Soft-HaT: Software-Based Silicon Reprogramming for Hardware Trojan Implementation

MD MAHBUB ALAM and ADIB NAHIYAN, University of Florida
MEHDI SADI, Auburn University
DOMENIC FORTE and MARK TEHRANIPOOR, University of Florida

A hardware Trojan is a malicious modification to an integrated circuit (IC) made by untrusted third-party vendors, fabrication facilities, or rogue designers. Although existing hardware Trojans are designed to be stealthy, they can, in theory, be detected by post-manufacturing and acceptance tests due to their physical connections to IC logic. Manufacturing tests can potentially trigger the Trojan and propagate its payload to an output. Even if the Trojan is not triggered, the physical connections to the IC can enable detection due to additional side-channel activity (e.g., power consumption). In this article, we propose a novel hardware Trojan design, called *Soft-HaT*, which only becomes physically connected to other IC logic after activation by a software program. Using an electrically programmable fuse (E-fuse), the hardware can be "re-programmed" remotely. We illustrate how Soft-HaT can be used for offensive applications in system-on-chips. Examples of Soft-HaT attacks are demonstrated on an open source system-on-chip (OrpSoC) and implemented in Virtex-7 FPGA to show their efficacy in terms of stealthiness.

CCS Concepts: • **Security and privacy → Malicious design modifications**;

Additional Key Words and Phrases: Hardware Trojan, unauthorized memory accesses, kill switch

ACM Reference format:
Md Mahbub Alam, Adib Nahiyan, Mehdi Sadi, Domenic Forte, and Mark Tehranipoor. 2020. Soft-HaT: Software-Based Silicon Reprogramming for Hardware Trojan Implementation. *ACM Trans. Des. Autom. Electron. Syst.* 25, 4, Article 35 (June 2020), 22 pages.
https://doi.org/10.1145/3396521

1 INTRODUCTION

The unprecedented pressure of time-to-market and the ever-increasing cost of integrated circuit (IC) design and fabrication processes are forcing modern IC design flows to rely on intellectual property (IP) blocks from third-party vendors and fabrication at third-party facilities. These untrusted entities introduce the risk of including malicious circuits known as Trojans into the hardware. In the last decade or so, design of hardware Trojans has received significant attention [10, 25, 47, 59] with more Trojans designed at the register transfer level (RTL) and gate level.

Authors' addresses: Md M. Alam and A. Nahiyan, Florida Institute for Cybersecurity Research, University of Florida, P.O. Box 116200, Gainesville, FL 32611, USA; emails: {mahbub.alam, adib1991}@ufl.edu; M. Sadi, Auburn University, USA; email: mehdi.sadi@auburn.edu; D. Forte and M. Tehranipoor, Florida Institute for Cybersecurity Research, University of Florida, P.O. Box 116200, Gainesville, FL 32611, USA; emails: {dforte, tehranipoor}@ece.ufl.edu.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Association for Computing Machinery.
1084-4309/2020/06-ART35 $15.00
https://doi.org/10.1145/3396521

However, fewer Trojans have been designed to be inserted into the IC layout [9, 63]. Becker et al. [9] proposed changes to transistor dopant location and density to implement an "undetectable" hardware Trojan; however, Sugawara et al. [54] later debunked this claim by demonstrating that these changes were indeed detectable by scanning electron microscopy (SEM). Later, Yang et al. [63] proposed the A2 Trojan that leveraged analog circuits to create small and stealthy Trojans. Further, A2 Trojans could be triggered from the software level, thereby allowing remote attacks. Recently, however, Hou et al. [20] proposed the R2D2 technique to guard a set of sensitive signals and initiate hardware interrupt requests when unusual toggling events (e.g., those in A2 Trojan) occur.

The majority of the hardware Trojan design and detection techniques have focused on a type of Trojan trigger that becomes live immediately when the chip is powered on [10, 25, 47, 59]. Although these Trojan triggers are stealthy by design, it is theoretically possible to detect them by exhaustive logic tests as they are powered on. However, a Trojan that is not powered on during production test, wafer test, assembly test, or acceptance test will never be detected using any of the standard logic tests and side-channel tests. Moreover, if such a Trojan gets powered on without physical access, it could bring ghostly modification in hardware. The sophisticated approach one could use is establishing a software trigger to make the Trojan live. Once it comes into existence, the hardware Trojan awaits its trigger to carry out its intended payload. The software trigger is the attacker's secret that can be carried out for a chip used in a cloud/data center, on internet-of-things (IoTs) and other network-connected devices. The software program can either be directly used by the attacker or by injecting it as malware to power-up the Trojan. We call such Trojans, Soft-HaT, software enabled hardware Trojan.

In the domain of security, most work assumes that the hardware is trusted and immutable when it is deployed in the field in contrast to software which can be updated. The security measures are developed to mitigate threats such as malware, considering underlying hardware is trusted. For instance, malware changing the behavior of a system can be mitigated by anti-malware or authentic software since it does not damage the physical hardware of a system. In developing Soft-HaT, we demonstrate that the hardware can also be modified remotely and invalidate the concept of trusted immutable hardware. Thus, the software-based hardware change can undermine the existing security countermeasures. Further, modification in hardware is permanent; hence, software or hardware patch cannot fix it, unlike malware.

In general, once an IC has been manufactured, the internal logic design cannot be changed, but the non-volatile storage mechanisms can provide new opportunities to modify it. Non-volatile memories, such as the E-fuse, anti-fuse, and flash, are fundamental components of an electrical system that are used to store boot code, encryption keys, firmware, configuration code, and parametric yield recovery, among others. [6]. E-fuses are popular solutions in this regard since they allow permanent storage at the post-fabrication stage [27, 46] by electrically blowing a fuse. Here, Soft-HaT leverages the E-fuse programming feature to achieve the desired post-fabrication silicon changes.

We employ hardware programming to create software-enabled hardware Trojans at the post-deployment phase by modifying the existing logic using E-fuse programming. The E-fuse consumes a small area and can be fabricated with traditional CMOS technology [30]. Thus, this electric component can be easily added and hidden in large ICs. In addition, the remaining Soft-Hat components (trigger and payload) are inactive before the fuse is blown; therefore, Soft-HaT does not conform to any of the previously designed Trojans. Currently, there is very little effort in designing such Trojans. The closest would be the *A*2 Trojan [63]. However, one of the fundamental differences of the proposed design compared to existing work (like A2) is that an attacker needs to execute the software program to physically change the hardware logic and create the

Table 1. Comparison of Trojan Detection Techniques

	Trust-Hub [49, 50]	Moles [33]	Reliability [51]	Dopant [9]	A2 [63]	Soft-HaT
Insertion	Design	Design	Fabrication	Fabrication	Fabrication	Fabrication
Trigger	Hardware	Always on	Always on	Always on	Software	Software
Effect	Temporary	Permanent	Permanent	Permanent	Temporary	Permanent
Mutable	No	No	No	No	No	Yes
Testable	Yes	Yes	Yes	Yes	Yes	No

Trojan. This property makes this design very stealthy since it is non-trivial to detect such one-time modification in hardware by runtime detection techniques. One way to identify logic modifications is physical inspection using imaging. However, this method requires a known "golden layout" of the design that is very hard to obtain, especially for commercial off-the-shelf (COTS) ICs.

Comparison with existing Trojan designs. Table 1 presents a relative comparison between existing Trojan designs and Soft-HaT. Trust-hub Trojans [49, 50] in column 2 represent a series of standard Trojan benchmarks in which greater than 90% are assumed to be inserted at the design stage. These design-level Trojans are mostly triggered by a hardware component, such as a counter, meaning that they may not be triggered remotely, and their payload effect is temporary, meaning that the payload's effect is gone when the system is reset. Moreover, these Trojans have been shown to be testable/detectable by both pre- and post-silicon Trojan detection techniques [37, 38]. Moles [33] is also a design-level Trojan that is "always on," meaning that the attacker has no control over its payload delivery, as its effect is permanent. Moles is also vulnerable to side-channel-based detection techniques [40]. Reliability- [51] and dopant [9]-based Trojans are inserted at the fabrication stage and have a permanent effect on the system. However, these techniques provide no control over the Trojan trigger and have been shown to be detectable [54, 58]. *A2* [63] is a state-of-the-art Trojan that is also inserted at the fabrication stage. One unique feature of *A2* is that it can be triggered remotely. However, *A2* is also shown to be detectable [20] and its effect is temporary, and therefore it can never be used to implement a kill switch, which can permanently disable the system. All of these Trojan designs are not mutable after fabrication, and they remain powered on during the testing phase, which in theory makes them testable. The Soft-HaT, however, is mutable after fabrication and remain powered off during the testing phase, which makes it truly untestable. In addition, this feature allows Soft-HaT to implement a kill switch that can permanently disable the system. Soft-HaT also features software-based programming and a triggering mechanism that allow a remote user to attack a server or a cloud system. In addition, Soft-HaT's payload effect can be made permanent to design a kill switch that can instantly disable any system.

