Experiences in Hardware Trojan Design and Implementation

Yier Jin*, Nathan Kupp*, and Yiorgos Makris†
*Department of Electrical Engineering, Yale University
†Departments of Electrical Engineering and Computer Science, Yale University

Abstract— **We report our experiences in designing and implementing several hardware Trojans within the framework of the Embedded System Challenge competition that was held as part of the Cyber Security Awareness Week (CSAW) at the Polytechnic Institute of New York University in October 2008. Due to the globalization of the Integrated Circuit (IC) manufacturing industry, hardware Trojans constitute an increasingly probable threat to both commercial and military applications. With traditional testing methods falling short in the quest of finding hardware Trojans, several specialized detection methods have surfaced. To facilitate research in this area, a better understanding of what Hardware Trojans would look like and what impact they would incur to an IC is required. To this end, we present eight distinct attack techniques employing Register Transfer Level (RTL) hardware Trojans to compromise the security of an *Alpha* encryption module implemented on a Digilent BASYS Spartan-3 FPGA board. Our work, which earned second place in the aforementioned competition, demonstrates that current RTL designs are, indeed, quite vulnerable to hardware Trojan attacks.**

I. Introduction

Due to global economic pressures, device fabrication foundries have spread around the world and the trend to move the IC supply chain from high-cost to low-cost locations is accelerating. Even once-trusted foundries are now vulnerable to attacks, and the threat that a foundry may be compromised and malicious circuits inserted in chips it fabricates is substantial. This motivates researchers to explore new testing methods, different from traditional functional and structural testing, because the characteristics of added malicious circuits (hardware Trojans) are different from previous manufacturing defects or functional errors. There are several reasons why standard testing methods are almost useless in detecting hardware Trojans:

1) Unanticipated behavior is not included in the fault list, i.e., structural pattern testing will likely not cover Trojan test vectors [1];
2) Additional functionality of genuine designs is hard to predict without knowledge of the Trojan inserted by attackers. Hence, routine functional testing is unlikely to reveal harmful extra functions;
3) Exhaustive input patterns testing is impractical as chips become more complicated with a large number of primary inputs and inner gates.

Based on these reasons, state-of-the-art EDA tools contribute little to the task of hardware Trojan detection. Only destructive reverse-engineering is still effective in checking the integrity and genuineness of manufactured chips, but with high testing cost. Furthermore, this method, as the name indicates, can only be used on a sample group of chips with no guarantee provided that untested chips are Trojan-free [2].

In order to demonstrate the threat of hardware Trojans and provide an extensive view on what Trojans will look like, The Polytechnic Institute of NYU hosted an Embedded Systems Challenge competition in October of 2008. In this competition, a hypothetic "Orange Army" developed a cryptographic device code-named *Alpha*, which was described at the RTL in a hardware description language (HDL). The device uses the strong 128-bit private key block cipher AES [3], which has been shown to be resistant to modern cryptanalysis techniques. Figure 1 shows the architecture of the *Alpha* design which, for the purpose of the competition, was implemented on a Digilent BASYS Spartan-3 FPGA board. To transmit a message, operators must do the following:

1) Select the private key using the "Key Select" slide switches.
2) Press the "INI System" button.
3) Input plaintext using a keyboard connected to the "KB IN" port. A VGA monitor can be connected to the "VGA OUT" port to display the plaintext. The "Input Status" LEDs indicate how much of the input buffer is used.
4) Press the "Start Encryption" button to encrypt. When this key is pressed the encryption engine reads the plaintext from the input buffer and writes the ciphertext into the output buffer.
5) Press the "Transmit" button to send the contents of the buffer out of the RS-232 port.

Competitors were asked to design and implement a set of Trojans, to undermine Alpha's cryptographic strength, and incorporate them into Alpha's HDL without failing validation testing [4]. In this paper, we described our experiences in designing and implementing such hardware Trojans for the purpose of this competition. Our team developed the highest number of alternative Trojans that cannot be detected through a normal testing stage and was the only one to exploit vulnerabilities across the entire datapath, eventually earning second place in the competition.

