MUTARCH: Architectural Diversity for FPGA Device and IP Security

Robert Karam[1], Tamzidul Hoque[1], Sandip Ray[2], Mark Tehranipoor[1], and Swarup Bhunia[1]

[1] University of Florida, Gainesville, FL 32608
[2] NXP Semiconductor, Austin, TX 78735

Abstract— Field Programmable Gate Arrays (FPGAs) are being increasingly deployed in diverse applications including the emerging Internet of Things (IoT), biomedical, and automotive systems. However, security of the FPGA configuration file (i.e. bitstream), especially during in-field reconfiguration, as well as effective safeguards against unauthorized tampering and piracy during operation, are notably lacking. The current practice of bitstreram encryption is only available in high-end FPGAs, incurs unacceptably high overhead for area/energy-constrained devices, and is susceptible to side channel attacks. In this paper, we present a fundamentally different and novel approach to FPGA security that can protect against all major attacks on FPGA, namely, unauthorized in-field reprogramming, piracy of FPGA intellectual property (IP) blocks, and targeted malicious modification of the bitstream. Our approach employs the *security through diversity* principle to FPGA, which is often used in the software domain. We make each device architecturally different from the others using both physical (static) and logical (time-varying) configuration keys, ensuring that attackers cannot use *a priori* knowledge about one device to mount an attack on another. It therefore mitigates the economic motivation for attackers to reverse engineering the bitstream and IP. The approach is compatible with modern remote upgrade techniques, and requires only small modifications to existing FPGA tool flows, making it an attractive addition to the FPGA security suite. Our experimental results show that the proposed approach achieves provably high security against tampering and piracy with worst-case 14% latency overhead and 13% area overhead.

I. Introduction

Recent years have seen a rapid proliferation in the use of Field Programmable Gate Arrays (FPGAs) in diverse domains, including automotive, defense, networking, health care, and consumer electronics. For many devices, a key requirement is the need for in-field hardware reconfigurability to adapt to changing requirements in functionality, energy-efficiency, and security. FPGAs have emerged as a popular electronic component for addressing this reconfigurability demand [1], as they provide high flexibility compared to custom ASICs, while entailing significantly higher energy-efficiency and performance than designs based on firmware/software running in processors. FPGAs often provide significant benefits in real-time performance, making them attractive as hardware accelerators. Furthermore, FPGA-based designs are known to be more secure than both ASIC and processor against supply-chain attacks, since design details are not exposed to untrusted foundries or design houses.

However, the FPGA configuration file, also called the *bitstream*, is susceptible to a variety of attacks, which can potentially lead to unauthorized reprogramming, reverse-engineering, and/or piracy of the intellectual property (IP). Modern high-end FPGA devices often include on-board decryption hardware, allowing for some measure of security; however, encrypted bitstreams are generally transmitted along with the decryption key, which creates a significant vulnerability. Furthermore, even dedicated decryption hardware can incur significant hardware overhead for area and energy-constrained systems, e.g. Internet-of-Things (IoT) edge devices. Mathematically, encryption algorithms are known to be highly secure against brute-force attacks. However, in many cases, attackers can have physical access, and most on-board encryption techniques are susceptible to side-channel attacks, *e.g.* by key extraction through power profile signatures [1]–[3].

Unless additional countermeasures are in place, an adversary can convert the bitstream to a netlist [4], enabling targeted malicious modifications (*e.g.* Trojan insertion) as well as IP piracy. The conversion step may not be necessary for Trojan insertion; techniques such as Unused Resource Utilization [5], which inserts Trojans in empty spaces in the configuration file, and Mapping Rule Extraction [6], a type of known design attack, can be mounted on a bitstream. Alternatively, if the hardware itself is cloned [7], a pirated bitstream could be used with a counterfeit hardware.