Our major contributions. We summarize the major contributions of this work as follows:

(1) We design a stealthy hardware Trojan, Soft-HaT, that becomes live by software-supported hardware programming. Since the Trojan is not powered on until programming, it can evade all existing logic tests and verification processes.
(2) We utilize existing IC signals, minimal analog circuitry, and a deliberately designed software program to permanently modify the hardware. This method offers the capability to remotely change the hardware of network-connected devices at the silicon level after fabrication or deployment in the field.
(3) We evaluate and verify the stealthiness of the proposed Soft-HaT against existing Trojan detection methods.

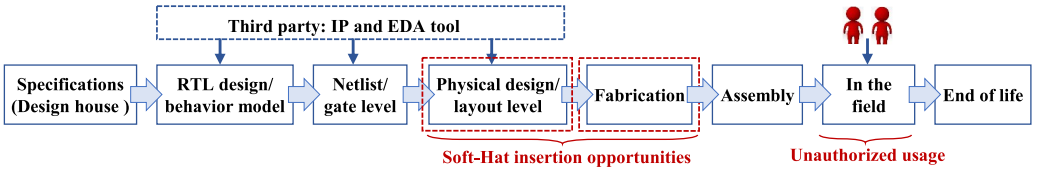

Fig. 1. Modern IC supply chain and vulnerabilities.

(4) We develop two examples of Trojan payloads to demonstrate the severity of attacks. One of them leaks data from privileged memory that is only accessible to authorized users and the other permanently disables critical systems.
(5) The proposed Soft-HaT design and payloads have been demonstrated in OrpSoC (an open source SoC with a RISC processor) in Virtex-7 FPGA.

The rest of the article is organized as follows. In Section 2, we discuss IC supply chain vulnerabilities focusing on the hardware Trojan. The proposed software-enabled hardware Trojan, Soft-HaT, and its threat model are discussed in Section 3. We describe the implementations in Section 4. We present payload designs in Section 5. Hardware demonstrations of Soft-HaT behavior are presented in Section 6. We examine the stealthiness of Soft-HaT in Section 7. Finally, Section 8 provides future directions and concludes the article.

2 PRELIMINARIES

In this section, we start with an overview of the IC design process and discuss the hardware Trojan threats.

2.1 Vulnerability of the IC Supply Chain

Advanced semiconductor technology requires prohibitive investment for each stage of the IC development procedure. For instance, the capital cost for a 40-nm-capable foundry is $20 billion [22]. For the newest technology node (e.g., 7 nm), the estimated average IC design cost is $271 million [22]. As a result, most semiconductor companies cannot afford to maintain such a long supply chain from design to packaging. To lower manufacturing cost and speed up the development cycle, the design houses typically outsource fabrication to a third-party foundry, purchase third-party IP cores, and use electronic design automation (EDA) tools from third-party vendors. The use of untrusted (and potentially malicious) third-party IP vendors and foundries increases security concerns of insertion or modification of hardware.

The complexity of IC design and fabrication has increased dramatically, as illustrated in Figure 1. At the first step of the process, IC specifications are translated into a behavioral description, generally represented by an RTL abstraction. The RTL design goes through extensive functional testing to verify the functional correctness of the IC. Next, RTL is synthesized with a manufacturing technology node into a netlist (i.e., a schematic containing logic gates, registers, and latches). The design house either integrates design-for-test structures to improve the testability in-house or outsources this step to third-party vendors. In the next step, logic gates are converted into physical layout format or GDSII (i.e., a set of planar geometric shapes and text labels representing transistors and their interconnections), and are handed to a foundry for fabrication. It is becoming more common for the design house to employ third parties for generating the physical layout and use hard IPs provided by third parties. The E-fuses used in Soft-HaT are added to the design in the deep layout by the foundry or these third parties. Once the foundry produces the wafers, each die goes through test. An assembly packages those ICs that pass wafer testing, re-tests them, and

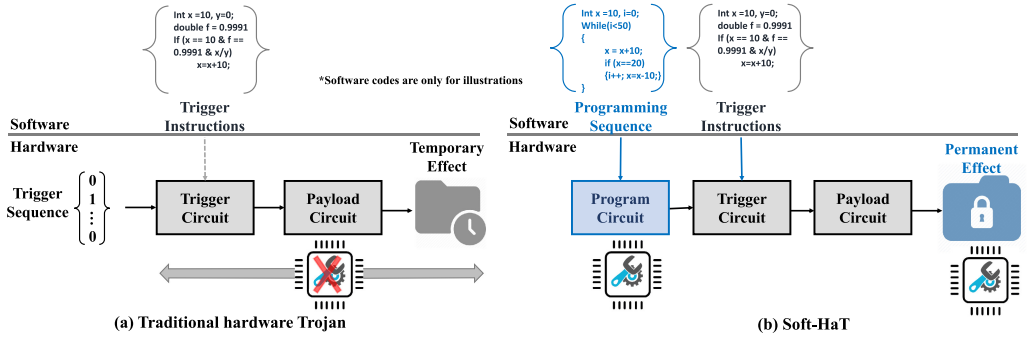

Fig. 2. Hardware Trojan operation in a traditional method and the proposed method.

sends them to the market. In an ideal world, the ICs are properly recycled at the end of the life cycle.

2.2 Hardware Trojan

A hardware Trojan is defined as a malicious, intentional modification of a circuit that results in undesired behavior when the circuit is deployed in a system. ICs containing a hardware Trojan may experience changes to their functionality or specification, leak sensitive information, experience degraded or unreliable performance, or suffer a denial of service (DoS) [4]. The modification can affect any type of IC, such as the microprocessor, graphics processing unit (GPU), digital signal processor (DSP), application-specific integrated circuits (ASICs), system-on-chip (SoC), and field-programmable gate array (FPGA). The Trojans can affect a system by themselves alone or provide a foothold for software-based attacks, where colluding software is aware of the inserted Trojan. A more detailed review of these threats can be found in various surveys [8, 11, 47].

Hardware Trojans are designed to be stealthy by intelligent adversaries. The Trojans could be inserted at the behavioral code, gate-level code, or physical layout stage. In this work, we consider that malicious hardware has been added at physical layout design, and thus changes are made to the IC layout. As mentioned earlier, foundry and third-party vendors are involved in layout design and fabrication; they are considered adversaries in this work. Figure 1 shows rogue/untrusted entities in the red boxes who can create Soft-HaT malicious hardware.

In general, a Trojan contains two basic parts: trigger and payload. A Trojan trigger is an optional part that monitors various signals and/or a series of events in the circuit. The payload usually taps signals from the original (Trojan-free) circuit and the output of the trigger. Once the trigger detects a pre-specified event or condition, the payload is activated to perform malicious behavior. Typically, the trigger is expected to be activated under extremely rare conditions, so the payload remains inactive most of the time. When the payload is inactive, the IC behaves like a Trojan-free circuit, making it difficult to detect the Trojan.

3 SOFT-HAT

We describe the fundamental concept of Soft-HaT attack and discuss the threat model in the following sections.

3.1 Basic Concept

Figure 2 shows a high-level overview of a traditional Trojan and Soft-HaT. A traditional Trojan is typically triggered by signals originating from hardware, such as a counter. This property limits the attacker to trigger the Trojan remotely. The A2 Trojan overcame this limitation by supplying

Table 2. Summary of the Soft-HaT Attack Model

Adversaries	– Foundry. – Third-party hard IP vendors.
Threats	– Evade the post-fabrication test and verification. – Hardware modifications after deployment. – Immune to software patch. – Can be designed to steal sensitive information (i.e., access privileged memory, obtain private key).
Assumptions	– IC is connected to a network. – Adversaries have access to layout file. – Adversaries have knowledge about IC behavior. – Physical attacks (e.g., optical probing) and focused ion beam attacks are not considered.

the trigger from the software level assuming that an adversary can run the triggering program (shown with a dotted gray line in Figure 2(a)). However, all existing Trojans remain powered on regardless of being triggered, meaning that they are testable. In addition, traditional Trojan payloads are mostly limited to hardware, meaning that they cannot be exploited from the software level. In addition, Trojan payloads have temporary effect, meaning that the effect will be nullified once the system resets. Soft-HaT addresses these fundamental limitations by making the hardware programmable after fabrication from the software level.

Soft-HaT introduces two additional components: program circuit and software payload (marked in blue in Figure 2(b)) to the traditional Trojan circuit. The program circuit utilizes the E-fuse to achieve post-fabrication programmability of hardware. The program circuit, as well as the trigger circuit, are controlled from the software level using an attacker's secret program. One of the novelties of Soft-HaT is that the trigger has no impact on payload until the Trojan becomes active. Thus, the Trojan can be triggered by using any internal wire, unlike the traditional methods where triggering conditions have to be rare to avoid detection. As a result, Soft-HaT provides a wide range of opportunities to carry out the attack remotely with a non-malicious program without compromising stealthiness. Once Soft-HaT is programmed, it can be triggered to deliver its payload both at the software and hardware levels. Software-level payload will allow the attacker to remotely retrieve secret information from the hardware, whereas the hardware payload will allow the attacker to modify the hardware logic and design specifications, and to make irreversible changes in hardware.

