FLATS: Filling Logic and Testing Spatially for FPGA Authentication and Tamper Detection

Adam Duncan*†, Grant Skipper*, Andrew Stern‡, Adib Nahiyan‡, Fahim Rahman‡,
Andrew Lukefahr*, Mark Tehranipoor‡, Martin Swany*

*Intelligent Systems Engineering, Indiana University, Bloomington, Indiana 47401 USA
†NAVSEA Crane, Crane, Indiana 47522 USA
‡Electrical and Computer Engineering, University of Florida, Gainesville, Florida 32611 USA
Email: adamdunc@indiana.edu

***Abstract*—Security-critical field programmable gate array (FPGA) designs traditionally rely on bitstream encryption and hashing to prevent bitstream modifications and provide design authentication. Recent attacks to extract bitstream encryption keys, and research in automated bitstream manipulation tools, have created a class of vulnerabilities involving post-synthesis low-level FPGA editing. Current authentication and tamper (e.g., malicious modification) detection approaches dependent upon hash-based comparison mechanisms and register transfer level safeguards are vulnerable to these post-synthesis exploits. In this paper, we propose FLATS, which provides filling logic and testing spatially to combat such vulnerability. FLATS fills unused lookup tables (LUTs) within the FPGA design and inserts infrared-emitting spatial watermarks into the partially used LUTs at the post-synthesis stage for physical authentication and tamper detection using backside infrared imaging. FLATS takes an existing synthesized design and re-purposes a portion of its LUT initialization to function as a watermark allowing for the detection of changes to the post-synthesis placement and initialization. Experimental results validate the FLATS architecture on a 28nm Xilinx FPGA with less than 12% look-up table utilization overhead and negligible compromises in power and speed.**

***Keywords*—FPGA; 3PIPs; watermark; infrared;**

I. Introduction

Field programmable gate array (FPGA) devices must navigate a delicate balance between reconfigurability and security. The reconfigurable hardware aspect of FPGAs makes them attractive candidates for system designers in search of high performance with less development time and cost than an application specific integrated circuit (ASIC). However, the inherent reconfigurability of FPGAs also enables new threat vectors for FPGA-centric hardware attacks such as hardware Trojan insertion [2], encryption key extraction [5],[6], bitstream modifications [4], and intellectual property (IP) reverse engineering [1]. FPGA vendors, such as Xilinx and Intel/Altera, have begun to address various security concerns by introducing more hardware security measures into each iteration of their hardware designs [15]. Nevertheless, hardware security researchers continue to find new vulnerabilities and attack vectors, creating a cyclical attack, defense, and counterattack scenario.

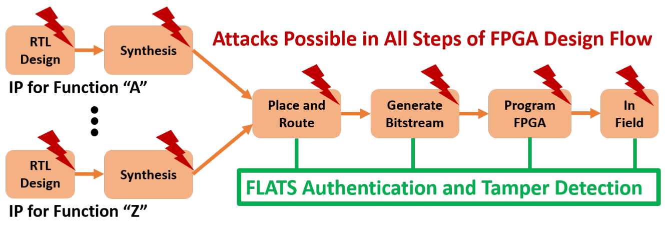

Fig. 1: A typical design flow for a FPGA instantiating 3PIPs is shown illustrating the potential attack vectors. FLATS provides post-synthesis tamper detection and authentication.

A typical FPGA design flow as shown in Figure 1, progresses through the following steps: *RTL design*, *synthesis*, *place and route*, *bitstream generation*, *FPGA programming*, and usage *in field*. Previous work to detect tamper events, such as hardware Trojans, have been focused almost primarily within the RTL design flow step. Multiple techniques have been published to detect Trojans in this step using structural analysis [17], functional analysis [16], logic testing [18], formal verification [19], and information flow tracking [20]. However, in reality an attack may be possible at any step within the design flow depending on the motivation and capability of the attacker. Recently published attacks and research in Xilinx 28-nm FPGAs have shown the ability to extract bitstream encryption keys [5],[6], defeat IEEE 1735p2 IP encryption standards [3], as well as create application programming interfaces (APIs) to automate bitstream manipulation [4]. These attacks provide the components necessary to initiate a post-synthesis attack that can either insert malicious functionality or replace sections of a design.

In this paper, we introduce the Filling Logic And Testing Spatially (FLATS) methodology for post-synthesis IP authentication and tamper detection. FLATS takes a synthesized design, instantiates filler logic, and then modifies all logic into dynamically controlled partial look-up table (LUT) oscillators. These oscillators are categorized as *beacons*, *authenticators*, and *detectors* and are configured for periodic activation during run-time to create a design-dependent watermark. The locations of these three oscillator types are measured insitu on a running design through backside Silicon infrared (IR) imaging to determine their precise spatial locations. Beacons serve as reference points and enable distance calculations between the authenticators and detectors, which depend upon the specific LUT inputs, including inputs involved in the original design logic. Modifications to *any* LUT configurations, will affect these distance calculations and enable the detection of a tamper event. In addition, the notion of *sequences* which select the type of oscillator, oscillator location, and oscillator enabling logic is presented. The sequence generation logic is a function of a user-generated input and an on-chip identifier integrated into a linear feedback shift register (LFSR) making it extremely difficult to reverse engineer. The major contributions of this paper are as follows:

- We experimentally demonstrate a post-synthesis attack on typical hash-based bit-stream authentication technique commonly used for encrypted designs in FPGAs with partial reconfiguration capability. The demonstrated attack provides strong rationale why it is extremely important to employ novel post-synthesis authentication and tamper detection schemes for preserving the security of the proprietary design and integrity of the system.
- We introduce the concept of embedding and dynamically enabling an 'infrared imaging-based' watermark into an FPGA core for physical spatial verification of the design.
- We propose a post-synthesis technique (FLATS) that fills unused logic elements and incorporates spatial watermarks into existing logic elements for tamper detection and authentication.
- We present the methodology of applying lock-in thermography to detect infrared watermarks through backside silicon and validate our proposed technique for bitstream authentication and tamper detection.

II. Background

Historically, the FPGA design flow following the synthesis step has been assumed to be secure because of proprietary bitstream formats, bitstream encryption and hash-based authentication. Recent research has exposed several vulnerabilities that should be addressed when considering the use of an FPGA in a security-critical system. We discuss possible attacks below for each step in the FPGA design flow presented in Figure 1. Lastly, we present a motivational post-bitstream attack on a SHA-3 IP core, which shows that existing countermeasures do not necessarily eliminate all vulnerabilities providing opportunities for new attacks.

A. Post-Synthesis Attacks

Following the synthesis step, a netlist is created that maps an original hardware description language design to programmable logic block (PLB) components such as LUTs and flip-flops (FFs). Malicious logic can potentially be added to this netlist if appropriate safeguards, such as netlist checksums and version controls, are not maintained. An attack within the design tool infrastructure has also been proposed at this stage in the design flow [11].