Such attacks are made possible by the fact that all FPGAs of a given family have physically identical architectures. In other words, the decrypted bitstream taken from one specific product can just as easily be mapped as a blackbox to another (identical) product. Similarly, a *maliciously modified* bitstream that is successfully deployed on one system can be mapped to another. This is analogous to a computer virus (the maliciously-modified bitstream) which infects a particular version of an operating system (the FPGA), and can propagate to other computers with the same OS (FPGA) because program execution (the architecture) is identical. For products which are intended to remain in the field for long time (10-30 years), such as automotive systems, an attacker could feasibly modify the configuration of a safety-critical FPGA, and deploy this to other vehicles of the same year, make, and model. Therefore, though encryption offers a security layer, when used alone, it is not sufficient to protect devices with long in-field lifetime,

978-1-5090-1558-0/17/$31.00 ©2017 IEEE

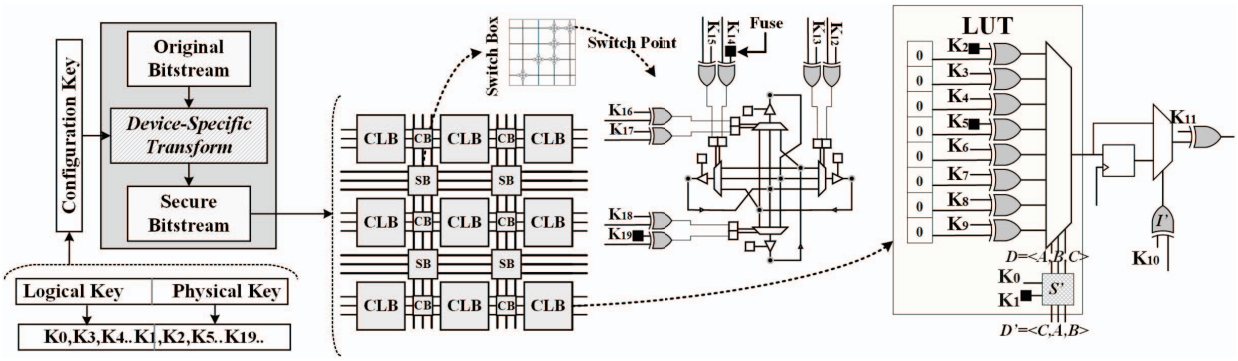

Fig. 1. Overview of the time-varying FPGA Architecture, from left to right: a two part (logical and physical) key is used to perform a device-specific obfuscation transform. The secure bitstream is mapped to appropriate FPGA. Resources are augmented with logic which implements the inverse transform. This ensures that bitstreams mapped to unauthorized devices will be nonfunctional. Because the logical key is time-varying, the architecture is *mutable* and thus prevents known design attacks.

TABLE I
PROPERTIES AND QUALITATIVE COMPARISON OF PHYSICAL AND LOGICAL KEYS

Key Type	Time Var.	Storage	Area Ovhd.	In-field Upgrades	Known Design	Destructive RE
Physical (*P*)	No	Fuses	Low	Not Secure	Weak	Strong
Logical (*L*)	Yes	Runtime	Mod	Secure	Strong	Weak
Combined	Yes	Mixed	High	Secure	Strong	Strong

especially those with network connectivity.

In this paper, we propose *MUTARCH*, a novel architecture with associated CAD technology for FPGA security which provides provably robust protection against in-field bitstream reprogramming and IP piracy. Figure 1 illustrates the overall approach. MUTARCH permits wireless reconfiguration that does not *rely* on encryption, though it can still be used in conjunction as an extra layer of defense when available. The proposed approach presents a fundamental departure from existing protection approaches that rely on cryptographic techniques. It is rooted in the idea of "security through diversity", where each FPGA device will have a unique architecture, despite being manufactured with existing processes and techniques, that can additionally *mutate* over time. This supplies robust protection against brute force attack, as well as security against known design attacks through a moving target defense. This also results in a unique bitstream-to-device mapping which has several major benefits: (1) device identification is an intrinsic requirement, which ensures that only valid devices receive upgrades; (2) upgrades sent to unauthorized or counterfeit devices will not function because the bitstream cannot be mapped correctly, mitigating an attacker's economic motivation for device cloning, and preventing reverse engineering of valuable IP blocks; and (3) maliciously modified bitstreams cannot be mapped to another device, so that breaking one device does not put others at risk. It is distinct from existing logic encryption or hardware metering techniques [8] because: (1) it applies to FPGA bitstreams rather than an ASIC design; (2) it allows changing the architecture over time; (3) it is integrated into the FPGA's reconfigurable fabric and the application mapping tool flow; and (4) it does not require any expensive on-chip resource, *e.g.* support for public key cryptography.