3.2 Soft-HaT Threat Model

In this section, we present the threat model of Soft-HaT attack. Table 2 shows the summary of the attack model, and details are described in the following sections.

3.2.1 Potential Adversaries. The proposed attack model can be implemented at the fabrication phase, as it contains analog circuitry that is integrated at the back end of the design. Thus, we consider that a foundry is a potential adversary to mount this attack. Another possible adversary is the third-party physical layout vendors, since they provide pre-verified "hard" IP. Such hard IP is often provided as a black box to the design house to maintain a competitive advantage in the niche market [25] and thus does not necessarily reveal itself to the designer. The proposed Trojan circuits are added to the layout design. We assume that other entities involved in the design process, such as IC design, verification, and synthesis, are trusted.

3.2.2 Capabilities of Adversaries. The potential adversary has access to the layout file and can perform reverse engineering to find the potential victim circuit to insert a Trojan. The adversary has some idea of the design that helps in designing effective Trojan payload. Design specifications and fabricated products that are available in the market could be used as a reference model to reverse engineer the layout file. One of the main challenges for the adversary is to maintain the layout structure as defined by the design process. Any modification in layout might alter the physical location for some or all of the design components, thus changing their delay and power characteristics that would facilitate Trojan detection. Soft-HaT uses existing wires to generate hardware programming signals and adds minimal analog circuitry to implement the Trojan payload. Given the large size of the modern IC, these additions are insignificant and unlikely to increase the dimensions of the chip or require existing components in the IC to be moved around.

3.2.3 Assumptions. The proposed threat model assumes that IC design follows the specifications. These lead to the fact the RTL design, placement, routing, layout design, test, and verification processes are appropriately conducted. Thus, it ensures that the logic design reverse engineered by adversaries is functioning correctly and Trojan insertion will result in the desired change/addition of functionality of adversaries. We assume that the manufacturing test follows state-of-the-art testing, such as structural and functional tests. The proposed Trojan is based on the E-fuse, which remains inactive as long as the proper software programming sequence is not executed. The current test procedure does not necessarily cover the test pattern for these hidden circuits and thus is unlikely to blow the E-fuse. We also assume that physical inspections cannot differentiate Soft-HaT from the rest of the circuit. We consider that Soft-HaT-inserted ICs are connected in a network and running in the field. An attacker can gain access to the connected device and execute a program to activate Soft-HaT. Network-connected systems are found in nearly every aspect of modern society, and they are vulnerable to Soft-HaT. For example, consider a modern SoC in a PC or smartphone, which are systems often connected to the Internet. As the user browses a website, a script from the website could execute the secret activation program on the SoC.

4 SOFT-HAT IMPLEMENTATION

Soft-HaT needs hardware components and a program to modify the inserted hardware. The hardware components are E-fuses and programming and trigger circuits. Additionally, we can include a sensor circuit that allows an attacker to read the programming status of the E-fuse. The software program that powers on Soft-HaT is known to the attacker. A brief discussion of each of these components is given in the following section. After that, we describe Soft-HaT programming and triggering mechanisms to carry out the intended payload.

4.1 Hardware Components of Soft-HaT

4.1.1 E-fuse. In general, hardware designs are static and cannot be changed once the designs are fabricated. However, the E-fuse has been extensively used by manufacturers and design houses to store information, calibrate post-silicon performance [30], provide integrity of firmware [24], and disable selected tests and debug [14], among others, after fabrication. In this work, we utilize the E-fuse to make changes in the hardware design. In contrast to the aforementioned common uses of the E-fuse, the changes we make to the chip occur post-deployment rather than at the fab or original equipment manufacturer (OEM).

The E-fuse exploits electro-migration phenomena to change the resistance of a metal interconnect from an unprogrammed to a programmed state [55]. An unprogrammed E-fuse exhibits a low resistance state (50 to 100 ohms) and therefore represents a short circuit. Once programmed, the E-fuse is said to be "burnt." A programmed E-fuse exhibits a high resistance state (approximately

Table 3. Technology Parameters of an E-fuse

Process	32-nm high-K metal gate
Cell area	1.37 μm^2
Operating voltage	0.5 − 1.5 V
E-fuse programming voltage	1.8 − 2 V
E-fuse programming time	1 μs

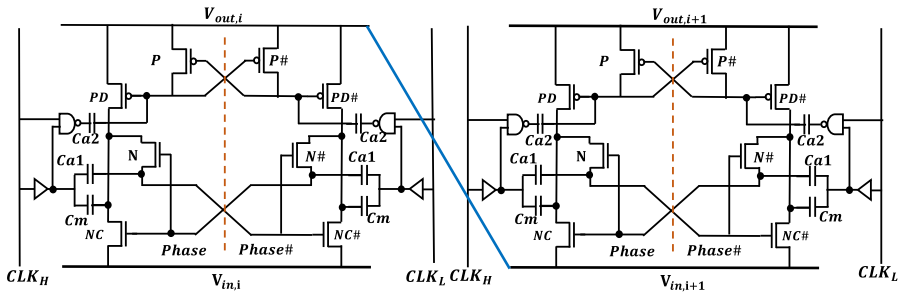

Fig. 3. The i^{th} and $(i+1)^{\text{th}}$ stages of the voltage doubler as the programming circuit. The number of stages depends on the design parameters and the required output voltage.

10 to 100 Kohms [1]) and thus can be represented as an open circuit. Table 3 presents the characteristics of and requirements for programming a state-of-the-art E-fuse for a 32-nm technology node. As shown in Table 3, the E-fuse requires a programming voltage approximately twice the operating voltage of gates in the IC. Therefore, additional circuitry is required to burn the E-fuse. The activation circuit shown in Figure 3 performs this operation. Table 3 also shows that the area footprint of an E-fuse is quite small (1.37 μm^2 in a 32-nm technology node), typically in the same order of a standard NAND gate. Therefore, it can hide deep in a modern SoC design that contains millions of logic gates.

Note that our proposed technique is also compatible with anti-fuse technology [36]. The difference between an anti-fuse and an E-fuse is that an anti-fuse exhibits a high resistance value in an unprogrammed state. It changes to low resistance when programmed. This work considers the E-fuse as a programming component, as it requires lower programming voltage than anti-fuse.

4.1.2 Programming Circuit. The programming circuit is responsible for generating a high voltage to burn the E-fuse. In many applications, such as power ICs, filters, memories, and switched-capacitor transformers, voltages higher than the power supplies are frequently required [16, 53, 57]. Voltage doubler circuits are CMOS compatible and widely used in many devices to convert on-board low voltage to high voltage. A voltage doubler can be integrated on chip for supporting the burning process of the E-fuse. Kulkarni et al. [29] presented the high volume manufacturing of E-fuses and demonstrated the programming process in 1-Kbit E-fuses fabricated in a 22-nm technology node. This design is fully compatible with low-power SoCs. Since Soft-HaT targets SoCs, the charge pump circuit designed and fabricated in Kulkarni et al. [29] is well suited for Soft-HaT. The output level can be controlled by the number of stages and/or the frequency. Figure 3 shows the basic stages (consecutive stages any i^{th} and $(i+1)^{th}$) of the charge pump that produces the voltage doubling. Each stage has two unit cells (*Phase* and *Phase#*) driven by non-overlapping clocks (CLK_H and CLK_L) and operates with complementary timing. In this work, a software program (known only to the attacker) is used to provide this clocking signal. Capacitors are used

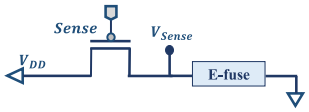

Fig. 4. Sensor circuit for Soft-HaT Trojan.

for the main boosting (Cm) and supporting capacitors ($Ca1$, $Ca2$) that drive the charge-discharge switches ($NC - PD$). The details of the circuit parameters are proprietary to the industry and are outside the scope of this work, as this work focuses mainly on the design of a hardware Trojan.

The design of the voltage doubler (i.e., the number of stages, the transistor, and capacitor parameters) depends on the input voltage and the required output voltage, as well as the frequency and the pulse width of the clocking signal. It also depends on the process technology used in voltage doubler topology. This work considers using an existing voltage doubler circuit that is currently in use in the industry. However, other CMOS-based voltage multiplier circuits, such as the Dickson charge pump [35], can also be used.

4.1.3 Sensor Circuit. Once the E-fuse has been programmed, the attacker needs to know its modified status. A simple sensing circuit has been incorporated into this architecture to read the status of the E-fuse. Figure 4 shows the design of the sensing circuit where the *Sense* signal activates the PMOS pass transistor and V_{Sense} provides the voltage across the E-fuse. The V_{Sense} signal provides a high voltage if the resistance of the E-fuse is high (i.e., in the programmed state) and low voltage if the resistance is low (i.e., in the unprogrammed state). The V_{Sense} signal is passed through a circuit to make it readable from the software level. The status of the E-fuse confirms that hardware modification has been completed. This change is permanent, and thus reading it once might be sufficient for the attacker.