B. Post-Place and Route Attack

Following synthesis, place and route is performed to implement the PLBs into specific locations within the FPGA through the use of specific programmable logic interconnect (PLI) components. Upon completion of place and route, a direct netlist modification is not possible, but a change to the PLB properties, such as LUT truth table initialization *is* possible and can, therefore, be used to insert a hardware Trojan further discussed in Section E).

C. Post-Bitstream Attacks

Once a bitstream has been created, the entire configuration of the FPGA is located in a single file. To facilitate the programming of the FPGA, there is typically a protocol or format that is used to encode the bitstream and, commonly, is of public knowledge. For example, the bitstream encoding protocol for Xilinx 7-series FPGAs is documented in user guides allowing users to obtain the basic knowledge required for a bitstream modification [21]. The recently published BITMAN tool does just this, providing an API to perform manual edits on various 7-series and newer Xilinx FPGAs [4]. To prevent the unauthorized loading of a modified bitstream, the Xilinx security user guides provide recommended settings for utilizing AES-256 encryption and Hash Message Authentication Code (HMAC) checking during the bitstream loading. With these security provisions, both the AES and HMAC keys are required to load a modified bitstream into the FPGA. However, the AES key may be leaked or extracted as discussed in the following section. Additionally, the HMAC key may be leaked, or potentially reprogrammed into the FPGA by an adversary.

D. Post-Programming Attacks in the Field

Attacks are also possible in the field after an encrypted bitstream has been loaded into the FPGA. In this scenario, an attacker is assumed to have physical access to the system and is capable of extracting and decrypting the bitstream [5], [6]. The attacker begins by extracting the encrypted bitstream, which may be stored within the system on an external non-volatile memory such as a serial peripheral interface (SPI) Flash chip. Once the encrypted bitstream and bitstream encryption key are obtained, decrypting the bitstream is possible and a hardware Trojan may be inserted into unused LUTs via the above-mentioned bitstream manipulation. Finally, the modified bitstream can be re-encrypted using the bitstream encryption key and reprogrammed into the non-volatile memory. Similarly, the HMAC signature can also be modified to pass the authentication test by manipulating or obtaining the HMAC key stored within the FPGA.

E. Motivating Attack Examples

An attack on a design in any of the post-synthesis steps may take one of three major forms as illustrated in Figure 2. In this example, the original untampered design realizes $Z = (A\&B\&C)\hat{}D$, shown in Figure 2(a). The simplest attack involves the changing one of the behavioral properties of a

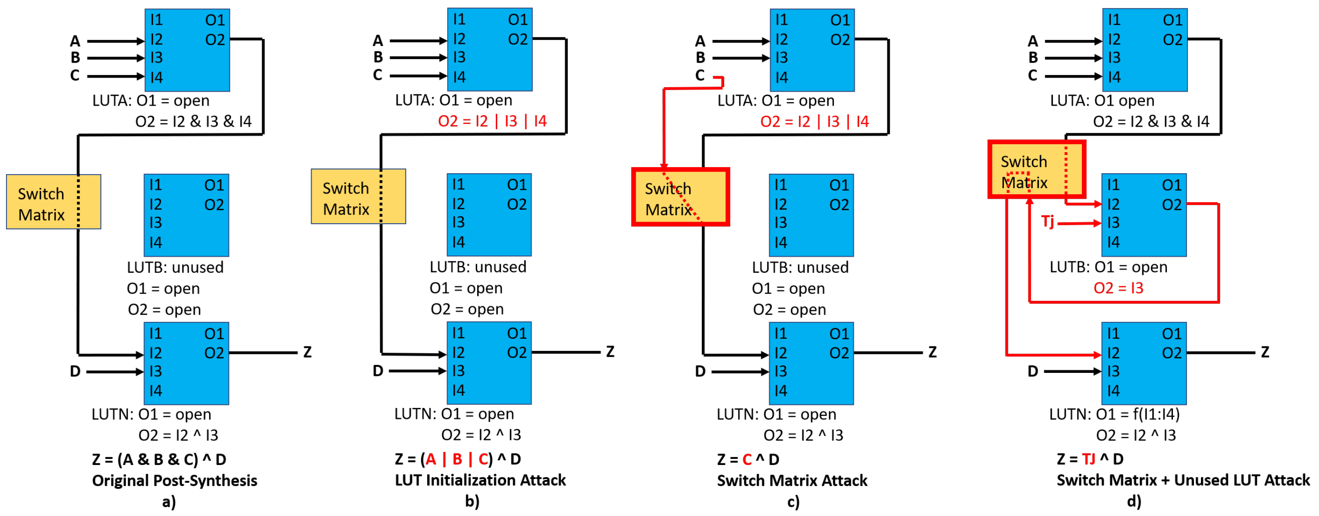

Fig. 2: Post-synthesis attack types: a) Original design, b) PLB attack, c) PLI attack, d) PLB + PLI attack

programmable logic block (PLB), such as modifying the LUT truth in LUT A so that $Z = (A \mid B \mid C)\hat{}D$ (see Figure 2(b)). Another attack can change a PLI, commonly used to implement a switch matrix connection to route another node to Z so that $Z = C\hat{}D$ (see Figure 2(c)). The third form involves the changing of both a PLB and PLI and can potentially be used to add malicious logic in an unused LUT and change the switch matrix connections to route this malicious logic to $Z = Tj\hat{}D$, where Tj is referred to as a hardware Trojan signal (see Figure 2(d)).

Here, we present a simple post-synthesis PLB attack on an open source SHA-3 512 authentication circuit [22] as further motivation for this work. The SHA-3 design was obtained from opencores and ported to the `Vivado 2017.2.1` software targeting a `xc7k70T Kintex-7` FPGA. Wrapper logic was added to the SHA-3 core to map different 512-bit inputs, along with a comparator output to indicate whether a successful 'match' was obtained. This design was validated via simulation and then loaded into the FPGA following the steps shown in Figure 1.

The attacker's goal is to modify the LUT configuration of the match comparator 'pass'/'fail' logic to force a 'pass', regardless of the input. We assume that the attack either takes place between the synthesis and bitstream generation steps, or takes place after bitstream generation using bitstream reverse engineering techniques [4]. We began by taking our post-place and route design and running a power analysis within the Vivado software. The power analysis is designed to report switching probabilities for design nodes to help estimate the dynamic power consumption. However, the power analysis also reports the probability of a given node being a logical '1', given the logical representation of the complete design. We then assumed that the node with the lowest probability corresponded to the final comparator decision logic for the SHA-3 match signal. With this knowledge, we manually changed the post-synthesis netlist LUT initialization for this 4-input LUT node to `16'hFFFF` so that the output is '1' in all cases. We then re-ran our simulations with correct and incorrect SHA-3 inputs and verified that the match output was stuck at '1' independent of the input provided. Next, we analyzed the bitstream and located the specific row, column, and minor frame address corresponding to this LUT. We then instantiated a simple state machine and internal configuration access port (ICAP) primitive into our test design to dynamically change the logic for this LUT during runtime. We experimentally verified our change by taking the *unmodified* SHA-3 design with the original LUT initializations, and used our state machine and ICAP to dynamically modify the single critical SHA-3 output LUT to report a pass under all conditions. Since both of these attacks involved the modification of pre-existing LUT configurations, they are detectable with FLATS, as will be discussed in Section IV.