The rest of the paper is organized as follows: Section 2 lists recently proposed Trojan detection methods and argues for the importance of designing practical Trojans when developing Trojan detection approaches. Section 3 presents the various

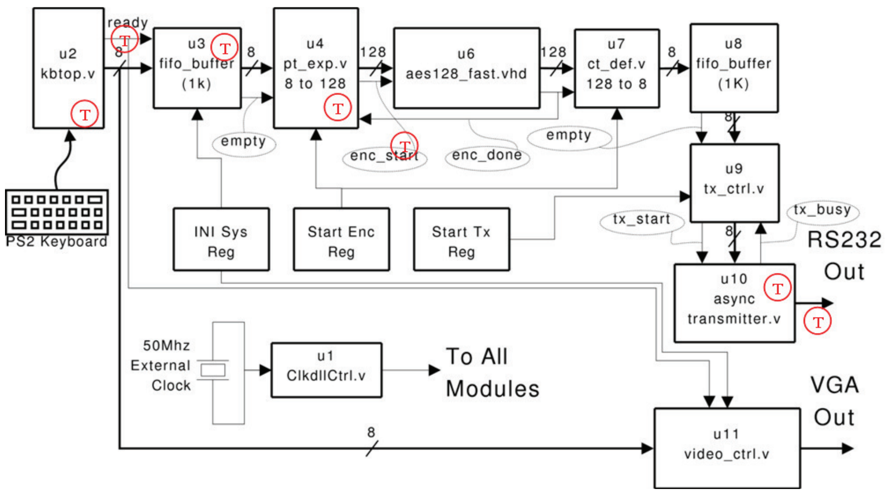

Fig. 1. Alpha architecture

categories of hardware Trojans, which guide us in the design of practical attacks for the purpose of the competition. Section 4 details all the Trojans that we designed in the target circuit, and conclusions are drawn in Section 5.

II. Previous Detection Approaches

In order to overcome the shortages of traditional testing methods in Trojan detection, new low-cost testing schemes are of high priority to secure the whole design chain when the fabrication foundry is untrusted. Several Trojan detection schemes have already been proposed, among which two main techniques are functional testing and side-channel fingerprint generation. In [1], the author argues that attackers would only choose rarely occurring events as triggers and proposes equations to compute low frequency events as a complement to input patterns generated by commercial ATPG tools. The assumption here is quite weak, since as soon as attackers know the testing scheme, they will surely do the same computation and choose more frequently occurring patterns as triggers. [2] is the first paper to present the idea of differentiating Trojan-inserted chips by comparing the side-channel fingerprints of tested chips with those generated from gold models. It analyzed the common behavior of various types of Trojans and demonstrated the feasibility of building effective fingerprints for an IC family to detect Trojan-inserted ICs. Noise modeling was used to construct the fingerprint for an IC family and Karhunen-Loeve (KL) expansion was a computational tool to separate the randomness and the time-variation of a random process. This method is useful in detecting Trojans when the Trojan circuit is large enough compared to the whole chip area and the process variation is low. The false alarm rate will emerge and increase quickly if the Trojan only occupies a trivial percentage of the whole chip area and there is large process variation.

In [5], a test strategy to detect anomalies introduced by the Trojan in the power ports currents was proposed. This method relies on numerous power ports to divide the whole circuit into smaller power grids. [6] expanded this approach by presenting four different signal calibration techniques used to reduce the adverse effects of process and test environment variations. Though this method is proven quite effective, it requires many power ports to divide the whole chip into small power grids.

In [7], path delay fingerprints were generated to differentiate nominal chips from chips with Trojan insertion. It meets an obstacle when the tested chip becomes more complicated, as the delay paths and test patterns increase exponentially. It is not easy to implement a similar method in high-end processors with a deep-pipeline structure and large memory to store data and instructions.

The above discussion summarizes previously proposed Trojan detection methods and their limitations. One shortcoming common to all of these papers is that the authors did not implement practical hardware Trojans which would escape traditional detection but would still be able to destroy the chip or leak sensitive information. Rather, as their target Trojans, they only considered addition of hardware structures, randomly located in the chip, which are not really capable of exhibiting a particular malicious behavior. In other words, these added hardware structures do not resemble what practical Trojans would look like. Moreover, the impact of these added hardware structures on the IC is strongly influenced by the method that is proposed for detecting them. For example, in [2], where global power traces are collected, the inserted Trojans include a 16-bit counter, an 8-bit sequential comparator and a 3-bit combinational comparator, which are large enough to consume the power necessary for the proposed method to detect them. Similarly, in [5], where local currents are measured, NAND gates are used to represent Trojans and are precisely located in place-holder areas where the actual Trojans are expected to reside and which are covered by the current measurements that are taken. Finally, in [7], a 2-bit comparator and a 4-bit counter are used to represent implicit payload and explicit payload Trojans, respectively. While these Trojans are more concrete, it has not been demonstrated that they can evade functional and/or structural testing, hence it is not obvious that they constitute practical Trojans.