The sensor circuit is an optional part of Soft-Hat and is included to provide the attacker with a confirmation that the E-fuse has been successfully burnt and the hardware modification has been completed. This confirmation facilitates the attacker to coordinate the release of Soft-HaT payload. In the absence of a sensor circuit, the attacker can run the trigger program but might not be able to directly observe the status of the E-fuse or the payload.

4.1.4 Trigger Circuit. Once the Soft-HaT programs the E-fuse, the trigger circuit can deliver the Trojan payload. The reason for having a trigger circuit in addition to a programming circuit is that the adversary may not want the effect of Trojan payload to be permanent and deliver the payload only during the attack to remain stealthy. The trigger circuit allows the adversary to deliver the payload during the attack. The input to the trigger circuit is similar to the input of the programming circuit. A specific set of opcodes and/or data can serve as a trigger. Alternatively, the trigger circuit can also be designed intelligently to make the payload more effective. The difference between programming and trigger circuits is that the programming circuit requires a long sequence of clocking signals to program the E-fuse, whereas the trigger circuit requires one or a few signals to deliver the payload.

4.2 Soft-HaT Program

The Soft-HaT program is the attacker's secret that is used to generate the clocking signal that burns the E-fuse. The program must not raise any suspicious behavior in both the hardware and software, meaning that it resembles a set of legal instructions, such as addition, subtraction, multiplication, and comparison. The program contains the specific instructions and/or operands that create the clocking signals for Soft-HaT. Hence, the program is non-malicious and does not raise concerns.

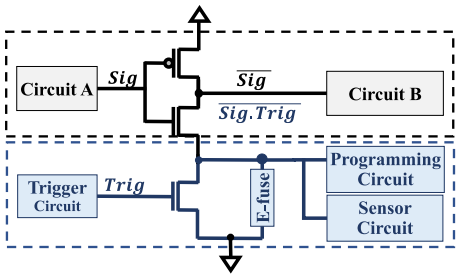

Fig. 5. Illustration of a Soft-HaT infected circuit.

A set of instructions is helpful to bypass any protection against running a code for several clock cycles.

Another requirement of the Soft-HaT program is that the probability of finding it in any application's program needs to exponentially low. Let us assume that L number of specific instructions and data required to power on Soft-HaT are n and m bits long, respectively. The probability of finding this exact program can be calculated as

$$P_{detect} = \frac{1}{2^{(m+n)*L}}. \tag{1}$$

Assuming a 32-bit architecture where both instruction and data are 32 bits and the required number of clocking pulses is 100, the calculated P_{detect} is 2^{-6400}. In other words, it is computationally infeasible to find the Soft-HaT program.

4.3 Soft-HaT Operations

Here we incorporate the Soft-HaT components into a simple inverter circuit to demonstrate programming E-fuse, sensing the E-fuse status, and then triggering techniques to deliver the payload. Figure 5 shows the inverter (black region) and Soft-HaT (blue region). Here, the inverter receives a signal Sig from circuit A and feeds $\overline{Sig}$ to circuit B. An attacker wants to control the Sig via the $Trig$ signal coming from the trigger circuit to implement a payload. The following sections discuss Soft-HaT programming and triggering to achieve the attacker's goal.

4.3.1 Programming Method. At first, Soft-HaT sends the clocking signal to the programming circuit received from the software. The programming circuit increases the voltage of the clocking signal (i.e., supply voltage) to the E-fuse programming voltage. The E-fuse programmer will generate the clocking signal until the sensor circuit confirms the programming status. Here, the software program is designed in such a way that the duration of the clocking signal meets the E-fuse programming time.

Note that the unprogrammed E-fuse in Figure 5 acts as a short circuit path to ground. Therefore, the $Trig$ signal from the trigger circuit has no impact on the overall design (i.e., the Sig input going to circuit B). After the programming phase, the E-fuse has very high resistance and modifies the circuit functionality from an inverter to a NAND. Instead of $\overline{Sig}$, $\overline{Sig \cap Trig}$ input will go to circuit B. In other words, Soft-HaT becomes live and the trigger circuit gains control of circuit B.

The programming method described previously causes the hardware structure itself to change, such as changing an inverter to a NAND gate. This change is permanent and irreversible. In addition, note that before Soft-HaT programming, all nets of programming, trigger, and sensor circuits are untestable. The reason is that the unprogrammed E-fuse prevents theses circuits from having any effect on the overall design.

4.3.2 Triggering Method. The attacker will initiate the triggering method once he or she knows that the programming has been completed via the sensor circuit. Here, the attacker uses a software program to generate the triggering signal $Trig$. In Figure 5, when the correct triggering signal is sent to the trigger circuit, it will deliver its payload by changing circuit B through the $Trig$ signal.

The preceding example shows how Soft-HaT can modify the existing circuit components to create a malicious one. Note that although we present this example using a simple inverter circuit, Soft-HaT can be implemented in any complex circuit.

5 SOFT-HAT ACTIONS: PAYLOAD DEMONSTRATION

One can design a sophisticated Trojan, but the severity of threats is ultimately bounded by the Trojan payloads. Attackers design payloads based on their attack goals, such as ranging from leaking sensitive information, creating an intentional side channel, changing the specification, disabling specific functionality, obtaining unauthorized access, or creating a DoS attack [8]. Soft-HaT is compatible with most of these payloads proposed in the literature given that payload can be inserted at the layout level.

A common assumption among Trojan researchers is that hardware Trojan payload needs to be hidden to prevent accidental activation or activation during standard testing. This concept is based on the fact that the Trojan payload is not detectable before deployment. It makes the attacker's job harder, and therefore the attacker needs to hide payloads under a complex trigger sequence. One of the novelties in our proposed attack model is that payloads become live in the field after hardware modification. Thus, it avoids the attacker's concerns of being detected in standard testing and provides a broad range of payloads.

Soft-HaT considers that Trojan-inserted ICs are connected in a network that allows an attacker to execute the program to activate Soft-HaT remotely. Since network-connected systems are in nearly every aspect of modern society, the attack surface of Soft-HaT is enormous. In this work, we consider two payloads to demonstrate the severity of Soft-HaT attacks as a proof of concept. One leaks sensitive information, whereas the other implements a kill switch to permanently disable a network-connected IC.

5.1 Test Payload I: Leaking Information

This Soft-HaT attack aims to transmit sensitive information from an IC to the attacker without the knowledge of the affected users. The ICs are in every aspect of our daily life and contain a lot of sensitive information, such as the cryptographic key, log-in credentials, financial documents, and personal information, and piracy of such information hampers social and economic life. The payload can be designed to access restricted memory that facilitates privilege escalation. For example, Meltdown [34] and Spectre [26] attacks on Intel and AMD processors exploit hardware vulnerabilities to steal data from privileged memory locations and to get hold of secrets stored in the memory of other running programs. These attacks exploit unintentional vulnerabilities in hardware; however, Soft-HaT intentionally creates vulnerabilities that are not present in the original design.

In this payload design, we add E-fuses into the memory management unit (MMU), which is responsible for memory protection and cache control, and translation of virtual memory addresses to physical addresses. The memory protection ensures that a process is not accessing memory that is not allocated to it. We aim to disable this protection in the proposed payload that facilitates unauthorized memory accesses.

A high-level block diagram of payload design is illustrated in Figure 6. We consider that Soft-HaT modifies the hardware to power on the Trojan and triggers sequences that can impact the payload. In general, the MMU converts the virtual memory addresses to a physical address to

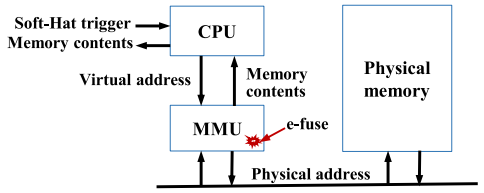

Fig. 6. Soft-HaT modifies the MMU and leaks restricted memory. For the sake of simplicity, only MMU, CPU, and memory blocks are shown.

provide requested contents to the central processing unit (CPU). It also maintains privileged memory accesses of processes. The proposed payload uses the trigger sequences to bypass the access restriction and allows malicious read. The payload can be extended by writing malicious contents to other processes or privileged memory. Pseudocode and experimental results of this payload are discussed in Section 6.

5.2 Test Payload II: Kill Switch

The Soft-HaT attack provides a broad range of possibilities for implementing kill switch and DoS attack on network-connected devices. Existing payloads that range from partial service degradation to complete and permanent disabling of a device can leverage the Soft-HaT attack. One of the primary challenges in designing the kill switch is to make irreversible modifications in the circuit, and existing payloads take a long time to do that. For instance, payloads generating accelerated aging [12] and consuming excess battery energy [44] require a long time to make an impact. The runtime detection techniques [20] or on-chip sensor could detect such long-term anomaly. However, the proposed Soft-HaT is capable of making an instant change on hardware by hardware programming. In the current state of the art, E-fuse programming time is less than a microsecond; thus, the kill switch can be instantaneous and does not provide enough time for detection and prevention. These characteristics make the Soft-HaT attack a dangerous kill switch.