TABLE II
KEY ALLOCATION FOR VARIOUS FPGA RESOURCES

Arch. Level Configuration	Physical Config.	Logical Config.
Output Inversion	No	Yes
LUT Input Reordering	Yes	Yes
LUT Content Inversion	No	Yes
Switch Box Config. Bits	Yes	Yes
Mux Selection Bits	No	Yes

In particular, the paper makes the following major contributions: (1) it investigates the concept of mutable FPGA architectural fabric for the purposes of device and IP security, including efficient hardware modifications to enable unique and time-variant mappings, as well as the communication protocol for remote in-field upgrades; (2) it presents a detailed security analysis for the proposed approach, considering all possible attack models; and (3) it demonstrates the viability of this approach using a complete CAD framework that we have developed based on a widely-used open-source FPGA mapping tool called VTR [9] and evaluates bitstream security as well as overhead for a set of common benchmark circuits.

II. FPGA HARDWARE SECURITY

In this section, we describe the proposed mutable FPGA architecture in details, including the static and time-variant architecture configuration layers. Next we present the secure

FPGA mapping tool, which guarantees functional correctness when mapping a design.

A. Mutable FPGA Architecture

The security of the MUTARCH architecture arises from two separate "layers" – one physical, and the other logical – with one configuration key for each layer. Qualitatively, these keys differ in terms of time variance, storage, overhead, and resilience to various attacks, as shown in Table I. Together, these keys can also be considered as the FPGAs unique *architectural configuration* (Fig. 1), because they are used as input to the secure bitstream transform process. Hence, prior to upgrading, the device must be identified to ensure that the correct bitstream is sent to a system. The actual bitstream transformation can therefore occur towards the back-end of a vendor's tool flow (*e.g.* after place & route, but before bitstream generation), reducing overall compilation time. Because of the unique bitstream-to-device association, device authentication is necessary for the process. The design flow for MUTARCH is illustrated in Fig. 2.

A typical mapping function $F\left(S\left(K,B\right),I\left(K,B\right)\right)$ operates on disjoint subsets of the key K and bitstream B to modify the bitstream in such a way that, when mapped to the target FPGA, the internal logic implements an appropriate inverse transform, resulting in a functionally correct mapping. This is appropriate for modern FPGAs, in which bitstreams are generated such that they implement the desired functionality for a given device architecture. However, rather than a bitstream functioning on all devices of a given family, it will only work on one specific device. Therefore, it creates a unique bitstream-to-device association to prevent piracy of the IP mapped to the FPGA.

Given the highly flexible nature of FPGAs, there are many internal components which can be subject to physical configuration that, when changed from device to device, would represent a unique device architecture. A subset of these components is listed in Table II. It gives a designer the flexibility to balance overhead with the required level of security by implementing changes in as few or as many separate structures as needed. Additional related details are given in the case study (Section IV. Note that the physical key should not be allocated to the output inversion or the multiplexer select lines, which will induce static change to a bitstream and therefore, can be exploited by an attacker to gain knowledge of the architecture. Conversely, the logical key can be applied to any of the listed resources because it changes each reprogram cycle.