In summary, the target Trojans discussed in these papers can only be regarded as a portion of a practical Trojan, since they only reflect the power or delay impact that a practical Trojan may incur. In order to further support the development of appropriate detection methods, the design and implementation of practical hardware Trojans needs to be considered. Given that the most straightforward mechanism for a malicious attacker is to modify the RTL design, in the rest of this paper we investigate various attacks on a design at the RTL level. Specifically, we examine the possibility of designing hardware Trojans that are able to evade state-of-the-art detection methodologies, as well as pass structural and functional test, yet they are able to incur concrete and actual malicious effects. Overall, eight different implementations of hardware Trojans are proposed and their operational mechanism and caused damage are detailed.

III. Hardware Trojans

In order to demonstrate the efficacy of hardware Trojans with a range of sophistication, we designed 8 types of hardware Trojans (titled tro1, tro2,..., tro8 in the rest of the paper) and implemented them in the *Alpha* architecture. All these Trojans were designed on the RTL level, that is, the *Alpha* HDL code has been modified. In the following sections, we examine different characteristics of hardware Trojans and discuss practical aspects regarding tradeoffs between complexity and overhead.

A. Payload

For a hardware Trojan to be useful, it must carry a *payload*. In other words, we attempt to link the activation of the Trojan with some deterministic event that is favorable to the attacker. There are several categories into which hardware Trojans can be classified according to these payloads:

1) Broadcast to the attacker some internal signals, which are often sensitive data (e.g. the encryption key).
2) Compromise the function of the circuits (e.g. to replace the plaintext with other preset information).
3) Destroy the chip.

Our designs, tro1, tro4, tro5, tro7,and tro8 belong to the first group. Clearly, to an attacker, the most valuable information in a cryptographic device is the encryption key. When the Trojan is triggered, the key will be transmitted along with the cipher text. The attacker, by appropriately listening to the transmission channel, can thus acquire the key and break the system.

Tro3 belongs to the second group, the replacement of plaintext. With this Trojan being activated, the legitimate receiver will still receive reasonable plaintext. For example, whenever we input “Moscow”, it will be changed to “Boston”, i.e., the plaintext “launch the missile targeting Moscow” will be changed to “launch the missile targeting Boston”.

Tro2, tro6 belong to the third group, and are designed to destroy the whole chip. This approach has the advantage of being very difficult to be detected during functional test, as the configured sleep time of the Trojan will often be much longer than the testing time. So in the testing stage, Trojans are inactivated without causing any harm to the chips.

B. Triggers

Similarly, we can classify the Trojans according to their *trigger*, an event which enables the Trojan. This event is designed to evade functional test by either relying on rare/atypical events, such as undefined input sequences, or by using a counter after a long period. For triggers, we have the following categories:

1) The attacker can physically access the device and can give special input to trigger the Trojan directly.
2) The Trojan is triggered internally: by a specific input event, counter, or other signal change.
3) No trigger; the trojan is always activated.

Tro1, tro2, and tro8 belong to the first group in which the trigger methods can be quite obscure and hard to detect during normal testing. The most compact but useful one is to redefine an unused key in the keyboard to trigger the Trojan. Since only the alphanumeric characters are officially supported by *Alpha*, this trigger can reasonably be expected to escape functional test. An alternative is to use a specially designed input string. Using a special string as the trigger should easily escape functional testing since the input space is too large and it is unlike that this special string will be typed accidentally. Clearly, however, a trigger that relies on physical access is limited in application.

The second class of triggers (tro3, tro4, tro5, tro6, tro7) is for devices which cannot be physically accessed by an attacker. In this case, we note that some type of internal counter can be used as an activation mechanism for the Trojan. For example, we can implement a counter to track the number of transmission times in *Alpha*, and trigger after it exceeds a pre-specified number. A real-time counter can also be used as a trigger. Finally, the switch of internal signals can also be used as the trigger, such as in tro5. In this design, whenever the key has been changed, the Trojan will be activated. An even more sophisticated method inserts a RS232 receiver in order to let the attacker control the whole chip through RS232 channel.

