Why You Should Care About Don't Cares: Exploiting Internal Don't Care Conditions for Hardware Trojans

Wei Hu[†], Lu Zhang[‡], Armaiti Ardeshiricham[†], Jeremy Blackstone[†],
Bochuan Hou[‡], Yu Tai[‡] and Ryan Kastner[†]
[†]*University of California, San Diego, La Jolla, CA 92093*
[‡]*Northwestern Polytechnical University, Xi'an 710072, Shaanxi, China*
{weh040, aardeshi, jblackst, kastner}@ucsd.edu; {willvsnick, houbochuan, taiyu}@mail.nwpu.edu.cn

***Abstract*—Hardware Trojans are a significant security threat due to the globalization of hardware design and supply chain. We demonstrate a new type of hardware Trojan hidden behind internal don't care conditions. The proposed Trojans can pass through formal equivalence checking; they may reside after logic synthesis optimizations; and they are resilient to switching probability and side channel analysis. The new Trojans can create a surface for fault attack to retrieve secret information or downgrade performance by increasing power consumption. Experimental results show that these Trojans may stay after logic synthesis and that secret information can be retrieved using fault attack. We present detectability analysis and suggest synthesis optimizations as well as countermeasures that can help mitigate this new Trojan.**

***Index Terms*—Hardware security and trust, hardware Trojan, don't care, fault attack, countermeasure**

I. Introduction

The design and fabrication of modern hardware typically involve multiple teams spread around the world. This process often requires integrating disparate intellectual property (IP) cores developed by an untrusted third party. These IP components may be built with unspecified malicious functionality in addition to its specification. Additionally, an untrusted manufacture may also make undocumented changes to introduce desired functionality. Such malicious design modifications, identified as hardware Trojans, represent a major type of threat to hardware security and trust. They may provide a hidden channel to leak sensitive information or a back door for attackers to compromise a system. A number of reports show that hardware Trojans may even reside in critical devices responsible for protecting personal privacy [1], controlling high-assurance systems [2], and operating military weapons [3].

There are numerous examples demonstrating the possibility of inserting small and hard-to-detect Trojans into hardware designs, which is an important motivation for the development of more effective hardware Trojan detection methods. Tehranipoor et al. [4] and Karri et al. [5] were among the first to propose a comprehensive and complete taxonomy for a better understanding of potential threats caused by hardware Trojans. *Trust-HUB.org* further complemented these hardware Trojan classification systems and released a full set of Trojan benchmarks for evaluating the effectiveness of hardware Trojan detection methods. These benchmarks typically use rare events, e.g., a timer or a specific input pattern, to generate the Trojan trigger [6]. These simple trigger mechanisms are very sensitive to switching probability analysis [7]–[9]. More recent Trojan benchmarks use multiple discrete trigger signals to multiplex different portions of the design to the output [10], [11]. This will increase the switching probability of each trigger signal and better protect the Trojan from being detected. However, these hardware Trojan designs still assume explicit trigger signals. Hardware Trojan detection techniques using tagging or don't care analysis will identify the Trojan design as malicious or redundant [12], [13].

To prevent the Trojan design from being identified, researchers have leveraged unspecified functionality (i.e., external don't cares) for hardware Trojan insertion. Such external don't cares allow Trojan designers to exploit the unassigned states in the design space to implement additional functionality [14]–[17]. The resulting hardware design will be a super set of its original specification. Thus, formal verification will indicate that all specified functionalities are correctly implemented. Although this provides a promising approach for hardware Trojan insertion, it does not apply to completely specified functions, where all external don't care conditions are assigned to deterministic values.

We propose a new method that exploits internal don't care conditions for hardware Trojans. We search for internal don't cares originating from reconvergent fanouts to generate the Trojan trigger. Thus, our technique does not require explicit specification of don't cares, and will work even if the function is completely specified (unlike previous techniques using external don't cares). Our method inserts the Trojan in such a way that the designs with and without the Trojan are logically equivalent (unlike previous techniques implementing a super set). Consequently, formal equivalence checking tools will not find the inserted Trojan. We leverage the difference in path delay and use fault attack to force the design into an unreachable condition, which will allow the Trojan to activate and leak information. We also take advantage of the flexibility resulting from don't care condition to introduce redundant logic in order to downgrade performance.