Soft-HaT provides an excellent opportunity to create a controlled kill switch. An effective kill switch should permanently disable critical circuit components. We add extra logic using E-fuse to a microprocessor at the chip's layout design phase. There are a lot of critical circuit components, such as the bootloader, stack pointer, and program counter of a processor, driver, RAM, and non-volatile memory, to carry out the job of the kill switch. In our work, we choose the program counter circuit to modify the hardware. Accordingly, the E-fuse has been incorporated with the program counter of the processor. Soft-HaT allows access to these E-fuse devices through a specific program.

The program counter is one of the essential registers in a processor that contains the address of the next instruction going to be fetched [18]. As the current instruction gets fetched, the contents of the program counter are used as the address for fetching the next instruction. Thus, a processor in a system always keeps track of the address of the next instruction that must be fetched from the instruction memory. In this proposed method, permanent hardware changes do not allow to increase the program counter address and disables the regular program execution process. Figure 7 shows a program counter with added E-fuse programmable structure. For illustration purposes, we did not consider other factors, such as jump and branch. The M is the amount by which the program counter becomes updated in each cycle. The value of M is architecture dependent. The program counter values are being updated as long as the E-fuse stays at logic "1" as shown in Figure 7. The Soft-HaT program burns the E-fuse and results in permanent logic modification. As a result, the digital logic associated with the E-fuse is modified to logic "0" from "1." The program counter cannot be updated and becomes a constant address.

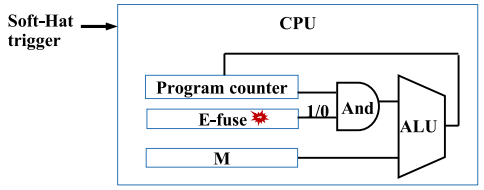

Fig. 7. Illustration of a kill switch to disable the program counter. Here, "M" is a constant added to the program counter value to fetch the next instruction.

Table 4. FPGA Implementation of OrPSoC

Core	LUT	LUTRAM	BRAM	Registers	Static Power
Processor	3,476	44	512	2,236	771 mW

6 EXPERIMENTAL RESULTS AND DISCUSSIONS

In this section, we demonstrate Soft-HaT attacks on an open source SoC benchmark and discuss the experimental results.

6.1 SoC Implementation in Hardware

We implemented an open source OrpSoC in FPGA to provide the proof of concept of Soft-HaT attacks. OrpSoC is an open source design provided by MIT Lincoln Laboratory [31]. In our experiment, an OR1200 processor [42], 32-KB of RAM, an 8-KB instruction cache, an 8-KB data cache, AES, UART, and JTAG cores have been used in the SoC. The cores communicate through wishbone buses. The OR1200 processor is an implementation of the 32-bit or1k instruction set. The test benches are developed using *c* and *assembly* programs to obtain the desired logic behavior from the SoC and are converted to low-level 32-bit instructions set using a GNU or1k tool chain.

A Xilinx Virtex-7 FPGA VC709 board [62] has been used to implement the SoC. The FPGA is manufactured at a 28-nm technology node and designed for high-performance applications. The block RAMs of FPGA have been used as RAM of the SoC that holds the 32-bit instruction sets. The standard UART modules are added to establish communication between the SoC and desktop. The JTAG interface has been utilized to read, write, and monitor the internal signal of the SoC. We used an FPGA onboard clock of 200 MHz for the system. Table 4 shows the hardware resources utilized for implementation. The standard logic gate behavior of SoCs are mapped into look-up tables (LUTs), registers, and LUTRAM. The FPGA used in this design does not offer the flexibility of connecting an eE-fuse to our desired logic block, such as the program counter or memory unit, and therefore limits us to perform E-fuse programming as required for the experiment. Thus, we inspect internal wires of the SoC for trigger signal and examine the E-fuse programming circuitry based on the observations of the silicon results.

6.2 Implementation of Soft-HaT

As discussed earlier, we used three steps for Soft-HaT implementation: Trojan programming, trigger, and payload. The hardware programming powers on the Trojan; trigger signals use the modified hardware to carry out the payload. We consider two different payloads regarding leaking information and the kill switch. The following sections provide experimental results to demonstrate them.

PSEUDOCODE 1: E-fuse Activation & Sensing

```
1:  procedure
2:      r0 ∈ E-fuse status register                  ▷ Programmed E-fuse: r0 = 1
3:      r1, r2 ⊂ General purpose registers           ▷ For comparison
4:      // initialization
5:      r0 ← 0
6:      r1 ← x                                       ▷ x ∈ random unsigned integer
7:      r2 ← x
8:      while r0 == 0 do
9:          Compare r1, r2                           ▷ Raise equal operand flag
10:         Read r0                                  ▷ Use custom instruction to read r0
11:     end while
12: end procedure
```

6.2.1 Implementation of Hardware Programming. The implementation process starts with the selection of victim wires that can be controlled by the Soft-HaT program. Since instructions are executed in a processor, wires in the processor provide decent controllability. The synthesized OR1200 processor has approximately 3,000 wires. Thus, the processor alone provides enough potential candidates for victim wires. Moreover, the analog circuitry for blowing the E-fuse requires a series of pulses. Thus, the Soft-HaT program needs to make sure that it can control the victim wire and generate the required pulses. Several wires, such as "operand equal," "greater than," and "less than flags," from the arithmetic logic operation unit of the processor meet the pulse train generation capability, and they are easily controllable from the Soft-HaT program. For the sake of demonstration, we choose the "operand equal" flag signal as the victim wire to provide pulses to the E-fuse. Note that attackers can choose wires from other IP blocks and consider low switching activity of the wire for victim wire selection to add more obfuscation on malicious E-fuse placement.

Pseudocode 1 presents the pseudo program for pulse signal generation to blow the E-fuse. Here, two equal operands stored in $r1$ and $r2$ registers are compared to set the value of the "equal operand" flag to logic "1." In the while loop, we check for E-fuse status from a register r0. Thus, the "operand equal" flag alternatively switches from logic "0" to logic "1" until the E-fuse is burnt. Since the E-fuse makes permanent changes in the hardware, modified logic is expected to last throughout the lifetime of the IC.

One of the challenging parts in this design is to read the status of the E-fuse since SoC does not provide direct access to an internal wire of hardware. An intelligent design process is required in this regard to avoid any change in regular functionality. In this demonstration, we take advantage of undefined logic in the OR1200 architecture to check the status of the E-fuse. The OR1200 architecture leaves few arithmetic instructions as unused. We used one of the unused instructions and modified the logic design to load the status of the E-fuse in a register. Attackers can also omit the step of reading the E-fuse status and try to run the payload directly to minimize the design effort.

Figure 8 illustrates the activation signal generated in the "operand equal" flag in FPGA implementation. We found that frequency of the activation signal is five times lower than the clock signal as the comparison operation takes five cycles in the OR1200 architecture and analog circuitry is designed accordingly. If necessary, the program can be designed to obtain other frequencies by inserting delay slots using a "no operation" instruction. Thus, the proposed method leads to frequency adjustment via a software program that adds more obscurity in Trojan design and provides a favorable condition to avoid the regular test and functional verification.

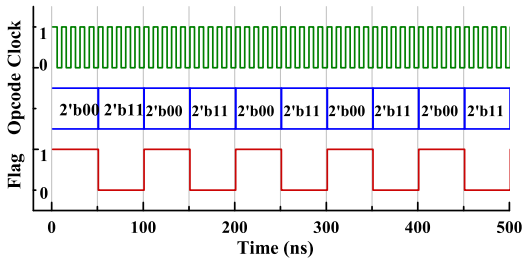

Fig. 8. Observation of the trigger wire in OrpSoC implemented in Virtex-7 FPGA. Here, flag refers to the "operand equal" flag, opcode 2′b00 means comparison, and 2′b11 represents reading r0.

```
data at address 0x0010a000: 0xffeeddcc
data at address 0x0010a004: 0xbbaa9988
data at address 0x0010a008: 0x77665544
data at address 0x0010a004: 0x33221100
```

Fig. 9. Reading protected memory using Soft-HaT attack.

PSEUDOCODE 2: Trigger Pattern Generation

```
1: procedure UNAUTHORIZED MEMORY ACCESS
2:     Check trigger status
3:     if trigger is activated then
4:         override MMU security protection
5:         disable data MMU
6:     end if
7:     print value of desired memory
8: end procedure
```

6.3 Implementation of the Soft-HaT Trigger

One of the main advantages of Soft-HaT is that a Trojan trigger has no effect on IC functionality until the Trojan becomes live. Thus, the Trojan trigger signals do not necessarily have to be rare, unlike the traditional methods, and leave plenty of choices to the attacker. For the sake of demonstration, we used "ALU opcode" that is not defined (illegal) in the OR1200 architecture to generate a trigger. In general, the architecture raises an "exception interrupt" in case of an illegal opcode. Our software programming step changes an illegal "ALU opcode" to a legal instruction and allows us to mount the attack.