The third group is more aggressive and configures the Trojan so that it is always activated. It requires the payload of the Trojan to be well hidden, i.e., by broadcasting secret information in ways undetectable by functional test. A shortcoming of configuring a Trojan this way is that the power usage will be measurably higher, potentially raising suspicions during test. Tro5 can be easily modified to fit in this group, although in the interest of space, we do not discuss it herein.

C. Code Optimization

Along with the trigger/payload-based Trojans described above, there are two Trojan targets which can be quite effective and hard to detect at the RTL[1].

[1]Note that these two kinds of Trojans platforms could be also be implemented at the gate level or the transistor level.

The first is targeting unoptimized HDL code. For example, some modules may be written in an unoptimized way and even with the help of advanced synthesis tools, the generated gate-level design will have significant amounts of redundant logic. The attacker, who is already aware of the specification of this module can rewrite the HDL code in a significantly more compact way while performing the same functionality. The on-chip resources saved by this optimization can then be allocated to a Trojan. For example, in our work we rewrite the serial-to-parallel data conversion module `pt_exp.v` which saves us 128 Flip-flops. In tro2, tro7 and tro8, the original `pt_exp.v` file is replaced by our own designed module to balance the total on-chip area usage and power consumption.

The second is targeting standard or commercial modules (IP cores). IP cores are now widely used both in academic and commercial designs to minimize developing time. As a reusable module, IP cores are typically designed to complete some generic function in order to suit many different usages. However, in most applications, not all of these functions will be used. This leads to redundant logic which can be optimized away by modifying the code to eliminate unused functions. Again, the chip area saved by this modification can be used to insert Trojans. As we will show later, the `Ps2interface.vhd` module is optimized to eliminate 13 Flip-flops and 10 4-input LUTs.

In tro2, and tro3, the original `Ps2interface.vhd` file is replaced by our optimized module to reduce the total on-chip area usage and power consumption.

IV. Trojan Implementations

A. Experimental Setup

The implementations of our Trojans on the *Alpha* platform are performed within the Xilinx ISE Webpack 10.1 environment, using the target chip XC3S250e-4tq144 on a Digilent Basys development board [8]. These Trojans are located throughout the whole *Alpha* architecture as shown in Figure 1 where "Ⓣ" means the location of each Trojan.

Figure 2 summarizes all eight Trojans discussed in the rest of the paper including their triggers, payloads, area overheads and the likelihood they might be detected through normal testing methods.

B. Trojan type I

Description and trigger mechanism: This Trojan is inserted at the front-end of the design and monitors the keyboard input. If the phrase "New Haven" is detected at the beginning of the plaintext, the Trojan will be triggered. Whenever the Trojan is triggered, the first block (128 bits) of the output ciphertext will be replaced by the encryption key. The majority of the Trojan code is located in `alphatop.v` module where a FSM is inserted. Figure 3 shows the working procedure of Trojan type I.

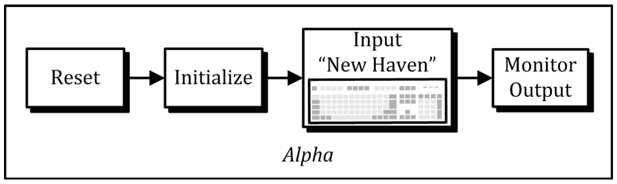

Fig. 3. Trojan type I architecture

Platform and area consumption: The original code without any modification was used as the platform for this Trojan. Altogether, 1486 Flip-flops and 4320 LUTs are used which correspond to +0.8% and +6.8% more usage than original design, respectively.

Implementation efficacy: The attacker who has access to the input (keyboard) and is listening to the output channel (RS232 serial port) can easily acquire the encryption key. A remote attacker who cannot access the device physically but can listen to the communication channel can still acquire the key, but is reliant on a user entering the special text as plaintext.

Limitations: As mentioned above, a special input string is required to trigger the Trojan. Therefore, either the attacker must have physical access to the system, or depend on the legitimate user entering this special string. Of course, the attacker must also have the ability to monitor the output. Moreover, the chip area overhead will probably lead to more power consumption, raising it to suspicious levels during test. Although it is very unlikely, there is a non-zero probability that the Trojan will be discovered during functional test, if the special keyword is used to test the system. Furthermore, when the Trojan is triggered, the legitimate receiver will still attempt to decrypt the ciphertext which has been replaced by the encryption key. This will result in the receiver acquiring garbage data instead of the correct plaintext, which could also lead to the detection of the Trojan.