In this paper, we exploit internal don't care conditions for a new type of hardware Trojan. We show that the Trojan may stay after logic optimizations and demonstrate the possibility of using fault attack to activate the Trojan to leak secret information. Our work expands the hardware Trojan spectrum and motivates the need for new Trojan detection methods. Specifically, we make the following contributions:

- Proposing a new type of hardware Trojan triggered by internal don't care conditions;
- Providing a fault attack to activate the Trojan to leak secret information;
- Presenting experimental results to evaluate our hardware Trojan designs.

The reminder of the paper is organized as follows. In Section II, we briefly review the related work. Section III describes our threat model. Section IV discusses internal don't care conditions and the details of our Trojan design. Section V presents experimental results and detectability analysis. We conclude in Section VI.

II. Related Work

We identify related work that enables a better understanding of the potential security threats caused by hardware Trojans. We focus on Trojan design approaches and only briefly cover hardware Trojan detection methods when relevant.

Tehranipoor et al. [4] and Karri et al. [5] provided hardware Trojan taxonomies. Hardware Trojans were classified according to the insertion phase, abstraction level, trigger mechanism, payload effects, and location. Such classification systems allow both hardware designers and security researchers a comprehensive understanding of the common characteristics of hardware Trojans. *Trust-HUB.org* released a set of hardware Trojan benchmarks that cover the classification systems [6]. These benchmarks have played an important role motivating the development of various Trojan detection methods. However, these benchmarks typically use a counter or specific input pattern to generate a trigger signal. Since Trojans are usually triggered only under rare events in order to escape from being detected, the trigger signal will have an extremely low switching probability. As a result, the *Trust-HUB* benchmarks are very sensitive to switching probability analysis [7]–[9].

To defeat switching probability based Trojan detection methods, researchers proposed more intelligent methods for hardware Trojan design. They use multiple discrete trigger signals to multiplex different portions of the design to the output [10], [11]. Trojan benchmarks designed in this manner are more resilient to switching probability analysis. However, these benchmarks still assume that the Trojan payloads will only be activated under rare conditions. Consequently, tagging based switching probability analysis and don't care based equivalence checking will still identify the Trojan design as malicious or redundant [12], [13].

The most relevant work uses unspecified functionality for hardware Trojans. When possible, it is desirable to specify signals under certain conditions as don't cares for logic optimization. Such external don't cares provide the flexibility to allow implementing additional functionality for Trojan insertion. Fern et al. [15], [16] proposed an approach to search among don't cares specified as X-states to insert hardware Trojans. Carefully chosen X-states are assigned to specific values under the don't care condition to leak information. In [18], a technique was proposed to detect hardware Trojans modifying unspecified functionality. The observation is that under condition C, an unspecified signal x should not affect critical points of the design otherwise could be malicious. It formulates a SAT problem to check if x could affect any critical points under given condition. Other work focuses on unoccupied state encoding to add extra states into the state machine [14], [17]. Such extra states are never reached during normal operation. An attacker can use fault attack to force the state machine into the desired state to perform malicious activities. These types of Trojans can be hard to detect because functional verification will indicate that all specified functionalities are correctly implemented; the additional Trojan design is out of the specification and thus will not be covered in verification. However, these methods will not work if the design is completely specified.

In this work, we exploit internal don't care conditions for hardware Trojan design. Such don't cares originate from signal correlations caused by reconvergent fanout. Thus, they widely exist even in completely specified hardware designs. We show that the proposed hardware Trojan can be implemented by slightly altering the description style of the design, and they can create a channel to leak secret information through fault attack.

III. Threat Model

The proposed Trojan can be inserted during various phases of the hardware design cycle, e.g., by a rogue designer at design entry, compromised synthesis tools while implementation or an untrusted manufacture during fabrication. In this work, we primarily focus our analysis on IP cores provided by an untrusted third-party. These IP products may come in the forms of either soft cores (in RTL code or generic gate-level netlist) or hard cores (in layout format). We assume that the untrusted vendors may insert the hardware Trojans described in this paper into their IP products.

We assume that the IP consumers have access to the Trojan free design specifications of the delivered IP products. They can perform functional verification to check the IP cores against their specifications and run different hardware Trojan detection methods to identify possible malicious design modifications. We assume that the IP products (in case of soft cores) may be re-synthesized to target a specific technology library or device. In case of hardware cores, the Trojan design will directly go through implementation.

We assume that the attacker has knowledge about the implementation details of IP products, including the Trojan inserted. He can manipulate the inputs (including the clock signal) to the IP cores and probe their outputs. We also assume that the attacker's primary goal is to retrieve secret information such as the cryptographic key instead of simply causing deny-of-service.