III. Related Work

This section discusses current run-time authentication techniques, watermarking approaches, and previous work on filling unused logic in FPGAs and ASICs.

A. Run-time Authentication

The integrity of a bitstream loaded into an FPGA is typically verified by reading the FPGA configuration memory and comparing to a known reference. The configuration memory reading and subsequent comparison may occur over a test interface (JTAG), or may happen internally using the ICAP [14].

We successfully attacked a standard JTAG configuration verification operation using the run-time tamper scenario in Figure 3a. Here, we utilized a piece of 3PIP containing access to the ICAP to maliciously modify the FPGA configuration at run-time. We also implemented logic into the 3PIP to detect activity on the JTAG pins and to restore the original configuration upon detection. The restoration of the original

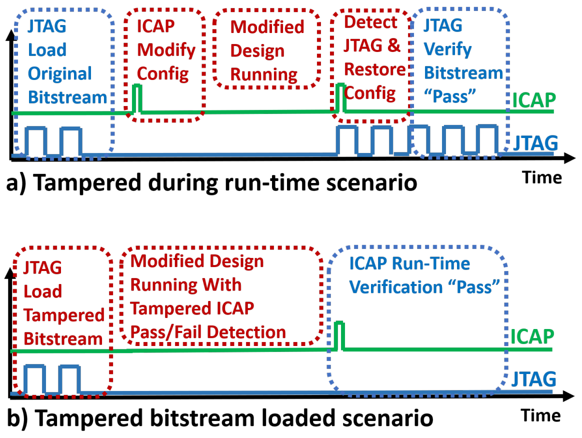
JTAG Load Original Bitstream
ICAP Modify Config
Modified Design Running
Detect JTAG & Restore Config
JTAG Verify Bitstream "Pass"
ICAP
JTAG
a) Tampered during run-time scenario
Time
JTAG Load Tampered Bitstream
Modified Design Running With Tampered ICAP Pass/Fail Detection
ICAP Run-Time Verification "Pass"
ICAP
JTAG
b) Tampered bitstream loaded scenario
Time

Fig. 3: Examples of run-time attacks that may not be detectable by the existing methods.

configuration data occurred *before* the JTAG verification operation completed, thus appearing as a successful 'pass' to the user.

Run-time verification approaches are also vulnerable the bypass attack conducted in Section II-E. Figure 3b illustrates this concept, where loading a tampered bitstream can be used to force the comparison result of the run-time verification to always report as 'pass'.

B. Watermarking Approaches

FPGA 3PIP watermarking research to date has focused primarily on providing proof of authorship for resolving IP piracy cases. These watermarks encode a design specific signature into a device property such as power consumption [7], or electromagnetic signatures [23]. With regards to tamper detection, there are several limitations with these approaches that are addressed in our work. First, the previous watermarks are designed primarily to provide a proof of authorship and do not provide proof of trust (i.e., whether the design is tampered). Second, these watermarks are typically static, and not available at run-time. Third, these techniques do not incorporate defenses against adding malicious circuitry in unused LUTs.

C. Filling Unused Logic

The concept of filling unused areas in an ASIC with filler cells incorporating a test methodology was presented as Built-in Self Authentication (BISA) in [9]. This approach was extended to the FPGA domain in recent papers as well [12], [13]. The FPGA approaches, however, concentrate primarily on the filling of unused logic and do not provide a comprehensive test methodology. FLATS fills the unused logic and incorporates the same verification methodology implemented in the original design LUTs to detect any tamper event in these newly filled LUTs.

A table comparing these works can be seen in Table I. Here, the HMAC comparisons can detect tampered IP, but cannot show proof of authorship or perform at the IP level.

TABLE I: A comparison of our FLATS work to existing approaches that provide proof of authorship and detection of tampered IP.

	Bitstream HMAC	Run-time HMAC	FSM WM	Power WM	EM WM	Filler Cells	FLATS
References	[21]	[14]	[7]	[7]	[23]	[12][13]	**This Work**
Proof of Authorship	No	No	Yes	Yes	No	No	**Yes**
Detect Tampered IP	Yes	Yes	No	No	Yes	No	**Yes**
Unique to FPGA SN	Yes	Yes	No	No	No	No	**Yes**
Fill Unused LUTs	No	No	No	No	No	Yes	**Yes**
Run-Time Technique	No	Yes	No	Yes	Yes	No	**Yes**
IP-Level Technique	No	No	Yes	No	No	Yes	**Yes**

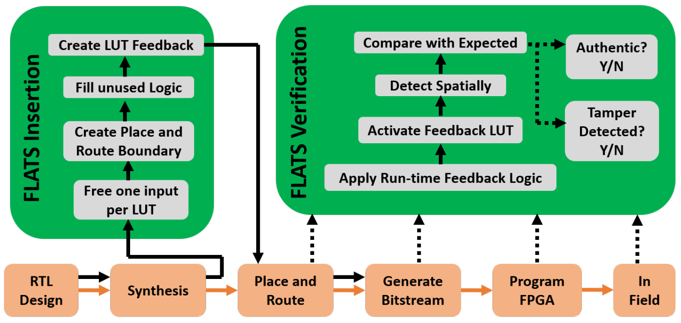

Fig. 4: FLATS architecture is introduced into the standard FPGA design flow for tamper detection and authentication.

Watermarking (WM) approaches can provide a proof of authorship, but cannot detect tampered IP or the filling of unused LUTs. FLATS addresses all of these areas in a using a single infrared imaging infrastructure that will be described in the next section.

IV. FLATS Architecture

The FLATS architecture adapts to the original FPGA design flow as shown in Figure 4. It is composed of two main phases, *insertion* and *verification*. The insertion phase takes place either during or after synthesis and either at the IP level or at the design/synthesis level. During insertion, the design logic is remapped into LUTs that reserve one LUT input and one LUT output. A place and route boundary is created around the logic, unsused LUTs, and other PLBs are filled, and each LUT converts its unused input and output in a feedback configuration to create a potential oscillator. Additional logic is then added to determine the behavior of each oscillator, with the oscillators grouped into either beacons, authenticators, or detectors for the forthcoming verification step.