C. Trojan type II

Description and trigger mechanism: This Trojan was designed to reduce power usage significantly to avoid suspicious increases in power usage due to a Trojan. Two Verilog files were edited to remove extraneous components and reduce space usage before the Trojan was implemented. For this Trojan, the trigger is an originally undefined key "F12". Whenever, the key is pressed, the triggered Trojan will lock and ignore any input unless the FPGA is reprogrammed. The primary Trojan code is located in the `kb2ascii.v` file where the "F12" key is defined. Figure 4 shows the working procedure of Trojan type II.

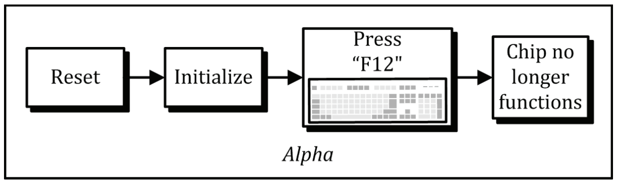

Fig. 4. Trojan type II architecture

Platform and area consumption: The code with optimized `pt_exp.v` and `kb_top.v` modules was used as a platform.

Trojan Summary Table

Type	Trigger: By Whom	Trigger: How	Payload: How	Payload: Leaking Channel	Changed Mod.
Tro1	Attacker with access to input device	Input "New Haven"	First block of ciphertext replaced by key	RS-232 TxD	N/C
Tro2	Attacker/legitimate user	Press "F12" key	The chip stops working	--	*pt_exp, kb_top*
Tro3	Legitimate user	Input "Moscow"	"Moscow" is replaced by "Boston" at output	RS-232 TxD	*kb_top*
Tro4	Legitimate user	Input > 1KB data	Last block of ciphertext replaced by key	RS-232 TxD	N/C
Tro5	Legitimate user	When key is changed	New key is hidden in the output	RS-232 TxD	N/C
Tro6	Legitimate user	Transmit > N	The chip stops working	--	N/C
Tro7	Attacker (using the RxD port)	Control w/ RxD port	Chip controlled by the attackers	RS-232 TxD	*pt_exp*
Tro8	Attacker with access to input device	Press "Caps Lock" key	The "Caps Lock" LED will reveal the key	Keyboard	*pt_exp*

Type	Area Overhead: Flop-flops	Area Overhead: 4-input LUTs	Testing Detection Likelihood: Functional	Testing Detection Likelihood: Power
Tro1	1486(+0.8%)	4320(+6.8%)	Unlikely[1]	Likely
Tro2	1336(-9.4%)	4198(+0.024%)	Unlikely	Unlikely
Tro3	1523(+3.3%)	4266(+2.4%)	Unlikely[1]	Likely
Tro4	1475(+0.068%)	4273(+1.8%)	Unlikely[2]	Likely
Tro5	1485(+0.75%)	4255(+1.4%)	Nearly impossible	Unlikely[3]
Tro6	1479(+0.34%)	4204(+0.17%)	Nearly impossible	Unlikely
Tro7	1409(-4.4%)	4401(+4.9%)	Nearly impossible	Unlikely
Tro8	1396(-5.3%)	4305(+2.6%)	Nearly impossible	Unlikely

[1]Except with exhaustive input patterns
[2]Except with buffer-overflow test
[3]Except with transmit power testing

Fig. 2. Trojan Summary Table

Altogether, 1336 Flip-flops and 4198 LUTs are used which are 9.4% less than the original design and 0.024% more than original design, respectively. Clearly, using too few Flip-flops is not problematic, as we can insert dummy FFs to match the original design at will.

Implementation efficacy: The attacker who can access the input can easily trigger the Trojan. A remote attacker who cannot access the device physically is reliant on a user mistakenly pressing the undefined key to trigger the Trojan.

Limitations: Due to the limitations of the synthesis tools, we cannot destroy the chip physically through RTL code, i.e., by making a connection from V_{DD} to ground. As the Trojan can be deactivated by simply reprogramming the device, this type of Trojan would likely not incur serious long-term negative effects for the user.