Our threat model is similar to those established in previous work [14], [15], [17]. We do not rely on unspecified functionality and further exploit the potential security threat triggered by internal don't care conditions resulting from signal correlation.

IV. Hardware Trojans Based on Internal Don't Care Conditions

In this section, we provide details about the design and implementation of the proposed hardware Trojan. Our Trojan utilizes internal don't care as the trigger. This allows us to create a Trojan such that the design with Trojan is functionally equivalent to its Trojan free correspondence. That is, our Trojan is undetectable using formal equivalence checking. If carefully deployed, the synthesis tools will not identify our Trojan design as redundancy primarily due to the NP-completeness of exact logic minimization. In order to activate the Trojan, we need to use fault attack to satisfy the don't care condition. This will allow the trigger to function maliciously in order to leak secret information.

A. Don't Care Conditions

A don't care condition is a particular function input under which the output does not matter. They provide flexibility for the optimization of Boolean functions [19]. The possible sources of don't cares include external don't cares (EDCs), satisfiability don't cares (SDCs), and observability don't cares (ODCs).

EDCs typically originate from incomplete specification of Boolean functions, e.g., values assigned to 'X' for logic optimization. As an example, the RSA cipher requires the key to be always an odd number. Thus, an even key value (i.e., logical `0` for the least significant key bit) can introduce an EDC condition. The output of the function under an EDC condition, when unspecified, can be leveraged for hardware Trojans, e.g., multiplexing the key to the ciphertext when the lowest key bit is logical `0`.

SDCs represent a never reached input condition for an internal node, e.g., two inputs to the node can never be logical `1` at the same time; ODCs represent input conditions under which the output of an internal node can be either logical `0` or `1`. In other words, under an ODC condition, the output of an internal node does not have an influence on any primary output. Unlike EDC, SDC and ODC are caused by signal correlations resulting from reconvergent fanout. Figure 1 shows some simple examples of SDC and ODC.

In Fig. 1, gate *g7* has an SDC condition. The combination `11` will never be observed at the inputs of this OR gate. Gates *g4* and *g5* have

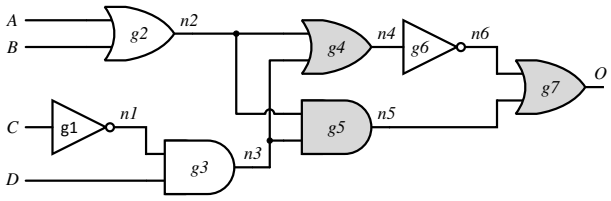


Fig. 1. Examples of SDC and ODC.

ODC conditions. Specifically, when $n2$ and $n3$ are both logical 0, the output from *g4* will dominate the primary output O. The output from *g5* does not have an effect on the output. Thus, 00 is an ODC condition for *g5*. Similarly, 11 is an ODC condition for *g4*.

The SDC and ODC signal pairs discussed above are typically connected to the same gate. We can further extend the don't care concept to signal pairs with a correlation but are topologically apart. As an example, consider signals *A* and *n6* in Fig. 1. Whenever *A* is logical 1, *n6* will be dominated to logical 0. Thus, it is impossible for them to be logical 1 simultaneously. This demonstrates a generalized don't care condition for our hardware Trojan.

Previous work has demonstrated the feasibility of employing EDC for hardware Trojan design [15]–[17]. However, EDCs can be fully eliminated if a Boolean function is completely specified. By comparison, the internal don't cares (SDC and ODC) are caused by signal correlation resulting from reconvergent fanout and thus widely exist even in completely specified Boolean functions. In the following subsection, we propose a new type of hardware Trojan that exploits internal don't care conditions.

B. Hardware Trojan Design and Implementation

The key idea behind our hardware Trojan is that don't care conditions provide the flexibility to pick the output value for certain internal nodes. With such flexibility, we can replace the normal output at those nodes with a desirable value (e.g., a key bit) under the don't care condition. Figure 2 shows two examples of such Trojans.

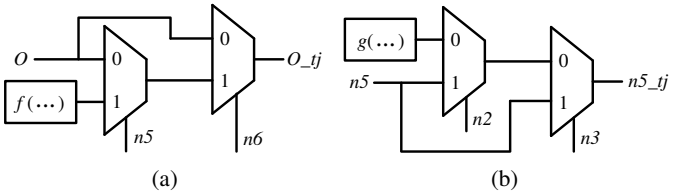


Fig. 2. Hardware Trojans based on internal don't cares. (a) A hardware Trojan based on SDC. (b) A hardware Trojan based on ODC.