6.4 Implementation of Payload I: Leaking Privileged Memory

We assume that Soft-HaT modifies hardware and allows us to run the trigger and mount the attack. The logic block of the MMU unit of the OR1200 processor has been modified to demonstrate the payload. The corresponding pseudocode to leak memory is given in Pseudocode 2. We started by executing a trigger instruction that modifies the MMU memory protection logic. Next, we send "load" instructions to read the privileged memory contents. The contents are read via a UART module connected to a desktop. We find that it allows to read memory contents that are only accessible in supervisor mode. Figure 9 shows the random memory contents that have been read from memory. Memory addresses $0x00010a000–08$ are privileged memory and only accessible in supervisor mode. This demonstration shows that the Trojan can cause a significant memory leak. A similar approach can also leak memory from other applications.

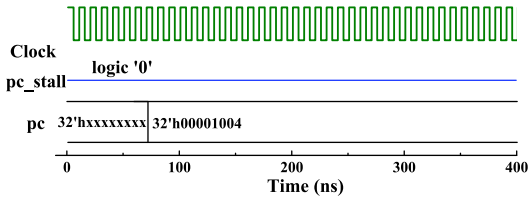

Fig. 10. Behavior of the program counter after kill switch activation.

PSEUDOCODE 3: Kill Switch Implementation

```
1:  procedure KILL SWITCH INSTRUCTIONS
2:      // Program E-fuse to change pc_stall permanently
3:      if kill_value then
4:          pc_stall ← 0
5:      end if
6:      // Restrict program counter to boot address
7:      if pc_stall == 0 then
8:          pc_reg ← boot_address
9:      end if
10: end procedure
```

6.5 Implementation of Payload II: Kill Switch

We implemented program counter deactivation as a kill switch in OrpSoC. The pseudocode added to CPU hardware logic is given in Pseudocode 3. Here, the kill value is designed by an attacker to perform hardware programming. We send the $kill_{value}$ to the SoC to deactivate it. As we have mentioned before that FPGA does not allow E-fuse programming in the logic block, we cannot make a permanent change in this case. However, we store that value using a logic call pc_{stall}. Pseudocode shows that pc_{stall} mimics the E-fuse behavior and make permanent changes in the program counter register. In this implementation, the program counter is stuck at the boot address of 32′h00001004 after hardware modification. The behavior of the kill switch is presented in Figure 10, showing that the program counter does not increase with time.

7 ANALYSIS OF STEALTHINESS

In this section, we present why traditional testing and verification techniques, as well as the previously proposed Trojan detection and prevention mechanisms, are not capable of detecting Soft-HaT. Moreover, as discussed in Section 4.3, the Trojan activation, trigger, and sensor circuits in our proposed architecture are untestable by the currently employed testing techniques and therefore itis inherently stealthier than the previously proposed Trojan designs. The stealthiness of Soft-HaT is examined with the following detection techniques.

7.1 Improbability of Accidental Activation and Detection

Soft-HaT powers on when an attacker provides the Soft-HaT program that generates the required number of clocking pulses that burn the E-fuse. As shown in Equation (1), the probability of generating the required number of clock pulses to burn the E-fuse through clock glitching or through a functional is extremely small (in the range of 2^{-6400}) for a 32-bit system. Some power side-channel-based Trojan detection techniques [19] rely on partial activation of the Trojan for detection. We develop an analytical expression to demonstrate that the power consumption of Soft-HAT due

to few clock pulses generated by accidental clock glitching or functional testing is negligible and cannot be detected by power side-channel-based Trojan detection techniques. Since the programming circuit (charge pump) is the largest portion of Soft-HaT and has parasitic components, we focus mainly on it for power analysis. The dynamic power of the programming circuit changes with frequency. For the sake of simplicity, the N-stage charge pump is modeled as a voltage source with an open-circuit output voltage ideally equal to $(N + 1)V_{DD}$ and an equivalent (series) output resistance $R_{out} = N/f2C$ following the work of Baderna et al. [5]. The dynamic loss is

$$P_{dyn} = k_1 N f C_{par} V_{DD}^2, \tag{2}$$

where k_1 is a technology- and topology-dependent coefficient, C_{par} is the total parasitic capacitance, and f is frequency of the pulse generated by the Soft-HaT program. The parameter C_{par} consists of the top plate parasitic capacitance (Cm, $Ca1$, $Ca2$) and the capacitance of the MOS switches. For a voltage doubler depicted in Figure 3, MOS switch capacitance is the sum of the gate capacitor of $NC\#$, the source-bulk capacitor of pre-charge NC, and gate and source-bulk capacitors of the PMOS pass transistors P and $P\#$, respectively.

In a sub-micron process, the parasitic capacitance is in the order of picofarads [32]. A few accidental clock pulses during the testing period correspond to very low frequency (in order of hertz). Therefore, the average dynamic power consumed by Soft-HaT due to accidental clock pulses is negligible as compared to the overall system (around 12 orders of magnitude smaller; see Table 4). Therefore, power side-channel-based Trojan detection techniques [19] would be incapable of detecting Soft-HaT.

As shown in Equation (1), the probability of generating the required number of clock pulses to burn the E-fuse through clock glitching or through functional testing is extremely small (in the range of 2^{-6400}) for a 32-bit system. Activation of Soft-HAT using structural test pattern is not feasible for the following reasons:

- The scan clock operates at much lower frequency than the frequency of the clock signal required to activate the E-fuse.
- Scan patterns generated from the automatic test pattern generator (ATPG) will never provide the repeating activation pulses required to burn the E-fuse.
- The structural pattern generated during the "design for testability" (DFT) stage has no knowledge of the Soft-HaT design that is added by the foundry at the fabrication phase.

Similarly, built-in self-test (BIST)-based techniques that are typically used for testing memory elements (SRAM, DRAM, etc.) would not be effective in detecting Soft-Hat. BIST only covers the nets that exist in the original design. Soft-HaT is inserted after BIST incorporation, and as a result, BIST cannot cover the nets and the E-fuses that are going to be inserted by the adversary.

7.2 R2D2 Trojan Detection Technique

The R2D2 detection mechanism analyzes a set of software-controllable registers or memory-related signals for successive toggling events to detect a suspicious event and the possible presence of a Trojan. The authors claim that their proposed technique is capable of detecting the A2 Trojan by analyzing the unusual toggling events in the divide-by-zero flag. However, this technique will not work for our proposed Trojan. As mentioned in Section 4.2, Soft-HaT is activated by a typical program that possesses no malicious properties. The programming and the trigger circuit is inserted in the design by the adversary (e.g., a rogue foundry after the R2D2 mechanism is added by the design house). Therefore, there is no possibility that R2D2 will cover wires of the E-fuse activation circuit.

7.3 Pre-Silicon Trojan Detection

Multiple pre-silicon Trojan detection techniques have been proposed that try to detect the Trojan in the design stage. These techniques range from code coverage [38] to HDL analysis [48], rare node identification [49, 56], finite state machine extraction [39], formal verification [15], and information flow tracking techniques [37]. These are typically developed to detect Trojans inserted by third-party IP vendors and requires access to the RTL or gate-level design. Note that in our proposed approach, the adversary inserts the Trojan in the layout shipped to the foundry, and therefore the design house does not have access to the Trojan-inserted design. Thus, none of the proposed techniques are capable of detecting our proposed Trojan.

7.4 Post-Silicon Trojan Detection

The post-silicon Trojan detention techniques can be broadly classified into functional verification tests and side-channel analysis. Traditional functional or structural testing for finding defects or bugs are not suited for Trojan detection [60]. Banga and Hsiao [7] and Chakraborty et al. [13] developed test pattern generation methods to trigger such rarely activated nets to improve the possibility of detecting Trojans. However, in our proposed technique, the design house does not know the programming or trigger circuit inserted by the adversary, and therefore these techniques cannot be applied to any wires of our Trojan circuit.

7.5 Side-Channel-Based Trojan Detection

Side-channel-based Trojan detection techniques rely on identifying the Trojan's effect on side-channel parameters such as leakage current [2], power [3], electromagnetic (EM) radiation [52], or delay [23] to detect the presence of a Trojan. Although these techniques generally do not require the Trojan to be completely activated, they require a golden reference (Trojan-free chip) to generate a golden side-channel profile or signature. Note that Soft-HaT is inserted at the fabrication stage, and therefore no golden chip or layout is available. He et al. [17] proposed to use an RTL design to model a design's golden EM signature and therefore do not require a golden chip or layout. However, Soft-HaT is introduced in the design along with analog circuitry that cannot be modeled with an RTL design available to the defender, and thus this technique cannot detect Soft-HaT. Hoque et al. [19] proposed a self-referencing-based technique that does not require a golden design but requires a Trojan-infected design to traverse through one or more Trojan states to create a difference in the power signature. However, Soft-HaT does not possess any states associated with Trojans, and therefore this technique also will not work on Soft-HaT. Park et al. [43] proposed to use the power signature of a malware program for malware detection. Soft-HaT is triggered by a typical program that possesses no malicious properties and therefore is not detectable by Park et al. [43]. Because of the presence of the E-fuse, and programming and sensing circuits, some extra resistance and capacitance are added to the pull-down path of the original inverter circuit (shown in the upper dotted box in Figure 5). The E-fuse exhibits around 50 ohms of resistance [29], and capacitances are in the range of tens of femtofarads [21]. This extra delay is very small (in the range of tens of picoseconds). Since this block (in Figure 5) will be part of a circuit path with other gates, the extra delay will be masked and essentially indistinguishable from process variations, power noise, crosstalk, and so on. Now, modern processors and SoCs operate at a few gigahertz frequency (a time period of nanoseconds). Therefore, the change in delay with respect to the time period is negligible. Moreover, in a real chip, there are millions of circuit paths, and hence an exhaustive path delay fault is impractical [28]. In addition, detection by transition delay fault (TDF) requires the delay to be large enough to mimic a stuck-at fault. Therefore, a delay test will be unable to detect the Trojan for all practical purposes.