1) Physical Layer: The first of two security layers is based on physical architectural modifications to the underlying FPGA fabric. This layer is comprised of a network of fuses judiciously placed on different configurable components and programmed after fabrication using techniques commonly applied for defect and fault tolerance. This programming should be performed by the manufacturer and not at the fabrication facility, rendering it less susceptible to supply chain attacks. In addition, because each FPGA must eventually be programmed with its vendor's specific toolset, the physical modification prevents the fabrication facility from overproduction or cloning attacks at foundry.

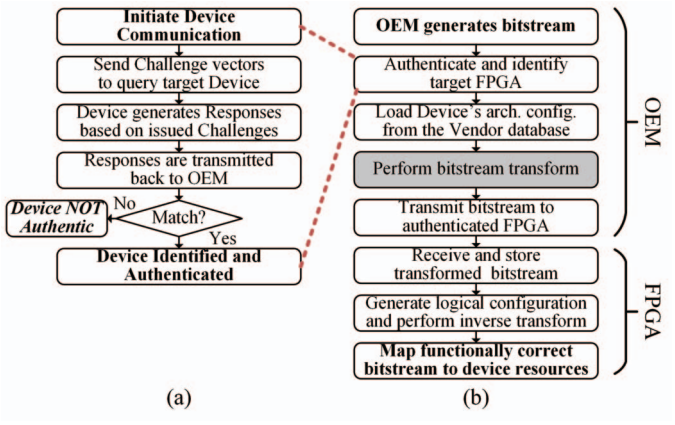

Fig. 2. Overall design flow for MUTARCH: (a) device identification enables the system designer to obtain device configuration keys; (b) the unique bitstream for a specific device is used in the upgrade process.

As a concrete example of the physical security layer, consider the inputs to a given lookup table (LUT). Inputs can come from other LUTs in the design, and both the value of the input, as well as their order, is crucial to proper functionality. By inserting a switch network whose inputs can be programmed by a fuse, the order of inputs to a given LUT can be permanently modified (Fig. 1, right). This must be factored in during bitstream generation, so that proper functionality is preserved.

2) Logical Layer: The second security layer is based on in-field architectural modifications rather than physical changes. This enables the underlying FPGA fabric to effectively *change with time*, making it a *mutable* during its life-time. This property of time-variance is essential to security against known design attacks. This layer is realized through the run-time configuration of permutation and inversion networks which modify the functions mapped to the FPGA (e.g. the lookup table contents) and how the LUTs are connected together. This layer takes as input a subset of the bitstream, as well as a key that is generated *internally* using, for example, a cryptographically secure pseudorandom number generator (CSPRNG) such as PUFKY [10]. PUFKY is attractive because it leverages Physical Unclonable Functions (PUFs) for challenge/response-based seeding of the CSPRNG, which fits well with the desired properties for a remotely reconfigurable and upgradeable system (i.e. enables device identification through unique signatures).

Just like in the physical layer, the logical network requires that the bitstream be modified during the vendor tool flow, applying the transform to various structures, such as the LUT content, connection and switch boxes, and other supported FPGA resources. Because known design attacks require a large number of mappings of designs to a device, changing the logical key during each reprogramming cycle presents a robust moving target defense.

B. Secure FPGA Mapper

To evaluate MUTARCH comprehensively, we have implemented a complete secure FPGA mapping tool based on

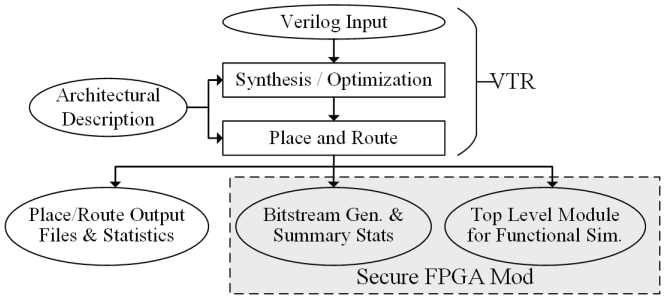

Fig. 3. Secure FPGA mapping – modifications to the VTR mapping flow, denoted by the shaded box.