D. Trojan type III

Description and trigger mechanism: This Trojan is designed for the special case where the attacker knows the usage of the circuit and can construct a plaintext-replacement scheme appropriately. In this Trojan, whenever the word "Moscow" is detected in the plaintext, it will be replaced by "Boston". The phrase of interest and the replacement phrase can be changed according to the usage of the Trojan and the chip. The Trojan code is located in the `alphatop.v` module where a FSM is inserted. Figure 5 shows the working procedure of Trojan type III.

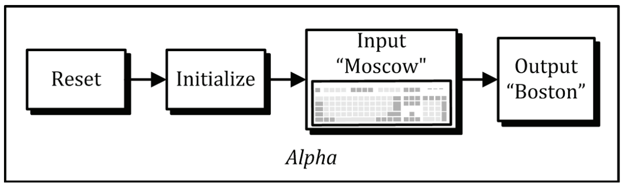

Fig. 5. Trojan type III architecture

Platform and area consumption: The code with optimized `kb_top.v` module was used a platform. Altogether, 1523 Flip-flops and 4266 LUTs are used which are +3.3% and +1.6% more than the original design.

Implementation efficacy: The attacker does not need to access the input equipment nor does he need to monitor the output. The selection of words to replace is critical in the ability of the Trojan to modify the function of the system in some way useful to the attacker.

Constraints: As mentioned above, the selection of keywords is critical when designing the Trojan. Again, however, this type of Trojan would succumb to exhaustive testing of the input space during functional test. Furthermore, if TMR (Triple Modular Redundancy) is used with three different encryption modules (the others without the hardware Trojan) to encrypt and transfer data, the modified result will be discarded and the effect of the Trojan will be nullified.

E. Trojan type IV

Description and trigger mechanism: The Trojan is custom-designed targeting the mismatched buffer sizes in the system. The plaintext buffer size is only 1 KB, while the video buffer is 4 KB. This makes it difficult for the operator to know whether he or she has overflowed the input buffer with too much plaintext. When the overflow of input buffer occurs, the Trojan is triggered and the payload is to replace the last block with the encryption key. This approach takes advantage of a detail in the original design: according to the design, if the input is 1KB, the last byte will be discarded in order to transmit the key index. So a legitimate receiver will discard the last block as garbage. Since the receiver will discard this block, the Trojan can simply replace the block with the entire encryption key. The Trojan code is located in the `alphatop.v` module. Figure 6 shows the working procedure of Trojan type IV.

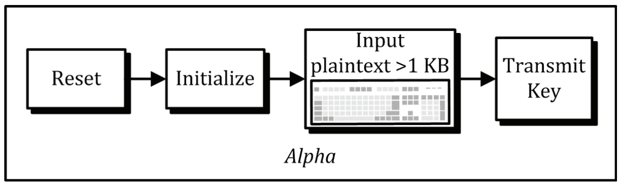

Fig. 6. Trojan type IV architecture

Platform and area consumption: The original code without any modification was used as a platform. Altogether, 1475 Flip-flops and 4273 LUTs are used which are +0.068% and +1.8% more than original design.
Implementation efficacy: The attacker need not physically access the device, but should have access to the communication channel to listen for the encryption key after a buffer overflow has occurred.
Constraints: The attacker must continuously monitor the communication channel, and check the last block of each transmission burst. If a buffer-overflow test is included in the functional test plan, it may detect this type of Trojan.

F. Trojan type V

Description and trigger mechanism: This Trojan is designed to compromise the communication channel by hiding information in transmissions over the RS-232 channel. In our design, the baud rate of the serial port is changed from 9600 bps to 19200 bps. In the original design, each RS-232 packet contains 1 start bit, 8 data bits, and 2 stop bits. At the 19200 bps baud rate, these parameters were changed to 1 start bit, 8 data bits and 1 stop bit. This enables the Trojan to interleave other information, i.e., the encryption key, along with the legitimate data. For each legitimate 8-bit packet, as many as 7 bits can be inserted. Figure 7 compares the original RS-232 package at 9600 Baud rate with the compromised channel of 19200 Baud rate packages. The "Tro" indicates where leaking bits can be inserted to reveal the secret. In our design, we use four of these bits. When the key index is changed, the Trojan is triggered to insert the encryption key into the hidden channel. The legitimate user or functional test equipment will not observe any difference in the output if the baud rate on the receiver is set at 9600 bps. Thus, the only possible means for detecting this Trojan is through comparing power consumption. The Trojan code is located in `async_transmitter.v` module. Figure 8 shows the working procedure of Trojan type V.