During the verification step, the beacons, authenticators, and detectors are enabled at run-time to generate signatures detectable in the infrared spectrum at precise spatial locations. Reference signatures are obtained by enabling different authenticators and detectors, and determining their distances from the beacons by the analysis of their infrared emissions. Once the reference signatures are obtained, authentication and tamper detection activities may be performed at any time by repeating a measurement, or creating a new measurement using information from the place and route step to approximate locations of the different LUTs within the design. This step may be performed at time after insertion, either with the final FPGA design, or with a preliminary bitstream and/or representative FPGA.

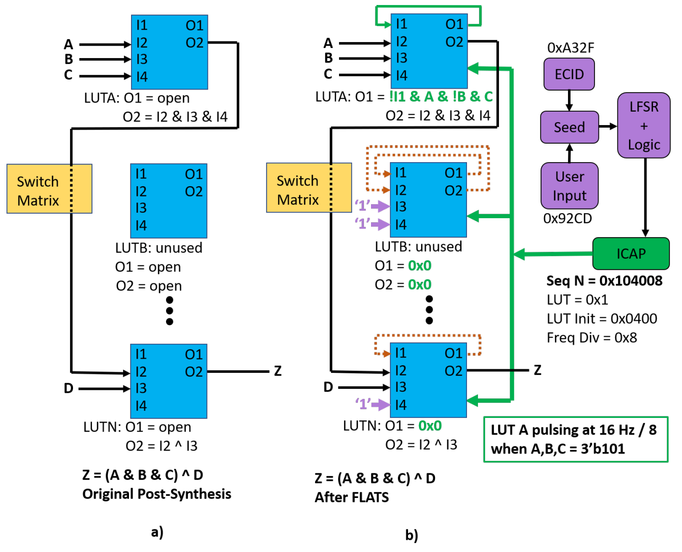

Fig. 5: a) Original post-synthesis design. b) Modified design after FLATS insertion and with a sequence applied to select LUTA.

A. Insertion

The process of inserting the FLATS architecture into the standard FPGA design flow is described in this section. A post-synthesis illustration of an unmodified design and a post-FLATS design can be seen in Figure 5 to assist in the discussion.

FLATS insertion begins by adding configurable oscillators to portions of the LUTs in a synthesized design. This is done by modifying the desired number of LUTs of a synthesized design so that one of the LUT outputs is connected to one of the LUT inputs in a feedback configuration to allow for potential run-time oscillator configuration. If a design has yet to be synthesized, a constraints file is used to mark the required input and output as unused. If a design has already been synthesized, logic remapping is used to remap the logic into this configuration. Additional unused LUT inputs are tied to a logic '1'. For authentication purposes, a small fraction of LUTs may be selected to achieve the desired confidence level. For tamper detection, the percentage of LUTs chosen is directly proportional to the detection probability. This technique also achieves significant tamper detection benefits when selecting LUTs with a primary output connected to a critical node.

Once the LUT modifications have been made, additional logic is added at the hardware description level (HDL) level to implement the control circuitry for enabling and disabling specific LUT oscillators. A LFSR is inserted to obfuscate the oscillator selection from the user input to defend against spoofing attacks by making it difficult for an attacker to learn the correlation between user input and oscillator selection properties. The LFSR seed input is a concatenation of the on-chip Electronic Chip Identifier (ECID) with a portion of the user input value. The remaining portion of the user input controls the amount of clock cycles to operate the LFSR before stopping the clock to obtain the LFSR output. The LFSR output is referred to as the *sequence*, and contains information regarding which LUT oscillator(s) to select, which frequency to enable and disable said LUT oscillator(s), and what LUT function to implement to tie the oscillation behavior (enabled/disabled) to the current state of the original design. Clock gating circuitry is added to the original design so that the controller may 'pause' the original design, preserving state, for tamper detection. Clock divider circuitry is added for implementing the LUT oscillator pulsing. Lastly, an interface to the on-chip ICAP controller is created to generate the appropriate frames for configuring the LUT initialization values of the selected LUTs.

An example illustration of the ICAP writing a configuration frame is shown in Figure 5b. A sequence of `0x104008` is produced by the controller logic which corresponds to the selection of LUT `0x1` for LUTA, with a LUT output $O1$ initialization of `0x0400`, and a clock divider divisor of `0x8`. The LUT initialization corresponds to equations

$$O1 =\sim [!I1 * I2 * !I3 * I4]$$

which when rewritten in terms of the circuit nodes, looks like

$$Z =\sim [!I1 * A * !B * C]$$

The equation states logically that the oscillator built into the LUTA $O1$ output should oscillate when A,C = '1' and B = '0'. The equation for $O1$ in general can be written as

$$O1 = f(I1 : I4)$$

which covers all permutations of the LUT inputs. This example makes use of a 4-input, 2-output LUT for simplicity and can be extended to the more common 6-input, 2-output LUTs found in modern commercial FPGAs.

A place and route boundary is created in preparation for the place and route step. A first pass of place and route is then performed which places the synthesized elements and controller logic within the boundary and creates routing connections. All 'placed' cells after the place and route step are fixed, with their placement constraints saved to the master constraints file to ensure their placement remains unchanged. The placement information is then exported to a script which checks for unused LUTs within boundary and applies a filling algorithm upon detection. The filling algorithm edits the synthesized design to instantiate each unused LUT with a feedback path with each unused output connecting to a single unused input, as described in the previous sections. The remaining inputs on these newly instantiated LUTs are set to a logical '1'. The filling algorithm edits the master constraints file to also include the placement of the filler cell(s). A second pass of place and route completes the routing connections for all cells placed within the master constraints file. At this point, the design is in a mature state to perform the registration step for use in the forthcoming authentication and tamper detection activities. From here, the design can then progress through the standard integration and bitstream generation steps, or be archived in a checkpointed format. The registration process begins with a post-place and route simulation to select several LUTs from the design, typically around corners or other spatially significant points to serve as 'beacons'. The sequences responsible for selecting the beacons are obtained and the user inputs are calculated using the LFSR algorithm for their generation depending upon the ECID and a selected user inputs. These user inputs and their beacons are then 'registered' for future use in the verification process. The authentication step within the registration process selects several non-beacon LUTs, referred to as 'authenticators' and runs similar simulations to determine the appropriate sequences for their activations. The locations of the authenticators can be chosen to resemble a pattern, or may be chosen at random. Each authenticator is then registered for future use in the authentication step of the verification process. For tamper detection, the registration of 'detectors' may be made by simulating the functional design to specific point, pausing the design and then choosing LUTs around particular nodes of interest and determining and registering their sequences.

B. Verification

The verification step in the FLATS architecture uses run-time imaging of the FPGA to record the spatial locations of the beacons, authenticators and detectors as described in the previous section. An illustration of these elements while at the place and route stage and after performing image analysis is shown in Figure 6.