VTR [9], a popular academic tool for FPGA architecture research. An architecture description file implementing a variable number of 4-input LUTs is used for mapping a series of benchmarks. The VTR tool takes as input a file in Verilog HDL, parses and decomposes the circuit into appropriately sized LUTs (as defined by the architecture description file), and then with the Versatile Place and Route (VPR) tool [9], performs packing, placement, and routing.

We modified VPR as shown in Fig. 3 to perform the function described in the Algorithm 1. This function takes as input a design, a pre-determined physical key, and the seed for the logical key. The number of inputs (num_inputs) and the original truth table (tt) are calculated. Next, a number of bits equal to the LUT size are obtained from the CSPRNG, after which the tt undergoes addition with the key modulo 2, followed by a bitwise permutation as defined by first the physical key, then the logical key. Next, the current LUTs output inversion status (OI_Status) is set based on the logical key, and the status of the output inversion for all fan in nodes are checked. This affects the permutation of the current LUTs content, and the current tt must undergo one last output inversion transform (oi_xform) defined by the fan in output inversion status ($FIOIS$).

This function results in two bitstreams defined by their LUT content bits. It allows us to determine the level of security, as determined by the Hamming Distance between the original and secure bitstream, and the difference in pairwise intra-bitstream distance. The mapping tool additionally modifies the existing Verilog writing functionality in VTR to enable functional simulation. The existing LUT primitives, defined in Verilog, are modified to support the bitstream transforms through the addition of LUT input switch networks, and XOR gates on the LUT content bits and output bit. The top level module generation code, which instantiates the CLBs and interconnects, is modified as well, to support input of the physical and logical key networks. A test bench is also generated by the tool which instantiates both the original and secure FPGA mapping, generates the test patterns from known input patterns for the benchmark circuits, and finally compares the output of both the original and secure mappings.

C. Correctness of Mapped Design

An important aspect of the FPGA synthesis process is to ensure that the mapped design is functionally correct. In the proposed scenario, correctness is guaranteed by construction.

Algorithm 1 Secure Bitstream Transform

Input: Circuit C, Physical Key K_p, Logical Key Seed K_s
Output: Original Bitstream B_o, Secure Bitstream B_s

```
InitCSPRNG(K_s)
for each Blocks B in C do
  for each Primitives P in B do
    if P is type LUT then
      FIOIS ← 0
      numInputs ← getLUTinputs(P)
      tt ← getTruthTable(num_inputs, P)
      B_o ← append(B_o, tt)
      subKey ← getNextKey(1 << numInputs)
      tt ← physicalXform(tt, K_p)
      tt ← logicalXform(tt, sub_key)
      OI_Status ← oiXform(P, sub_key)
      for each Fan In fi in P do
        FIOIS ← FIOIS | getStatus(fi)
      end for
      tt ← oiXform(tt, FIOIS)
      B_s ← append(B_s, tt)
    end if
  end for
end for
```

The mapper tool is cognizant of the architecture (both physical and logical) of the target device as defined by the configuration key. Device-specific modifications in the bitstream are done by the mapper tool in such a way that correspond to the specific architectural mutations. For example, if the order of LUT inputs is changed, the interconnect bits in the bitstream are correspondingly reordered for a functionally correct mapping.

III. Results

In this section, we provide a thorough security analysis considering brute force, side channel, destructive reverse engineering, as well as known design attacks. Using the secure FPGA mapping tool described in Section II-B, we generate results for a set of 10 benchmark circuits.

A. Security Analysis

We provide a security analysis for four possible attack scenarios, namely 1) brute force, 2) known design, 3) side channel, and 4) destructive reverse engineering. We assume that the attacker has knowledge of the bitstream format, and has access to the obfuscated bitstream.