In Fig. 2 (a), the Trojan design uses the SDC condition for gate *g7* in Fig. 1 to multiplex between the normal output O and some function f. Equation (1) shows the Boolean function for the Trojan.

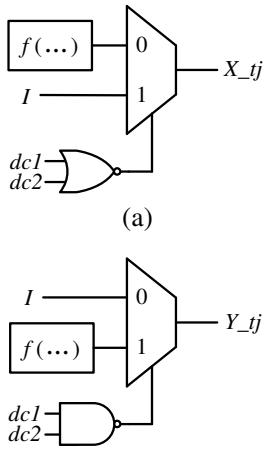
$$O\_tj = n5 \cdot n6 \cdot f + \overline{n5 \cdot n6} \cdot O \tag{1}$$

Since the condition $n5 = n6 = 1$ will never be reached under normal operation, $n5 \cdot n6$ will always be 0, and Equation (1) will reduce to $O\_tj = O$. Therefore, the Trojan will not be detected using formal equivalence checking. And, if we carefully craft the Trojan, the synthesis tools will keep this Trojan. Then, it is possible to create a communication channel to leak information about f through fault attack as we will show in Section V-B. Additionally, the redundancy introduced by f can be used to downgrade system performance, e.g., increasing power consumption.

Similarly, in Fig. 2 (b), the Trojan design uses the ODC condition for gate $g5$ in Fig. 1 to select between the normal output $n5$ and function g. The difference is that the input combination $n2 = n3 = 0$ can occur, but the output $n5$ does not have an effect on the primary output in this case. Thus, formal equivalence checking will still indicate that the modified circuit is equivalent to its Trojan free correspondence. However, there can be additional switching activities from $n5\_tj$ all the way through some dominating node when $g \neq n5$, which can be used to leak information about function g or increase power consumption.

It is important to point out why we use two multiplexers to construct the Trojan. As an example, the Trojan shown in Fig. 2 (a) can be described using Equation (1), which yields a single multiplexer for implementation. However, the select line of the multiplexer, i.e., $n5 \cdot n6$ will be constantly logical 0 due to a never reached don't care condition. This allows hardware Trojan detection methods based on switching probability analysis to easily identify the select line as a malicious signal. By introducing an additional multiplexer, each signal in the Trojan design has a high ability to switch, which will protect the Trojan from being detected.

When the Trojans are synthesized for implementation, they can result in different netlists as shown in Fig. 3 depending on the hardware design tool used. The resulting netlists for an ASIC toolchain (e.g, *Synopsys Design Compiler*) are shown in Fig. 3 (a) and (b). The tool will typically reduce the addition multiplexer and add a NOR (for the 11 don't care condition) or NAND (for the 00 don't care condition) gate to produce the select signal of the remaining multiplexer. As mentioned, such optimization will increase the detectability of the Trojan since the select signal may stay constant (for Trojans using SDC conditions).



dc1	*dc2*	*I*	*f*	*X_tj*	*Y_tj*
0	*0*	0	*0*	0	*0*
0	*0*	0	*1*	0	*1*
0	*0*	1	*0*	1	*0*
0	*0*	1	*1*	1	*1*
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	1	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	1	1
1	0	1	1	1	1
1	*1*	0	*0*	*0*	0
1	*1*	0	*1*	*1*	0
1	*1*	1	*0*	*0*	1
1	*1*	1	*1*	*1*	1

(c)

Fig. 3. Hardware Trojans implemented by different tools. (a) A Trojan based on 11 don't care condition implemented by ASIC tool. (b) A Trojan based on 00 don't care condition implemented by ASIC tool. (c) Trojans based on don't cares implemented by FPGA tool.

When the Trojans are implemented using FPGA tools (e.g., *Xilinx Vivado*), they are synthesized into a look up table (e.g., a LUT-4) whose truth tables are shown in Fig. 3 (c). For a Trojan that uses a 11 don't care condition, the entries marked as italic green (where $X\_tj = f$) will never be accessed while for a Trojan that uses a 00 don't care condition, the entries in italic red (where $Y\_tj = f$) will never be accessed. This reduces the Boolean functions to $X\_tj = I$ and $Y\_tj = I$ respectively. In both cases, the inputs and output of

the LUT still have a high switching probability, which will protect the Trojan from being detected.

