in 1979, and the Ph.D. degree in computer science from the California Institute of Technology in 1983.

He was employed at VLSI Technology from 1982 to 1988. Since 1988 he has been at Xilinx, San Jose, CA, holding a number of positions. He is currently a Xilinx Fellow, heading the Circuits and Architectures group in Xilinx Research Labs in San Jose, CA, USA. He is an author and editor of five books as well as dozens of papers and journal articles. He is an inventor on more than 200 U.S. patents in the areas of integrated circuit (IC) design, field-programmable gate array (FPGA) and application-specific integrated circuit (ASIC) architecture, computer-aided engineering (CAE), 3-D die stacking semiconductors, and cryptography.

Dr. Trimberger is a four-time winner of the Freeman Award, Xilinx's annual award for technical innovation. He is a Fellow of the Association for Computing Machinery (ACM).

Jason J. Moore received the B.S. degree in electrical engineering from New Mexico State University, Las Cruces, NM, USA, in 1992.

He is currently a Director of Market Segments Engineering at Xilinx, Albuquerque, NM, USA, focused on security and safety architectures. Previous to his assignments at Xilinx, he was responsible for the development of field-programmable gate array (FPGA)-based communication security equipment in a wide range of avionics and ground-based platforms at the Motorola Government Group. He has been awarded multiple patents on cryptographic design in addition to novel approaches for logical and functional isolation within a single FPGA.

Mr. Moore is a two-time winner of the Freeman Award, Xilinx's annual award for technical innovation.