1) Brute Force Attack: A brute force attack represents the most challenging and time consuming attack on the system. For a given design, there can be thousands of feasible interconnected LUTs. Modern FPGAs typically support up to 6-7 input functions for each LUT. Thus, there are a huge number of possible combinations, which can be represented by even a small number of LUTs and which grows rapidly as the number of LUTs increases. When factoring in the potential content bit inversion, the programmable interconnect network inversion, and other architectural modifications, both static and time variant, the process of modifying some LUT bits, their input ordering, and the connections between them, mapping to FPGA, and testing for proper functionality becomes intractable. For example, consider a small design with 128 LUTs with 6 inputs (64 content bits). Each content bit may or may not be inverted, and the correct ordering of bits is unknown. This can be represented as the number of permutations (64!)

TABLE III
MAPPING RESULTS AND QUANTITATIVE COMPARISON BETWEEN ORIGINAL AND SECURE BITSTREAMS

Benchmark Name	# CLBs	Crit. Path Nodes	Bitstream Size (Bytes)	D_1	D_2 (Original)	D_2 (Secured)	x Latency (sec.)
alu4	430	4	6878	8.00	1.68	8.00	1.14
apex2	520	13	8316	7.99	1.70	8.00	1.12
apex4	249	8	3974	8l05	1.32	8.00	1.14
des	973	12	15558	7.95	1.69	8.00	1.12
ex5p	159	4	2540	8.01	1.00	8.00	1.13
ex1010	387	9	6192	8.05	1.02	8.00	1.14
misex3	384	9	5554	8.01	1.61	8.00	1.14
pdc	996	6	15922	7.99	1.48	8.00	1.10
seq	506	8	8096	7.95	1.68	8.00	1.15
spla	894	12	14296	8.05	1.42	8.00	1.11

multiplied by the number of ways in which the LUTs can be connected ($^{128}C_6$). Hence, even with known input and output pairs, mounting such a brute force attack is not feasible in a reasonable time frame with current technology.

2) Known Design Attack and Bitstream Tampering: Known design attacks utilize small benchmark circuits (*e.g.* a single AND gate) mapped to the target FPGA, which enables an attacker to observe how the resulting bitstream changes. By launching this attack repeatedly, it is possible to reverse engineer the bitstream format – as well as the IP – in modern devices. It is also possible to tamper with the design for targeted malicious modifications. However, our approach can protect against known design attacks because it provides a *moving target defense*, whereby the architecture's logical security layer changes each time the bitstream is recompiled. Therefore, even if a small benchmark is repeatedly mapped to the target device, no new information about the architectural configuration is leaked.

3) Side Channel Attack (SCA): Compared with brute force attacks, SCA is a more refined and powerful attack. We first assume the attacker has used power analysis to discover the key to the logical security layer, which is generated at runtime. However, because the key generation uses non-linear functions, and so is not susceptible to machine learning attacks, the next key will not be known, and therefore the moving target defense due to the time-varying architecture will prevent deobfuscation of the bitstream. Furthermore, even in the unlikely case that the next key can be guessed correctly, the ability to deobfuscate one bitstream does not enable the attacker to maliciously modify the design so that they can map it to another device, because all other devices will have their own unique physical and time-varying security keys. This effectively eliminates the economic motivation for an attacker.

4) Destructive Reverse Engineering (DRE): DRE is an expensive and time consuming process, but it can reveal the inner workings of the device. We present two scenarios of using DRE attacks. In the first case, DRE is used to reveal the structure of the physical security layer by identifying which fuses have been programmed. This alone would not compromise bitstream security, since the logical key is generated at runtime, and different keys are generated during each re-programming. Furthermore, this will only reveal the physical key network for one specific device, and therefore is not economical, given the high cost of DRE. In the second case, DRE is used to reveal the CSPRNG structure. However, this too would not be sufficient, because every device still has a different physical key network.