Our Trojan design relies upon the inexactness of the logic synthesis process, which involves many NP-hard problems. As a result, synthesis tools often do not identify the Trojan design as redundant logic. However, there is still possibility that the Trojan design will be synthesized away using certain types of optimization with a high effort as we will shown in Section V-A. Synthesis attributes (e.g., the *keep* attribute) provides a possible technique to protect the Trojans from such undesirable optimizations. However, explicit specification of these attributes may expose the Trojan design. As an alternative approach, we can add two flip-flops for the select lines of the multiplexers to isolate the Trojan into a different combinational block other than the one where the don't care condition originates. Since most logic synthesis optimizations target local combinational regions, they will not be able to identify the correlation between the registered select lines. Formal equivalence checking has shown that the design with Trojan is still logically equivalent to the Trojan free baseline after adding two registers for the don't care signals.

When deployed, the Trojans are always on but will not cause any observable side effect before external attack. For ODC Trojans, the don't care conditions can be satisfied frequently. However, it is not possible for them to cause an observable change in any outputs. For SDC Trojans, the don't care conditions are theoretically never met. However, it is still possible to force these conditions to appear using fault attack. In the following subsection, we present our attack model for SDC Trojans.

C. Attack Model

Our attack model leverages the difference in path delay of the don't care signals to launch a fault attack. We add two registers for the don't care signals. Under normal operation, the registered don't care signals cannot reach the don't care condition to trigger the Trojan. However, since there is a delay difference in the paths to the don't care registers, there is a possibility that one register fails timing while the other does not. In this case, the register that fails timing can output a faulty value to allow the don't care condition to satisfy, which will trigger the Trojan.

Before launching a fault attack, we first run the design with Trojan under a normal operating clock frequency to initialize the design. This will allow the secret information (e.g., the cryptographic key) that is going to be leaked to load and stabilize. Such secret information typically will not be frequently updated. Thus, the secret data input to the Trojan will stay constant during the attack while the other input can switch. In this way, there will be a significantly higher probability to observe the constant secret value once the Trojan is activated.

We then gradually increase the clock frequency to cause faults. Before either of the don't-care registers has a timing violation, the faults should be caused by other critical path timing failures. Such faults tend to be random and thus do not reveal useful information. When only one of the registers fails timing, there will be a high probability to satisfy the don't care condition. In this case, the fault pattern will be more deterministic and thus can leak a significant amount of secret information. After further increasing the clock frequency, both don't-care registers will fail timing and the fault pattern will become random again. We use a concrete example to demonstrate our attack in Section V-B.

V. Experimental Results

In this section, we present experimental results. We insert Trojans into IWLS benchmarks [20] and show that they may survive after synthesis optimizations in Section V-A. We present fault attack analysis in Sections V-B to activate the Trojan in order to leak secret information. In Section V-C, we perform detectability analysis and provide possible defense techniques.

A. Trojan Design and Synthesis

We use several combinational benchmarks from *IWLS.org* [20] for Trojan insertion. We use pure combinational benchmarks because it is easier to perform an automated complete equivalence checking between the designs without and with Trojan using SAT tools [21], [22]. Figure 4 shows our hardware Trojan design and test flow.

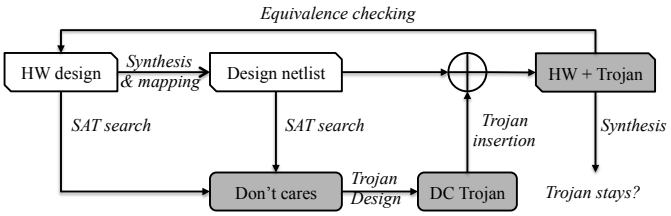


Fig. 4. Hardware Trojan design and test flow.

In our test, we first use *Synopsys Design Compiler* to synthesize the benchmarks. We use SAT tools [21], [22] to calculate the SDC and ODC conditions for each gate in the synthesized netlist and then use these don't cares to insert single bit version Trojans as shown in Fig. 2. We add an additional input to leak through the Trojan to a randomly selected primary output. In this way, we can use a SAT tool to automatically check if the Trojan remains after logic optimization. In an initial equivalence checking step, we have formally verified that all designs with Trojan are logically equivalent to the corresponding Trojan free benchmarks. This indicates that equivalence checking cannot detect our don't-care Trojan. The Trojan designs are then synthesized using ABC [21] with different optimizations to see if the inserted Trojan will stay after logic synthesis.