FLATS verification begins by aiming and focusing an IR imager directly above FPGA surface. Many Xilinx and Intel/Altera sub-45nm FPGAs are available in flip-chip packages with the backside Silicon already exposed, making them ideal candidates for backside infrared and photon-emission imaging. Infrared imaging equipment is significantly less expensive than photon emission equipment. An in-situ infrared-based solution is also possible with the FLIR Lepton imager, which is available in a cell phone camera form factor. Photon emission equipment typically allows for finer spatial resolution, albeit with a smaller field of view and at an increased cost.

After setting up the imager, the sequences obtained in the registration process can be applied to enable the desired LUT oscillator. Upon the enabling of a sequence, frames may be captured by an imager for post-processing. During post-processing, lock-in analysis techniques are applied to determine which pixel correlates most strongly to the expected sequence frequency. The beacons should be enabled first individually to establish points of reference. Once the beacons have pixel coordinates established, authenticator and detector sequences may be applied with their euclidean distance calculated from each beacon and added to the appropriate registration entry. Results obtained from the verification process can be used in the authentication and tamper detection activities discussed below.

1) Authentication: An authentication of a post-synthesis design that has been embedded with the FLATS methodology is possible at any step once registration has been completed. An illustrated example of a successful authentication from the

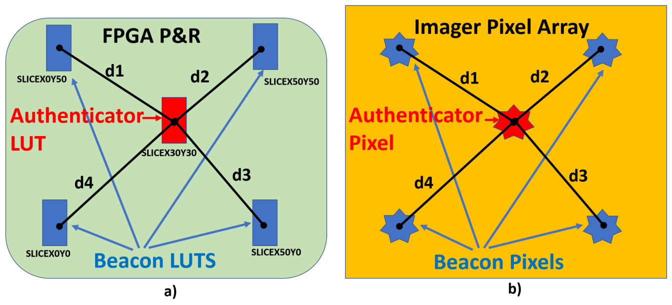

Fig. 6: (a) A trusted IP vendor inserts a watermark composed of ring oscillators (RO) into the IP. (b) An end-user physically verifies the watermark at a later stage using an IR camera to determine authenticity.

viewpoint of place and route LUTs and the pixels detected during imaging is shown in Figure 6. The simplest scenario to authenticate a design involves a user running the beacon sequences while imaging as described above to obtain pixel location of the beacons in the design. Next the user applies the detector sequences to determine the pixels corresponding to each detector. Distances between each authenticator pixel and each beacon pixel can be calculated and compared to a pre-computed reference value. These pre-computed reference values may be obtained via a trusted bitstream or may even be computed via the place and route coordinates of the associated LUTs in the FPGA design software. Indicators of an inauthentic design would be a lack of beacon pixels, or a difference between actual and expected distances among authenticator/beacon pairs.

2) Tamper Detection: Tamper detection can be performed in a similar manner to authentication once the verification step has been completed. First, the desired set of detector sequences is obtained for the desired number of LUTs. Next, pixel locations and associated distances to beacons are obtained for each sequences and stored as a reference for future comparisons. Once the reference values are stored, a comparison may take place by replicating a given sequence and comparing associated beacon differences to said reference. A deviation from the reference distance values would indicate a change to the LUT inputs of the LUT under test, or a change to the physical placement of the LUT. Each of which indicates a tamper event.

V. Experimental Results

Experiments were performed on a 28-nm `Xilinx Kintex-7` FPGA to verify and quantify the components of the FLATS architecture. A 28-nm Xilinx Kintex-7 FPGA with part number XC7K70TFBG676 was used as the FPGA under test. A long wave infrared (LWIR) `FLIR A325sc` 320 x 240 pixel, 60 frames per second camera was used as the imager. A 10 millimeter focal length lens achieving 25 micron per-pixel resolution used with the camera as well. The camera was mounted vertically using a Glide Gear table top camera mount with standoffs placed underneath the FPGA printed circuit board to achieve the 10 millimeter distance from the

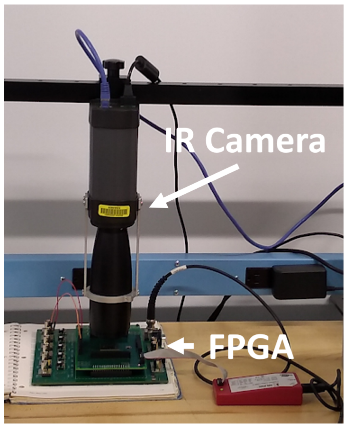

Fig. 7: The experimental setup is shown, including a Xilinx Kintex-7 FPGA and FLIR LWIR imager.

TABLE II: A comparison of ISCAS-85 benchmark circuit resource utilization before and after application of the FLATS partial LUT oscillator modifications.

ISCAS-85 ckt	Gates	LUTs pre-FLATs	LUTs post-FLATS	% LUT Increase
c432	160	60	51	0%
c499	356	59	63	6.8%
c880	293	68	76	11.8%
c7552	3512	270	301	11.5%

camera lens. All test conditions for the imager and FPGA were kept within the datasheet electrical specifications. An image of the experimental setup is shown in Figure 7.

A. Benchmark Circuits

Designs from the ISCAS-85 benchmark suite were utilized to test the post-synthesis modifications. The designs were initially obtained in gate level verilog and were subsequently synthesized using `Xilinx Vivado 2017.2.1`. Synthesis was first performed using standard Vivado constraints as a control, and then performed again while reserving a LUT input and one output from the dual LUT outputs for the FLATS implementation. Table II illustrates several benchmarks from this suite, their gate count, pre-FLATS LUT count and Post-FLATS LUT count to show the resource utilization before and after application of the FLATS partial LUT oscillator modifications. As seen, the resource utilization is less than 12% for the benchmarks.

The filling of unused LUTs was performed on c432 benchmark, with an image of the initial and post-FLATS place and route shown in Figure 8. The place and route boundaries, referred to as a 'p-block' by Xilinx are visible. It can be seen that before FLATS, there are several unused LUTs within the p-block (Figure 8(a)), and that those LUTs are filled and routed after the application of FLATS (Figure 8(b)).

Once FLATS was inserted into the designs, several sequences were chosen to activate LUTS as beacons, authenticators and detectors as described in the section IV. Figure 9 shows our Kintex-7 die, beacons and authenticators within the place and route step, a single IR image frame, and the

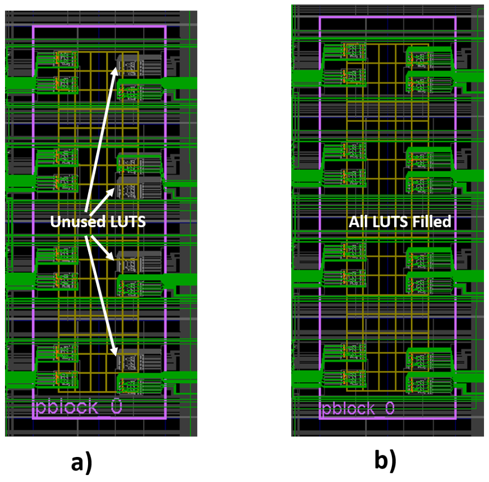

Fig. 8: a) Initial place and route. b) Place and route results after the filling of unused LUTs.