7.6 Runtime Trojan Detection

The runtime Trojan detection mechanisms typically rely on a concurrent error detection mechanism to detect a Trojan inserted by untrusted 3PIP vendors. They assume that the comparison circuitry that detects any mismatch between the redundant modules indicates tampering [45]. However, in our case, the adversary, such as a rogue foundry, can tamper with the comparison circuitry. As a result, the comparison circuitry cannot identify the malicious modification created by Soft-HaT. Moreover, the adversary can insert Soft-HaT into the comparison circuit itself and disable the concurrent error detection mechanism when the Trojan is triggered.

7.7 Design for Trust

These Trojan defense techniques attempt to thwart the Trojan insertion by the adversary. An example of such techniques is the built-in self-authentication (BISA) approach that fills all empty spaces in the design with functional filler cells [61]. However, this approach does not take into account two aspects of modern IC design flow. First, many hard macros, such as SRAM memory blocks, TRNG, and some analog circuitry, are integrated into the modern SoC by the foundry. Therefore, an untrusted foundry has unlimited access to insert Soft-HAT in any of these hard macros. Second, engineering change order (ECO) cells, such as functional unused gates, are intentionally added in almost all SoCs for fixing timing violation, bugs, or functionality [41]. ECO cells cannot be protected by BISA, as these cells cannot be replaced by BISA cells.

8 CONCLUSION AND FUTURE WORK

This article discussed a stealthy hardware Trojan, Soft-HaT, based on hardware programming to change the computing logic when ICs are deployed in the field. Since Soft-HaT becomes live in the field, it is not testable in post-fabrication. We utilized Soft-HaT to perform an attack on an MMU that provided access to restricted memory. We also implemented a kill switch using Soft-HaT, which can instantly disable a system permanently. These approaches have been demonstrated in Virtex-7 FPGA. Since the functionality of the E-fuse programming circuit has only been verified via simulation and FPGA emulation, we plan to fabricate and verify Soft-HaT on an ASIC chip that allows implementation of analog circuitry and E-fuse programming. Further, we assumed that physical inspections would not detect Soft-HaT given the complexity and size of the modern ICs. In future work, we will verify this assumption using SEM imaging techniques to identify Soft-HaT in the fabricated ASIC chip.

REFERENCES

[1] Semiconductor Engineering. 2016. The Benefits of Antifuse OTP. Retrieved May 28, 2020 from http://semiengineering.com/the-benefits-of-antifuse-otp/.

[2] Jim Aarestad, Dhruva Acharyya, Reza Rad, and Jim Plusquellic. 2010. Detecting Trojans through leakage current analysis using multiple supply pad IDDQs. *IEEE Transactions on Information Forensics and Security* 5, 4 (2010), 893–904.

[3] Dakshi Agrawal, Selcuk Baktir, Deniz Karakoyunlu, Pankaj Rohatgi, and Berk Sunar. 2007. Trojan detection using IC fingerprinting. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy (SP'07)*. IEEE, Los Alamitos, CA, 296–310.

[4] M. M. Alam, S. Tajik, F. Ganji, M. Tehranipoor, and D. Forte. 2019. RAM-Jam: Remote temperature and voltage fault attack on FPGAs using memory collisions. In *Proceedings of the 2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'19)*. 48–55.

[5] Davide Baderna, Alessandro Cabrini, Marco Pasotti, and Guido Torelli. 2006. Power efficiency evaluation in Dickson and voltage doubler charge pump topologies. *Microelectronics Journal* 37, 10 (2006), 1128–1135.

[6] Eric Balard, Alain Chateau, and Jerome Azema. 2009. Run-time firmware authentication. US Patent 7,539,868.

[7] Mainak Banga and Michael S. Hsiao. 2009. A novel sustained vector technique for the detection of hardware Trojans. In *Proceedings of the 2009 22nd International Conference on VLSI Design*. IEEE, Los Alamitos, CA, 327–332.

[8] Mark Beaumont, Bradley Hopkins, and Tristan Newby. 2011. *Hardware Trojans—Prevention, Detection, Countermeasures (a Literature Review)*. Technical Report. Defence Science and Technology Organization Edinburgh (Australia) Command.

[9] Georg T. Becker, Francesco Regazzoni, Christof Paar, and Wayne P. Burleson. 2013. Stealthy dopant-level hardware Trojans. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems*. 197–214.

[10] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, Xuan Thuy Ngo, and Laurent Sauvage. 2013. Hardware Trojan horses in cryptographic IP cores. In *Proceedings of the 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'13)*. IEEE, Los Alamitos, CA, 15–29.

[11] Swarup Bhunia, Michael S. Hsiao, Mainak Banga, and Seetharam Narasimhan. 2014. Hardware Trojan attacks: Threat analysis and countermeasures. *Proceedings of the IEEE* 102, 8 (2014), 1229–1247.

[12] Rajat Subhra Chakraborty, Seetharam Narasimhan, and Swarup Bhunia. 2009. Hardware Trojan: Threats and emerging solutions. In *Proceedings of the IEEE International High Level Design Validation and Test Workshop (HLDVT'09)*. IEEE, Los Alamitos, CA, 166–171.

[13] Rajat Subhra Chakraborty, Francis Wolff, Somnath Paul, Christos Papachristou, and Swarup Bhunia. 2009. MERO: A statistical approach for hardware Trojan detection. In *Cryptographic Hardware and Embedded Systems—CHES 2009*. Springer, 396–410.

[14] Brian P. Deskin, William E. Hall, and David W. Pruden. 2010. Implementing enhanced security features in an ASIC using eFuses. US Patent 7,724,022.

[15] Farimah Farahmandi, Yuanwen Huang, and Prabhat Mishra. 2017. Trojan localization using symbolic algebra. In *Proceedings of the 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC'17)*. IEEE, Los Alamitos, CA, 591–597.

[16] P. Favrat, P. Deval, and M. J. Declercq. 1998. A high-efficiency CMOS voltage doubler. *IEEE Journal of Solid-State Circuits* 33, 3 (March 1998), 410–416.

[17] Jiaji He, Yiqiang Zhao, Xiaolong Guo, and Yier Jin. 2017. Hardware Trojan detection through chip-free electromagnetic side-channel statistical analysis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 10 (2017), 2939–2948.

[18] John L. Hennessy and David A. Patterson. 2011. *Computer Architecture: A Quantitative Approach*. Elsevier.

[19] Tamzidul Hoque, Seetharam Narasimhan, Xinmu Wang, Sanchita Mal-Sarkar, and Swarup Bhunia. 2017. Golden-free hardware Trojan detection with high sensitivity under process noise. *Journal of Electronic Testing* 33, 1 (2017), 107–124.

[20] Yumin Hou, Hu He, Kaveh Shamsi, Yier Jin, Dong Wu, and Huaqiang Wu. 2018. R2D2: Runtime reassurance and detection of A2 Trojan. In *Proceedings of the 2018 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST'18)*. IEEE, Los Alamitos, CA.

[21] Meikei Ieong, Vijay Narayanan, Dinkar Singh, Anna Topol, Victor Chan, and Zhibin Ren. 2006. Transistor scaling with novel materials. *Materials Today* 9, 6 (2006), 26–31.

[22] IC Insights. n.d. Home Page. Retrieved May 28, 2020 from http://www.icinsights.com/.

[23] Yier Jin and Yiorgos Makris. 2008. Hardware Trojan detection using path delay fingerprint. In *Proceedings of the 2008 IEEE International Workshop on Hardware-Oriented Security and Trust (HOST'08)*. IEEE, Los Alamitos, CA, 51–57.

[24] Petri Mikael Johansson and Per Ståhl. 2011. Secure end-of-life handling of electronic devices. US Patent 8,060,748.

[25] J. S. Rajesh, Koushik Chakraborty, and Sanghamitra Roy. 2018. Hardware Trojan attacks in SoC and NoC. In *The Hardware Trojan War: Attacks, Myths, and Defenses*, S. Bhunia and M. M. Tehranipoor (Eds.). Springer International Publishing, 55–74.

[26] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre attacks: Exploiting speculative execution. arXiv:1801.01203.

[27] C. Kothandaraman, Sundar K. Iyer, and Subramanian S. Iyer. 2002. Electrically programmable fuse (eFUSE) using electromigration in silicides. *IEEE Electron Device Letters* 23, 9 (2002), 523–525.