Table I summarizes the test results, where columns one and two show the benchmarks tested and the total number of gates in the synthesized design netlist; columns three and eleven report the number of SDC and ODC Trojans inserted respectively; the remaining columns indicate the number of Trojans stayed after logic optimizations using different synthesis commands and scripts in ABC [21]. Here, *resyn2*, *resyn2rs* and *drwsat2* are suggested logic synthesis scripts provided by ABC. They perform similar optimizations (i.e., combinations of technology-independent AIG rewriting, refactoring and restructuring) with increasing effort levels. The *dch* command computes alternative structural choices using SAT based simulation; *dch** runs the *dch* command with the *-r* option to skip choices with redundancy. *mfs2** adds the *-D 0 -M 0* options to invoke the *mfs2* command, which performs don't-care-based optimization. With additional command options, *dch** and *mfs2** both run at higher optimization efforts than *dch* and *mfs* respectively. We choose the *dch* and *mfs* commands since they both support redundancy removal. We refer interested readers to the documentation of ABC for more detailed information about the specific optimizations performed by these commands and scripts [21].

From Table I, internal don't care conditions widely exist. This provides the flexibility to search for don't cares for Trojan insertion. Take *C3540* as an example, there are 2476 gates in the design netlist. Among them, 588 gates have SDC conditions while 144 gates have ODC conditions for Trojan design. We can see the Trojans could possibly stay after logic synthesis, especially for larger benchmarks.

TABLE I
STATISTICS OF BENCHMARKS AND HARDWARE TROJAN DESIGNS.

Bench.	Gates	SDC Trojans								ODC Trojans							
		Ins.	resyn2	resyn2rs	drwsat2	dch	dch*	mfs2	mfs2*	*Ins.*	resyn2	resyn2rs	drwsat2	dch	dch*	mfs2	mfs2*
C1355	507	104	0	0	0	0	0	104	0	212	2	2	2	212	2	66	0
C1908	446	84	15	5	0	0	0	84	0	152	0	0	10	152	4	54	0
C2670	642	122	54	0	2	0	0	4	0	153	11	15	13	132	6	21	5
C3540	2476	588	359	171	22	1	0	479	0	144	91	85	69	144	63	99	19
C5315	3686	784	330	151	3	0	0	434	0	360	115	98	112	359	91	162	37
C6288	12625	1947	915	394	73	9	0	1910	0	1006	857	832	848	1006	792	957	372
C7552	3640	566	102	31	564	566	565	310	0	788	251	220	787	788	788	313	103
des	4671	1446	648	131	0	0	0	349	0	339	114	107	110	339	107	84	22
frg2	629	107	67	2	0	0	0	0	0	18	8	8	9	18	8	5	0
i10	3117	537	537	537	8	536	536	389	0	269	269	269	95	269	269	111	27
pair	2168	225	26	3	0	0	0	58	0	190	64	60	63	190	63	41	36

By comparing the results for *resyn2*, *resyn2rs* and *drwsat2* (similarly, *dch* vs. *dch** and *mfs2* vs. *mfs2**), higher optimization effort tends to eliminate more Trojans. The *mfs2** command will optimize away all SDC Trojans with the additional command options. However, ODC Trojans can still remain. The *dch** and *mfs** commands can synthesize away more Trojans than other scripts. This is because these two commands support redundancy removal and don't care optimization.

Table I reinforces that our Trojan designs can reside after logic synthesis. However, there are still optimization techniques that can better eliminate the Trojans, e.g., *dch* and *mfs2*. We further synthesize the Trojan designs listed in Table I using several synthesis tools with different optimizations. Test results show that Trojans inserted in relatively smaller benchmarks such as *C1908*, *C2670* and *frg2* are more likely to be optimized away. ASIC synthesis flows tend to keep the Trojan because the benchmarks are pre-synthesized using *Design Compiler* before Trojan insertion. FPGA synthesis flows tend to remove the Trojan when the design hierarchy is flattened. Table II indicates if one of the Trojan designs inserted in the *C6288* benchmark will remain after logic optimization using different synthesis tools.

TABLE II
THE EFFECT OF LOGIC OPTIMIZATION ON OUR HARDWARE TROJAN UNDER DIFFERENT SYNTHESIS TOOLS AND OPTIONS.

Synth Tool	Command	Option	Stays?
ABC	resyn2rs	–	✓
	drwsat2	–	✓
	dch	-r	✗
	mfs2	-D 0 -M 0	✗
yosys	proc; opt	–	✓
	synth_xilinx	–	✓
	synth_xilinx	flatten	✗
Quartus	normal flow	balanced	✓
	high effort	performance	✓
	Aggressive	area	✓
Vivado/ISE	default flow	hierarchy	✓
		none hierarchy	✗
Design Compiler	compile	–	✓
	compile_ultra	flatten	✓

In our test, we restricted the search for don't care conditions within signal pairs connected to the same gate. The number of don't cares can be far more than that if we extend our search to any signal pairs with a correlation, i.e., generalized don't cares as introduced in Section IV-A. Thus, it requires tremendous efforts to enumerate all possible don't care conditions for completely eliminating such hardware Trojans.