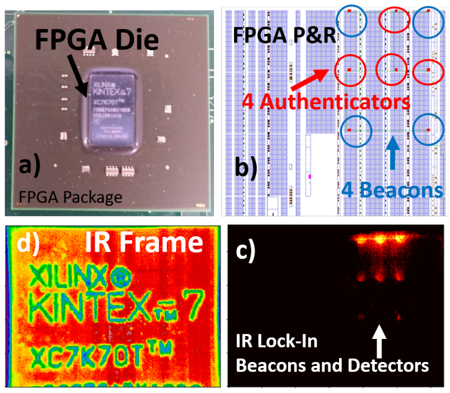

Fig. 9: a) Kintex-7 die. b) Beacon locations within place and route tool. c) Beacons detected after lock-in measurements. d) Single infrared image frame.

same beacons and authenticators after IR lock-in analysis. This particular implementation chose 4 LUTs to use beacons.

The power consumption of a single beacon, operating with and enabling/disabling frequency of 1Hz was measured to be approximately $15\mu W$. This measurement was performed by monitoring the supply current of the Kintex-7 VCCINT voltage supply over 600 periods. The actual oscillating frequency of the LUT was estimated to be approximately 300MHz. This estimate was obtained by routing the oscillator directly to an FPGA output pad and observing the frequency on an oscilloscope. The actual oscillating frequency of the LUT, however is not as important provided it dissipates sufficient power for IR detection. The important frequency is the *enabling and disabling of the LUT oscillator*, as this is precisely controlled by the FLATS controller circuitry and used in the lock-in measurements discussed in the next section. Because this enabling and disabling frequency is precisely controlled by the controller circuitry, it is immune to process, voltage, and temperature variations that plague oscillator-based physically unclonable functions.

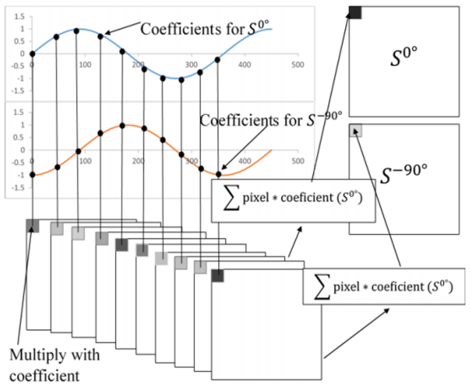
Coefficients for $S^{0°}$
Coefficients for $S^{-90°}$
$S^{0°}$
$S^{-90°}$
$\sum$ pixel * coeficient ($S^{0°}$)
$\sum$ pixel * coeficient ($S^{0°}$)
Multiply with coefficient

Fig. 10: Lock-in thermography principle illustrated by a reference signal with two phases [24].

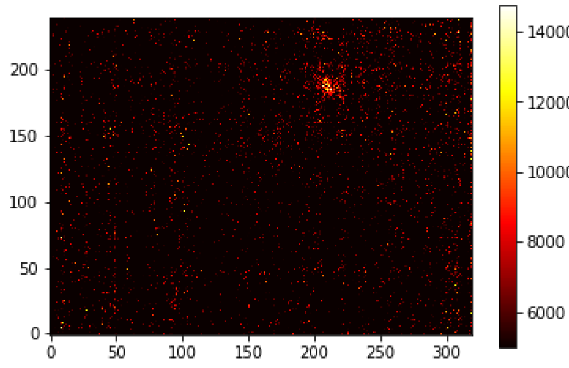

Fig. 11: Lock-in results a single detector sequence designed to activate pixel 208,180.

B. Infrared Imaging

The infrared imaging performed in these experiments utilized Python and the Activegige software development kit (SDK) from AB Software to communicate with the LWIR imager over the GigE interface. Images were acquired between 30 and 6000 frames per capture depending upon the desired accuracy. The longer captures acquired more periods of the LUT oscillator being enabled and disabled, allowing for better filtering of noise to pick out the relatively low infrared emissions coming from the LUT oscillators.

The infrared emissions resulting from the LUT oscillators are caused by power dissipating during the oscillation of the small circuit. The power dissipated generates heat, which in turn is present in the infrared spectrum. The total power dissipation is a sum of the static power and dynamic power. Since the static power is assumed to be constant, the changes in the infrared emissions can be attributed to the dynamic power. The primary component of the dynamic power is the power consumed during a switching operation,

$$Pswitching = a * f * Ceff * VDD^2$$

where, a is the switching activity or fraction of the circuit switching, f is the switching frequency, Ceff is the effective capacitance, and VDD represents the voltage supply. In our experiments, a is fixed at 0.5 for a 50 percent duty cycle, f is the switching frequency of the LUT oscillator when enabled, Ceff is dependent upon the number of cells loading the output, and VDD is the FPGA supply voltage VCCINT. If desired, one could then increase the magnitude of the infrared emissions from each LUT oscillator by increasing the VCCINT FPGA supply voltage. Increasing Ceff would have the negative side effect of decreasing f, since the circuit is essentially a ring oscillator. Likewise, increasing the switching activity ‘a’, by including more transistors in the oscillator feedback path, would also reduce the switching frequency f.

C. Lock-in Thermography

Lock-in thermography was applied to determine the pixel locations correlating to the LUT oscillators. The general concept of lock-in thermography generates reference signals at a specific frequency, with various phase offsets and multiplies these signals on a frame-by-frame basis to all pixels while calculating a running summation. The pixels with the largest summation for a given phase are mathematically more strongly correlated to the reference signals. This concept is shown in Figure 10 where S0 is a sine wave running at the reference detection frequency. S1 is the same signal with a 90 degree phase shift applied [24]. This concept is useful in our work, as the infrared magnitude of the switching power consumption of the single LUT oscillators is smaller than the infrared magnitude from the rest of the circuitry inside the FPGA. However, since the enabling and disabling frequency of these LUT oscillators is precisely controllable, and can be changed with different sequence values, it is possible to use these lock-in principles to determine their pixel locations.