[28] Angela Krstic and Kwang-Ting Tim Cheng. 2012. *Delay Fault Testing for VLSI Circuits*. Vol. 14. Springer Science & Business Media.

[29] S. H. Kulkarni, Z. Chen, B. Srinivasan, B. Pedersen, U. Bhattacharya, and K. Zhang. 2015. Low-voltage metal-fuse technology featuring a 1.6 V-programmable 1T1R bit cell with an integrated 1 V charge pump in 22 nm tri-gate process. In *Proceedings of the 2015 Symposium on VLSI Technology (VLSI Technology'15)*. IEEE, Los Alamitos, CA, C174–C175.

[30] S. H. Kulkarni, Z. Chen, J. He, L. Jiang, M. B. Pedersen, and K. Zhang. 2010. A 4 kb metal-fuse OTP-ROM macro featuring a 2 V programmable 1.37 μ m^2 1T1R bit cell in 32 nm high-k metal-gate CMOS. *IEEE Journal of Solid-State Circuits* 45, 4 (2010), 863–868.

[31] MIT Lincoln Laboratory. n.d. Common Evaluation Platform. Retrieved May 28, 2020 from https://github.com/mit-ll/CEP.

[32] Kexin Li and Shaloo Rakheja. 2019. A unified static-dynamic analytic model for ultra-scaled III-nitride high electron mobility transistors. *Journal of Applied Physics* 125, 13 (2019), 134503.

[33] Lang Lin, Wayne Burleson, and Christof Paar. 2009. MOLES: Malicious off-chip leakage enabled by side-channels. In *Proceedings of the 2009 International Conference on Computer-Aided Design*. ACM, New York, NY, 117–122.

[34] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown. arXiv:1801.01207.

[35] Bongani Christopher Mabuza. 2012. *Charge Pumps and Floating Gate Devices for Switching Applications*. Ph.D. Dissertation. University of Pretoria.

[36] Amr M. Mohsen, Esmat Z. Hamdy, and John L. McCullum. 1989. Programmable low impedance anti-fuse element. US Patent 4,823,181.

[37] Adib Nahiyan, Mehdi Sadi, Rahul Vittal, Gustavo Contreras, Domenic Forte, and Mark Tehranipoor. 2017. Hardware Trojan detection through information flow security verification. In *Proceedings of the 2017 IEEE International Test Conference (ITC'17)*. IEEE, Los Alamitos, CA, 1–10.

[38] Adib Nahiyan and Mark Tehranipoor. 2017. Code coverage analysis for IP trust verification. In *Hardware IP Security and Trust*. Springer, 53–72.

[39] Adib Nahiyan, Kan Xiao, Kun Yang, Yier Jin, Domenic Forte, and Mark Tehranipoor. 2016. AVFSM: A framework for identifying and mitigating vulnerabilities in FSMs. In *Proceedings of the 2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC'16)*. IEEE, Los Alamitos, CA, 1–6.

[40] Seetharam Narasimhan, Xinmu Wang, Dongdong Du, Rajat Subhra Chakraborty, and Swarup Bhunia. 2011. TeSR: A robust temporal self-referencing approach for hardware Trojan detection. In *Proceedings of the 2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST'11)*. IEEE, Los Alamitos, CA, 71–74.

[41] Nahmsuk Oh, Peivand Fallah-Tehrani, and Alireza Kasnavi. 2011. Generation of engineering change order (ECO) constraints for use in selecting ECO repair techniques. US Patent 7,962,876.

[42] OpenCores.org. n.d. OpenRISC OR1200 Processor. Retrieved May 28, 2020 from http://opencores.org/or1k/OR1200/OpenRISC/Processor.

[43] Jungmin Park, Xiaolin Xu, Yier Jin, Domenic Forte, and Mark Tehranipoor. 2018. Power-based side-channel instruction-level disassembler. In *Proceedings of the 55th Annual Design Automation Conference*. ACM, New York, NY, 119.

[44] Jeyavijayan Rajendran, Efstratios Gavas, Jorge Jimenez, Vikram Padman, and Ramesh Karri. 2010. Towards a comprehensive and systematic classification of hardware Trojans. In *Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS'10)*. IEEE, Los Alamitos, CA, 1871–1874.

[45] Jeyavijayan Rajendran, Huan Zhang, Ozgur Sinanoglu, and Ramesh Karri. 2013. High-level synthesis for security and trust. In *Proceedings of the 2013 IEEE 19th InternationalOn-Line Testing Symposium (IOLTS'13)*. IEEE, Los Alamitos, CA, 232–233.

[46] N. Robson, J. Safran, C. Kothandaraman, A. Cestero, X. Chen, R. Rajeevakumar, A. Leslie, D. Moy, T. Kirihata, and S. Iyer. 2007. Electrically programmable fuse (eFUSE): From memory redundancy to autonomic chips. In *Proceedings of the 2007 IEEE Custom Integrated Circuits Conference*. 799–804.

[47] Masoud Rostami, Farinaz Koushanfar, and Ramesh Karri. 2014. A primer on hardware security: Models, methods, and metrics. *Proceedings of the IEEE* 102, 8 (2014), 1283–1295.

[48] Hassan Salmani and Mohammed Tehranipoor. 2013. Analyzing circuit vulnerability to hardware Trojan insertion at the behavioral level. In *Proceedings of the 2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'13)*. IEEE, Los Alamitos, CA, 190–195.

[49] Hassan Salmani, Mohammad Tehranipoor, and Ramesh Karri. 2013. On design vulnerability analysis and trust benchmarks development. In *Proceedings of the 2013 IEEE 31st International Conference on Computer Design (ICCD'13)*. IEEE, Los Alamitos, CA, 471–474.

[50] Bicky Shakya, Tony He, Hassan Salmani, Domenic Forte, Swarup Bhunia, and Mark Tehranipoor. 2017. Benchmarking of hardware Trojans and maliciously affected circuits. *Journal of Hardware and Systems Security* 1, 1 (2017), 85–102.

[51] Yuriy Shiyanovskii, F. Wolff, Aravind Rajendran, C. Papachristou, D. Weyer, and W. Clay. 2010. Process reliability based Trojans through NBTI and HCI effects. In *Proceedings of the 2010 NASA/ESA Conference on Adaptive Hardware and Systems (AHS'10)*. IEEE, Los Alamitos, CA, 215–222.

[52] Oliver Soll, Thomas Korak, Michael Muehlberghuber, and Michael Hutter. 2014. EM-based detection of hardware Trojans on FPGAs. In *Proceedings of the 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST'14)*. IEEE, Los Alamitos, CA, 84–87.

[53] J. A. Starzyk, Ying-Wei Jan, and Fengjing Qiu. 2001. A DC-DC charge pump design based on voltage doublers. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 48, 3 (March 2001), 350–359.

[54] Takeshi Sugawara, Daisuke Suzuki, Ryoichi Fujii, Shigeaki Tawa, Ryohei Hori, Mitsuru Shiozaki, and Takeshi Fujino. 2014. Reversing stealthy dopant-level circuits. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems*. 112–126.

[55] William R. Tonti. 2008. eFuse design and reliability. In *Integrated Reliability Workshop Final Report*. IEEE, Los Alamitos, CA,114.

[56] Adam Waksman, Matthew Suozzo, and Simha Sethumadhavan. 2013. FANCI: Identification of stealthy malicious logic using Boolean functional analysis. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*. ACM, New York, NY, 697–708.

[57] Jieh-Tsorng Wu and Kuen-Long Chang. 1998. MOS charge pumps for low-voltage operation. *IEEE Journal of Solid-State Circuits* 33, 4 (1998), 592–597.

[58] Tony F. Wu, Karthik Ganesan, Yunqing Alexander Hu, H.-S. Philip Wong, S. Simon Wong, and Subhasish Mitra. 2016. TPAD: Hardware Trojan prevention and detection for trusted integrated circuits.*IEEE Transactions on CAD of Integrated Circuits and Systems* 35, 4 (2016), 521–534.

[59] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor. 2016. Hardware Trojans: Lessons learned after one decade of research. *ACM Transactions on Design Automation of Electronic Systems* 22, 1 (May 2016), Article 6, 23 pages.

[60] Kan Xiao, Domenic Forte, Yier Jin, Ramesh Karri, Swarup Bhunia, and M. Tehranipoor. 2016. Hardware Trojans: Lessons learned after one decade of research. *ACM Transactions on Design Automation of Electronic Systems* 22, 1 (2016), 6.

[61] Kan Xiao and Mohammed Tehranipoor. 2013. BISA: Built-in self-authentication for preventing hardware Trojan insertion. In *Proceedings of the 2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST'13)*. IEEE, Los Alamitos, CA, 45–50.

[62] Xilinx. n.d. Virtex-7. Retrieved May 28, 2020 from https://www.xilinx.com/products/boards-and-kits/dk-v7-vc709-g.html.

[63] Kaiyuan Yang, Matthew Hicks, Qing Dong, Todd Austin, and Dennis Sylvester. 2016. A2: Analog malicious hardware. In *Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP'16)*. IEEE, Los Alamitos, CA, 18–37.

Received August 2019; revised January 2020; accepted April 2020