In real design practice, we need to choose SDCs closer to the primary outputs while ODCs closer to the primary inputs for hardware Trojan design. The increase in the number of logic levels will make it harder for logic synthesis tools to identify and eliminate the Trojans as redundancy. Further, we can use general conflict conditions for hardware Trojan design. The difference in logic levels will protect such generalized don't cares from been identified and introduce a larger difference in path delay that can be leveraged by our fault attack method described in the following subsection.

B. Fault Attack Analysis

In this subsection, we show that the hardware Trojan can open up a door for fault attack to extract secret information, using the attack method described in Section IV-C.

We use a 128-bit pipelined *AES* core from *opencores.org* for fault attack analysis. We set one select signal for the Trojan to be an input to the SBox of the final round and use a SAT solver [22] to search internal nodes of the SBox for the second select line so that the two have a conflict condition (e.g., they cannot be logical `1` simultaneously and thus represent a generalized don't care condition). We add two flip-flops for the select lines and then use the registered signals to multiplex the key to the *Cipher* output. Figure 5 shows the AES Trojan design used in our attack.

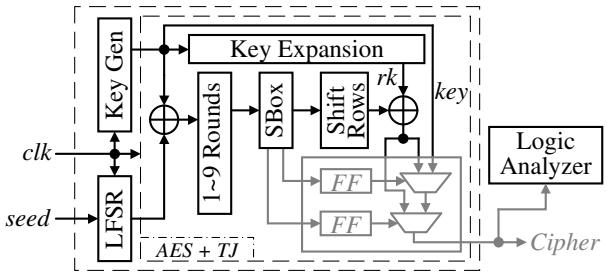


Fig. 5. AES design with an SDC Trojan implemented on FPGA.

We constrain the system clock frequency to 100 MHz and implement the AES Trojan design on SASEBO-W board with a Xilinx XC6S150T FPGA. We first run the design under 100 MHz using random plaintexts generated by a 128-bit LFSR (Linear Feedback Shift Register). Test results show that the AES design produces correct ciphertexts. In other words, the Trojan is not activated under normal operation. Another reason for running the core under a normal clock frequency prior to fault attack is that this will allow the secret key to successfully load and stabilize before the attack.

To launch a fault attack, we over-clock the design by gradually increasing the input clock frequency to the FPGA. Using the same random inputs, we start to observe faulty ciphertexts. However, the faults could be caused by a failed critical path in the original AES

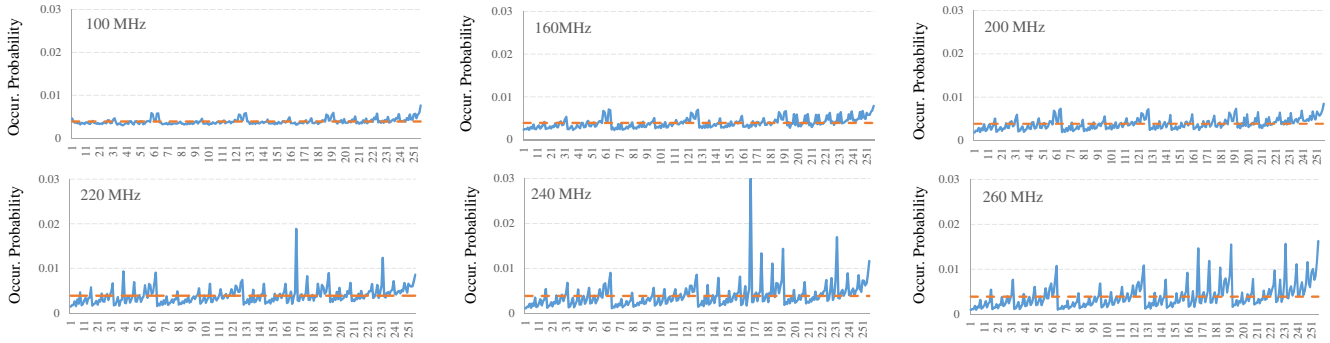


Fig. 6. Distribution of a ciphertext byte under different clock frequencies.