The lock-in approach was experimentally verified using a 6000 frame capture with a authenticator sequence chosen to activate a single LUT oscillator at 1Hz. The reference waveform was generated as a 1Hz waveform in Python with the frame-by-frame multiplication and summation occurring in Python as well. Since the equation is applied to each pixel independently, a two dimensional array with size equal to the pixel resolution is created containing the lock-in magnitudes for a given phase offset. A plot of these magnitudes can be seen as a heat map in Figure 11. The bright area near pixel coordinates `[208,180]` is the area of highest correlation to the reference waveform.

The reference signal, raw IR magnitudes, and lock-in value summation from the first 600 frames of this capture for pixel `[208,180]` are plotted with respect to the frame number in Figure 12. It can be seen that the raw IR magnitudes do have a strong correlation to the reference signal and that the lock-in value increases relatively linearly with increasing frame number. As a complementary example, pixel `[0,0]` is plotted in Figure 13, which does not have a significant correlation to the reference waveform. As such, the lock-in value stays close to zero as the frame numbers increase.

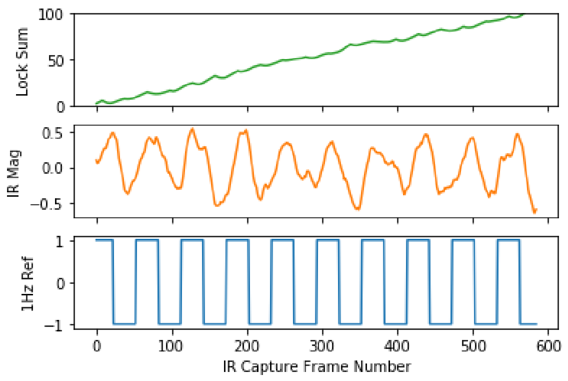

Fig. 12: The reference waveform, IR frame-by-frame magnitude of a watermarked pixel, and the cumulative lock-in summation.

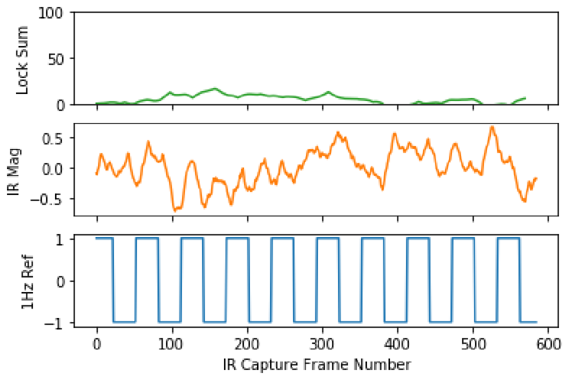

Fig. 13: The reference waveform, IR frame-by-frame magnitude of a non-watermarked pixel, and the cumulative lock-in summation.

D. Blob Detection

The lock-in magnitude arrays, such as the array in Figure 11, have visible blobs of higher pixel intensities that is larger than a signal pixel due to the properties of heat diffusion and the thickness of the backside Silicon. To detect the blob and the pixel representative of its center, the Laplacian of Gaussian (LoG) blob detection algorithm was performed. The LoG approach convolves the image by a Gaussian kernel and applies a Laplacian operator to determine whether a blob of a specific size was located. To detect blobs of varying sizes, the size of the Gaussian kernel was varied in a systematic fashion starting from large to small until a blob was detected. Once the blob was detected, the spatial mean of the blob was reported as the detected pixel.

E. Authentication Results

A real-time authentication was performed by instantiating the ISCAS 85 c432 benchmark circuit with the FLATS instantiation of 4 beacons and 4 authenticators as previously seen in Figure 9. The beacons are placed at the corners of the place and route boundary while 3 authenticators are places at the midpoint of boundary edges with the last authenticator placed roughly in the middle of the design. An initial verification run was performed with the golden design to establish a reference

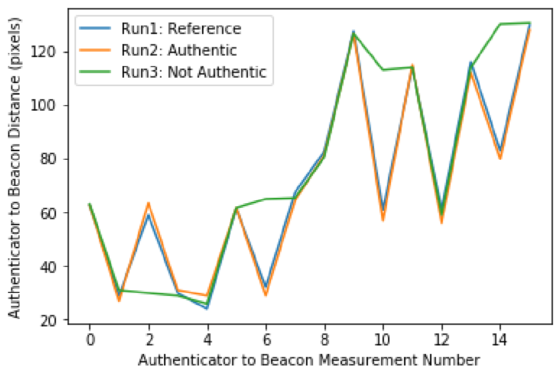

Fig. 14: Experimental results for FLATS design integrated into the ISCAS 85 benchmark c432. Registration, successful verification, and tamper detection events are shown.

pixel locations for the beacons and authenticator pixels, with the Euclidean distances computed between each beacon and authenticator. This reference design was then attacked by the movement of a single authenticator LUT to another slice. After the attack, the experiment was performed again and the distance between the attacked slice to beacon reference points were shown to vary by as much as 29 percent. A plot illustrating the distances computed by the FLATS post-processing step is shown in Figure 14. The x-axis in the figure includes all of the beacon/authenticator pairs and is ordered by beacon number. It can be see that every fourth measurement of one of the plot series experiences a significant change in distance, which is then used to label the design as not authentic.

F. Detection Results

Experiments were performed to evaluate various aspects of infrared imaging approach for the detection application. First, A FLATS instantiation of 32 beacon LUTs were placed in 8 adjacent slices with 4 LUTS per slice. The beacon LUTs were enabled and disabled in groups of 4 corresponding to their occupying slice at a frequency of 1 Hz for duration of 60 seconds. After lock-in analysis was performed, the means of each slice were plotted and had their pixel coordinates compared to their place and route coordinates. The pixel coordinates were shown to correspond to the place and route coordinates, with several places experiencing a mirroring effect, likely stemming from IC layout design choices during the FPGA IC floorplanning design. This can be seen visually in Figure 15, where mirroring is seen to occur horizontally between the bottom four slices.

Experiments were also performed to evaluate the accuracy of detecting and discriminating between the four single-LUT oscillators occupying a single slice. A FLATS instantiation of 4 detector LUTS were instantiated in a slice and activated individually at a frequency of 1 Hz for duration of 60 seconds. A total of 25 measurements were performed for each detector LUT to account for measurement error. After lock-in analysis was performed, the mean values of each set of 25 measure-

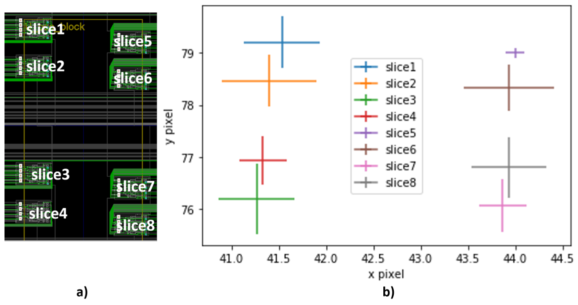

Fig. 15: a) Vivado place and route representation of 8 slices. b) Experimental verification of slice activation from infrared imaging.