B. Secure Mapping Results

We used the modified VTR mapper tool to investigate the effect of the secure bitstream transform on a set of common benchmarks from the MCNC benchmark suite [11]. With the VTR mapping flow, Verilog files were parsed into their BLIF representation using an architecture description file that defines a CLB as consisting of a 4-input LUT, a flip flop, and interconnect logic. Therefore, all benchmarks were mapped into a series of interconnected CLBs, with 16 content bits per LUT. These LUT content bits were taken as the bitstream for this particular FPGA. Unlike commercial FPGAs, which have a fixed size bitstream for each device, the bitstream sizes vary among benchmarks, since the number of available CLBs is a function of the resources required at run time by the tool. We present results in terms of inter- and intra- bitstream Hamming Distance. Inter-bitstream distance (D_1)is defined as the average distance between LUTs in the original bitstream (B_O) and the secured bitstream (B_S), as shown in Eqn 1. The intra-bitstream distance (D_2) is defined as the average pairwise distance between LUTs in a given bitstream.

$$D_1 = \frac{\sum_{i=0}^{N} HD(B_{O,i}, B_{S,i})}{N} \tag{1}$$

$$D_2 = \frac{\sum_{i=0}^{N} \sum_{j=i+1}^{N} HD(B_i, B_j)}{N} \tag{2}$$

While a high quality CSPRNG – especially one that is amenable to efficient hardware implementation – should be

used in the final design, we have used the standard *mt19937* generator for evaluation purposes. The average D_1 value was found to be normally distributed, with a mean and standard deviation of 8.00 ± 0.03 for the 16 bit LUTs. The value for D_2 for the original bitstream was found to be 1.5 ± 0.3, whereas the transformed bitstream D_2 result was nearly equivalent to D_1. This implies that, even with designs where the pairwise intra-bitstream LUT content only differs by 1 or 2 bits, the functionality can be effectively obscured. The addition of LUT content and output inversion logic will also affect the critical path delay. We assume the additional interconnect delay within the ALM is not significant compared to the XOR gate delay of 1.02 ns [12], This gives us an increase of around 2 ns per ALM. This yields moderate latency overhead for all benchmarks, with an average reduction of 1.14x in the maximum operating frequency.

IV. Cost Analysis Based on a Case Study

There is an inherent trade-off between the area/power/delay overhead and the level of security provided by the architectural modifications. The results presented in Section III-B represent a very high level of security, where every content bit in every LUT can be selectively inverted through an additional XOR gate, the LUT output can be selectively inverted, and when the select inputs to the LUT are permuted using a switching network. We can estimate the area overhead from such architectural modifications by adding the approximate area of the additional XOR gates, plus one switching network, to the area of a given ALUT. For the 65 nm Altera Stratix III FPGA, the area of one Logic Array Block (LAB) is estimated to be 0.0211 mm^2, and the core of the largest Stratix III (EP3LS340) has 13,500 LABs, comprising 72.4% of the total die area [13]. Thus, for a 4 μm^2 XOR gate implementation [12], additional XOR gates result in an overall 8.5% increase in die area (411 mm^2 to 447 mm^2). For SRAM FPGAs, this could potentially be reduced with a custom design leveraging the existing Q and $\overline{Q}$ signals and using pass transistors to select between them. Using Synopsys Design Compiler, and 90 nm cell library (with results scaled to 65 nm), we estimate the area of the switch network for the LUT select inputs to be roughly 135.5 μm^2, increasing the total die area to 465 μm^2. This does not consider the area of the programmable fuses used in physical key storage. As reported in Table II, physical key storage for the LUT select input ordering is appropriate; therefore, with a 6-input LUT, at most $\lceil log_2(6) = 3$ fuses can be used. Assuming an area of 15 μm^2 per fuse [14], the total area would increase to 466 mm^2, or 13% area overhead. In practice, programming every LUT content switch with a static value may not provide the highest level of security, since the content ordering will not vary with time. Instead, a smaller number of fuses (*e.g.* 128 or 256) can be used on certain inputs (with remaining inputs connected to the logical key network). This will help reduce overhead from fuse area and increase security against known design attacks.