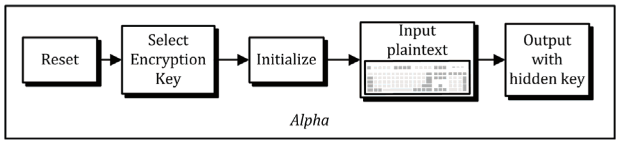

Fig. 8. Trojan type V architecture

Platform and area consumption: The original code without any modification was used as a platform. Altogether, 1485 Flip-flops (0.75% more than original design) and 4255 LUTs (+1.4% more than original design) are used.
Implementation efficacy: This is a sophisticated Trojan which is immune to functional testing within specification parameters, despite being triggered frequently. The attacks only need to monitor the RS-232 transmission channel at the 19200 Baud rate to acquire both the key and the ciphertext.
Constraints: Since the area overhead is insignificant, the Trojan cannot be detected by functional testing or by power trace testing for the majority of the time. However, when the Trojan is triggered, the power consumption at transmission stage can be higher than normal due to high Baud rate. Therefore, an on-line power profile monitoring method could potentially detect the Trojan.

G. Trojan type VI

Description and trigger mechanism: This Trojan is similar to a time-bomb. An inserted counter will increment for each character transmitted until it exceed a predefined number, N. When the Trojan is triggered, the output is locked to a binary 1. The Trojan code is located in `alphatop.v` module. Figure 9 shows the working procedure of Trojan type VI.

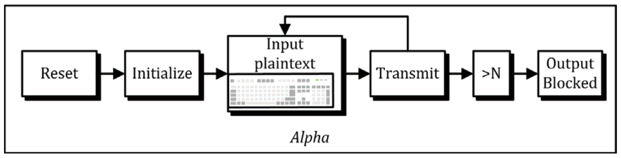

Fig. 9. Trojan type VI architecture

Platform and area consumption: The original code without any modification was used as a platform. Altogether, 1479 Flip-flops and 4204 LUTs are used which are +0.34% and +0.17% more than original design.
Implementation efficacy: The Trojan is solely a time-bomb which will not be activated until the count of transmissions

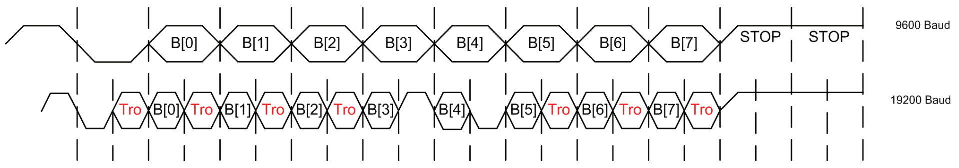

Fig. 7. Compromised RS232 channel

exceeds a predefined number. The attack should carefully set the number N.

Constraints: For this Trojan, the setting of N is critical to the whole design. A poor choice could result in detection during functional test. Again, due to the limitation of synthesis tools, we cannot destroy the chip physically through RTL code, which means that when the Trojan is triggered, reprogramming is an easy way to repair the chip. However, in a real environment, the resulting payload could result in destroying the chip, for example by activating a path that shorts V_{DD} to ground.

H. Trojan type VII

Description and trigger mechanism: This Trojan is an aggressive attempt to demonstrate how to fully control a device through a Trojan. In this design, the originally unused RxD port of Basys Board is configured as the trigger/control signal and one `async_receiver.v` module is added to the design. Along with a Trojan trigger signal, two more control signals are inserted in the original design: `tro7_trigger_rst`, and `tro7_trigger_tx`. These two signals are controlled through RxD port and act as remote versions of the Reset and Transmit buttons on the Basys Board. A new encryption key is also inserted, which is used to encrypt the information the Trojan transmits over the RS-232 communications channel. For example, in our implementation, the original encryption key is encrypted with this key, and then broadcast across the communication channel, enabling the attacker to acquire the encryption key. At the same time, the legitimate user is left unaware that the system has been compromised by the broadcast of the key, as the encrypted key looks like a garbage block. Table I displays a tabulation of the commands our implementation handles over the RS-232 RxD port. The Trojan code is located in `alphatop.v` and the extra `async_receiver.v` module.

Commands to RS-232 RxD port	Effect
`Trojaneftri`	Reset the system
`Trojanabtri`	Encrypt the encryption key with the Trojan key
`Trojancdtri`	Transmit encrypted key on RS-232 TxD port

TABLE I
TROJAN COMMAND KEYS

Platform and area consumption: The code with an optimized `pt_exp.v` module was used as a platform. Altogether, 1409 Flip-flops and 4401 LUTs are used which are 4.4% less than original design and 4.9% more than original design, respectively.