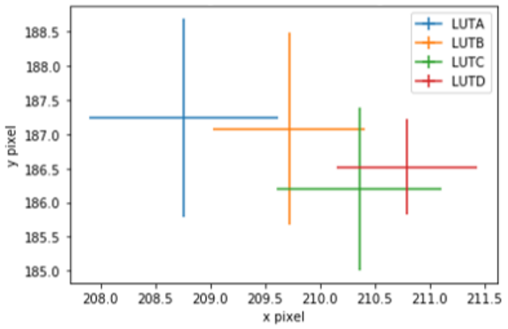

Fig. 16: Experimental verification of single LUT activation from infrared imaging.

ments were plotted, along with error bars to reflect a standard deviation. In all 25 cases, the LUT location was reported within 1 pixel of standard deviation. These pixel coordinates can be seen in Figure 16, where they appear to have slight overlaps between adjacent LUT error bars. This may be due in part to the other transistors outside of the LUT, but involved in the routing from the LUT output to the LUT input, being tightly co-located in the FPGA IC layout. This may be taken into account when implementing the FLATS technique by potentially utilizing a LWIR camera and lens combination with increased spatial resolution, or utilizing photon emission microscopy for the image acquisitions.

VI. Conclusion

The Filling Logically and Testing Spatially (FLATS) architecture was proposed here to provide authentication and tamper detection against post-synthesis attacks. Several post-synthesis attack concepts and example attacks were presented. The concept of embedding infrared-emitting dynamically controlled partial look-up table (LUT) oscillators was introduced to detect these attacks. Beacon, authenticator, and detector variations of the oscillators were presented and discussed. Backside infrared imaging techniques were presented to determine precise pixel locations of these oscillators for performing statistical analyses. Authentication and detection activities were experimentally verified using a 28-nm Xilinx Kintex-7 FPGA to demonstrate the effectiveness of detecting a post-synthesis attack down to a single LUT initialization value modification.

Acknowledgment

This work was funded in part by the Indiana Innovation Institute (IN3).

References

[1] T. Meade, S. Zhang, Y. Jin, Netlist Reverse Engineering for High Level Functionality Reconstruction, in *ASP-DAC*, pp. 655660, 2016.

[2] R. Chakraborty, I. Saha, A. Palchaudhuri, G. Naik, "Hardware Trojan Insertion by Direct Modification of FPGA Configuration Bitstream," in *IEEE Design and Test*, pp. 4554, 2013.

[3] A. Chhotaray, A. Nahiyan, T. Shrimpton, D. Forte, M. Tehranipoor, "Standardizing Bad Cryptographic Practice: A Teardown of the IEEE Standard for Protecting Electronic-design Intellectual Property," in *CCS*, pp. 1533-1546, 2017.

[4] K. Pham, E. Horta, D. Koch, "BITMAN: A Tool and API for FPGA Bitstream Manipulations," in *DATE*, 2017.

[5] A. Moradi, A. Barenghi, T. Kasper, C. Paar, "On the Vulnerability of FPGA Bitstream Encryption Against Power Analysis Attacks: Extracting Keys from Xilinx Virtex-II FPGAs," in *CCS*, pp. 111-124, 2011.

[6] S. Tajik, H. Lohrke, J.-P. Seifert, and C. Boit, On the Power of Optical Contactless Probing: Attacking Bitstream Encryption of FPGAs, in *CCS*, pp. 16611674, 2017.

[7] D. Ziener, "Techniques for Increasing Security and Reliability of IP Cores Embedded in FPGA and ASIC Designs," in *Ph.D. Dissertation at University of Erlangen-Nuremberg*, 2010.

[8] M. Tehranipoor, F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," in *IEEE Design and Test of Computers*, pp. 10-25, 2010.

[9] K. Xiao, M. Tehranipoor, "BISA: Built-in Self-authentication for Preventing Hardware Trojan Insertion," in *IEEE HOST*, pp. 45-50, 2013.

[10] O. Breitenstein, W. Warta, M. Langenkamp, "Lock-in Thermography: Basics and Use for Evaluating Electronic Devices and Materials," in *Springer*, 2010.

[11] C. Krieg, C. Wolf, and A. Jantsch, "Malicious LUT: A Stealthy FPGA Trojan Injected and Triggered by the Design Flow," in *ACM ICCAD*, pp. 43:143:8, 2016.

[12] N. Hazari, M. Niamat,"Enhancing FPGA Security Through Trojan Resilient IP Creation," in *IEEE NAECOM*, pp 362-365, 2017.

[13] B. Khaleghi, A. Ahari, H. Asadi, S. Bayat-Sarmadi, "FPGA-Based Protection Scheme against Hardware Trojan Horse Insertion Using Dummy Logic," in emphIEEE Embedded Systems Letters, Vol. 7, No. 2, pp. 46-50, 2015.

[14] T. Guneysu, I. Markov, A. Weimerskirch, "Securely Sealing Multi-FPGA Systems," in *Reconfigurable Computing: Architectures, Tools and Applications*, pp. 276-289, 2012.

[15] S. Trimberger, J. Moore, "FPGA Security: Motivations, Features, and Applications," in *Proceedings of the IEEE*, Vol. 102, No. 8, 2014.

[16] A. Waksman M. Suozzo, S. Sethumadhavan, "FANCI: Identification of Stealthy Malicious Logic Using Boolean Functional Analysis," in *CCS*, pp. 697-708, 2013.

[17] M. Hicks, M. Finnicum, S. King, M. Martin, J. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *IEEE S&P*, pp. 159-172, 2010.

[18] F. Farahmandi, Y. Huang, and P. Mishra, Trojan Localization Using Symbolic Algebra, in *ASP-DAC*, pp. 591-597, 2017.

[19] J. Rajendran, V. Vedula and R. Karri, Detecting Malicious Modifications of Data in Third-party Intellectual Property Cores, in *DAC*, pp. 1-6, 2015.

[20] A. Nahiyan, M. Sadi, R. Vittal, G. Contreras, D. Forte and M. Tehranipoor, "Hardware Trojan Detection through Information Flow Security Verification," in *ITC*, pp. 1-10, 2017.

[21] "Xilinx 7-series Configuration User Guide, UG470", v1.13.1, 2018.

[22] "https://opencores.org/project/sha3" [ONLINE], accessed Oct 2018.

[23] J. Balasch, B. Gierlichs, I. Verbauwhede, "Electromagnetic Circuit Fingerprints for Hardware Trojan Detection," in *IEEE EMC*, pp. 246-251, 2015.

[24] A. Stoynova, B Bonev, "Practical Consideration for Lock-in Thermography Effective Spatial Resolution", in *WSEAS TRANSACTIONS on ELECTRONICS*, pp. 66-72, 2017.