V. Conclusion

We have presented MUTARCH, a distinctive approach to FPGA security that enables secure wireless reprogramming and protects diverse FPGA-based systems against piracy and tampering attacks in the field. The central idea of MUTARCH is to create architecturally unique devices so that adversaries cannot interpret a bitstream, or use knowledge about one device architecture to break into another device. MUTARCH provides a low-cost, low-overhead, and scalable protection mechanism against field attacks on FPGA-based systems. Furthermore, it can be used in conjunction with existing encryption techniques for additional security, and as a means to lock a specific bitstream to a specific FPGA instance, preventing potential economic gains from IP piracy. Although MUTARCH requires minor architectural change and device programming during manufacturing test, it does not impact the functional behavior of a mapped design. It also does not impact routing and FPGA resource utilization during application mapping, reducing compile-time overhead during firmware upgrades.

With increasing usage of FPGA devices in diverse application domains, including the emerging IoT space, effective protection of FPGA-based systems, and the IPs mapped in them, is paramount. MUTARCH is particularly attractive for the emerging IoT regime, which involves a large number of identical, connected devices, where using knowledge of one device architecture to attack another device is more feasible. Although we focus on bitstream tampering and piracy in this work, the proposed approach is promising in preventing side-channel attacks. This is because architectural mutation can effectively obfuscate the correlation between secret key and the side-channel signature. Finally, the overhead of the proposed approach, although modest, can be further reduced through enhanced security versus hardware overhead trade-off analysis.

References

[1] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Fpga-oriented security," *Introduction to Hardware Security and Trust*, 2011.

[2] A. Moradi *et al.*, "On the vulnerability of fpga bitstream encryption against power analysis attacks: extracting keys from xilinx virtex-ii fpgas," in *CCS*, 2011, pp. 111–124.

[3] S. B. Örs *et al.*, "Power-analysis attacks on an fpga–first experimental results," in *CHES*. Springer, 2003, pp. 35–50.

[4] É. Rannaud, "From the bitstream to the netlist," in *FPGA*, 2008.

[5] R. S. Chakraborty, I. Saha, A. Palchaudhuri, and G. K. Naik, "Hardware trojan insertion by direct modification of fpga configuration bitstream," *Design & Test, IEEE*, vol. 30, no. 2, pp. 45–54, 2013.

[6] P. Swierczynski, M. Fyrbiak, P. Koppe, and C. Paar, "Fpga trojans through detecting and weakening of cryptographic primitives," *IEEE TCAD*, vol. 34, no. 8, pp. 1236–1249, 2015.

[7] K. Huang, J. M. Carulli, and Y. Makris, "Counterfeit electronics: A rising threat in the semiconductor manufacturing industry," in *ITC*, 2013.

[8] J. A. Roy, F. Koushanfar, and I. L. Markov, "Epic: Ending piracy of integrated circuits," in *DATE*, 2008.

[9] J. Luu *et al.*, "Vtr 7.0: Next generation architecture and cad system for fpgas," *ACM TRETS*, vol. 7, no. 2, p. 6, 2014.

[10] R. Maes, A. Van Herrewege, and I. Verbauwhede, "Pufky: a fully functional puf-based cryptographic key generator," in *CHES*, 2012.

[11] S. Yang, *Logic synthesis and optimization benchmarks user guide: version 3.0*. Microelectronics Center of North Carolina, 1991.

[12] K. Dhar, "Design of a low power, high speed and energy efficient 3 transistor xor gate in 45nm technology using the conception of mvt methodology," in *CICCT*, 2014, pp. 66–70.

[13] H. Wong, V. Betz, and J. Rose, "Comparing fpga vs. custom cmos and the impact on processor microarchitecture," in *FPGA*, 2011.

[14] K. Matsufuji *et al.*, "A 65nm pure cmos one-time programmable memory using a two-port antifuse cell implemented in a matrix structure," in *ASSCC*, 2007, pp. 212–215.