Implementation efficacy: Since the RxD channel in the original design is idle, the attacker must control this channel and monitor the TxD channel. And the `tro7_trigger_rst` signal can remove any traces the attackers left to make it even harder to detect the Trojan in daily use.

Constraints: To control the chip and trigger the Trojan, the attacker should acquire access to the RxD port of the device, which is more difficult than simply monitoring the output. Of course, if the physical connection to the RxD port does not exist, this Trojan will never be activated.

I. Trojan type VIII

Description and trigger mechanism: This Trojan was designed for the occasion when the attacker does not have access to the communications channel, but does have access to the input device—the keyboard. The technique we demonstrate here is one of many ways of extracting information from the device without using the RS-232 communications channel. For this Trojan, the trigger is the “Caps Lock” key, or any other undefined key on the keyboard. After a system reset, when the “Caps Lock” key is pressed, the Caps Lock LED will be on/off to indicate a ’1’ or ’0’ as the least-significant bit of the encryption key. The attacker need only to press the “Caps Lock” 128 times to acquire the entire key, progressing to the most-significant bit. The Trojan code in located in `kbtop.v` and `kb2ascii.v` modules. Figure 10 shows the working procedure of Trojan type VIII.

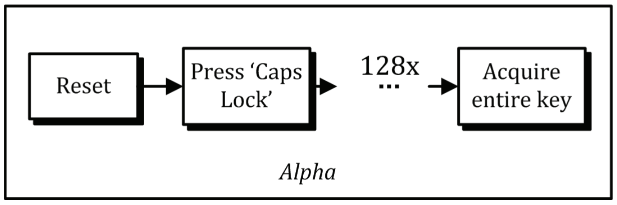

Fig. 10. Trojan type VIII architecture

Platform and area consumption: The code with an optimized `pt_exp.v` module was used as a platform. Altogether, 1396 Flip-flops and 4305 LUTs are used which are 5.3% less than original design and 2.6% more than original design, respectively.

Implementation efficacy: This Trojan removes the need for attackers to monitor the communication channel during an attack. The attacker need only monitor the keyboard LED to determine the encryption key and thus break the system.

Constraints: As mentioned above, the attackers should acquire access to the input device, which may not be feasible in many situations.

V. Conclusions

In this paper, we have explored eight different implementations of hardware Trojans. Each of these have different combinations of triggers, payloads, as well as unique sections of the architecture that each Trojan attacks. Most importantly, the Trojans explored herein are designed with varying levels of sophistication, allowing the attacker to tradeoff design time, ability to evade detection, and payload.

Our work presented the view that current RTL designs are indeed quite vulnerable to hardware Trojan attacks. There are clearly many points of vulnerability that can be exploited for malicious purposes. The designed Trojans also demonstrated that traditional functional testing can often be useless in detecting and preventing hardware Trojan attacks. This provides significant impetus to improve methods for detecting and combating hardware Trojans.

References

[1] F. Wolff, C. Papachristou, S. Bhunia, and R. S. Chakraborty, "Towards trojan-free trusted ics: Problem analysis and detection scheme," in *Design, Automation and Test in Europe*, 2008, pp. 1362–1365.

[2] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan detection using ic fingerprinting," in *Security and Privacy, IEEE Symposium on*, 2007, pp. 296–310.

[3] *Advanced Encryption Standard (AES) FIPS Pub 197 (2001, Nov.).*, [Online]. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[4] *http://isis.poly.edu/csaw/embedded*.

[5] R. Rad, J. Plusquellic, and M. Tehranipoor, "Sensitivity analysis to hardware trojans using power supply transient signals," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, 2008, pp. 3–7.

[6] R. M. Rad, X. Wang, M. Tehranipoor, and J. Plusquellic, "Power supply signal calibration techniques for improving detection resolution to hardware trojans," in *Computer-Aided Design, IEEE/ACM International Conference on*, 2008, pp. 632–639.

[7] Y. Jin and Y. Makris, "Hardware trojan detection using path delay fingerprint," in *Hardware-Oriented Security and Trust, IEEE International Workshop on*, 2008, pp. 51–57.

[8] *http://www.digilentinc.com*.