design *at first* because it is usually desirable to keep the Trojan off the critical path to escape from Trojan detection methods based on delay analysis. Extracting information about the secret key from such faults can be a challenging task due to the confusion and diffusion introduced by the AES algorithm. In addition, such faults can be dependent on the random plaintext and thus unpredictable. By comparison, if the fault is caused by the hardware Trojan (i.e., when either of the don't care signal register violates timing), *the secret key can be multiplexed to the faulty ciphertext directly*. Since the key is a stabilized constant value, a more deterministic fault pattern would happen (i.e., a significantly higher probability to observe certain values than others). Our observation has confirmed that increasing the system clock frequency will cause the distribution of the ciphertext bytes to change from close to even to significantly biased.

Figure 6 shows the distribution (occurrence probability) of a ciphertext byte under random inputs and different clock frequencies. We can see that at 100 MHz, the probability of observing each possible value ($0 \sim 255$) is close to 0.0039, i.e., 1/256 – the orange dashed lines in the plots. When increasing the clock frequency to 160 MHz and 200 MHz, there will be higher peaks caused by failed critical paths. At 220 MHz and 240 MHz, there will be a significantly higher peak, which corresponds to the correct key byte value. At this moment, one of the don't care registers should have failed timing and the Trojan is multiplexing the constant key value to the ciphertext. When further increasing the clock frequency to 260 MHz, the distribution moves towards even again, indicating that both don't care signal registers have a timing violation, which makes the fault pattern less predictable.

We then perform fault attack to recover the key, which is set to 0x0A8B1457_695355FB_8AC404E7_A79E0694 in our test. We first run the design under 100 MHz and then increase the system clock to 232 MHz to observe significantly biased distributions for all ciphertext bytes. Figure 7 shows the probability of observing different values for each ciphertext byte. We can see that the correct key values exactly correspond to the *highest* peaks for each curve in the figure. As demonstrated by the attack, we can simply over-clock the design to force the don't-care signals into a normally never reachable condition in order to activate the Trojan. We can precisely decode the key by calculating the distribution of the samples.

We further change the secret key and perform some additional attacks. Test results show that our attack method is independent of key values but more sensitive to input clock frequency changes.

C. Detectability Analysis

Hardware Trojan detection methods mainly fall into three different categories: functional and security verification [23], switching probability analysis [13], and side channel analysis [24].

Functional verification will not detect our Trojan since it hides behind internal don't care conditions. Such conditions are typically out of the reach of functional verification tools. Thus, functional verification tools will show that the design with Trojan is equivalent to its Trojan free specification. An important reason why our hardware Trojan is hard to detect is that hardware designs are usually described using functional languages (e.g., Verilog and VHDL) and models (e.g., Boolean function and Boolean gates). They do not carry the information required for modeling and reasoning about security properties. Thus, type enforced hardware design languages and tools [25], [26] provide a possible solution for detecting our hardware Trojans. These methods encode security attributes into the design for formal verification of hardware security properties. A promising technique is hardware information flow tracking [27], [28], which can precisely account for the flow of sensitive information even through unreachable states.

Switching probability analysis will fail to detect our Trojan as well since we use a don't care signal pair as the trigger. Consequently, every wire in our Trojan design has a high probability to switch. Using state-of-the-art switching probability analysis technique [13], the Boolean function re-constructed from partial simulation will be equivalent to the design under test, considering that the Trojan inserted and Trojan free designs are logically equivalent. Thus, it cannot detect our Trojan. However, if our Trojan is optimized and reduced to a single multiplexer, switching probability analysis will be effect in identifying it.

Side channel analysis cannot capture our Trojan due to several reasons. There is no golden reference design to compare the side channel statistics. The Trojans are always on but will not cause any observable side effect before external attack, which leads to no significant dynamics in the side channel measurements. In addition, our lightweight Trojan only uses two multiplexers as both the Trojan trigger and payload. This results in a significantly smaller fingerprint as compared to those complex Trojan designs. However, circuit frequency sweeping techniques [29], which gradually increases the circuit clock frequency to generate a number of delay signatures of the hardware design in order to analyze if the design contains a Trojan, take a similar approach to our fault attack. Thus, they can be used to activate our Trojan for further detection.

Although our hardware Trojan can be difficult to detect, there are still possible defense techniques. Logic synthesis analysis results show that our Trojan may be optimized away when using higher optimization effort. Further, logic synthesis techniques that enable redundancy removal or don't care based optimization (e.g., SAT based don't care search) can more efficiently eliminate our Trojans. In

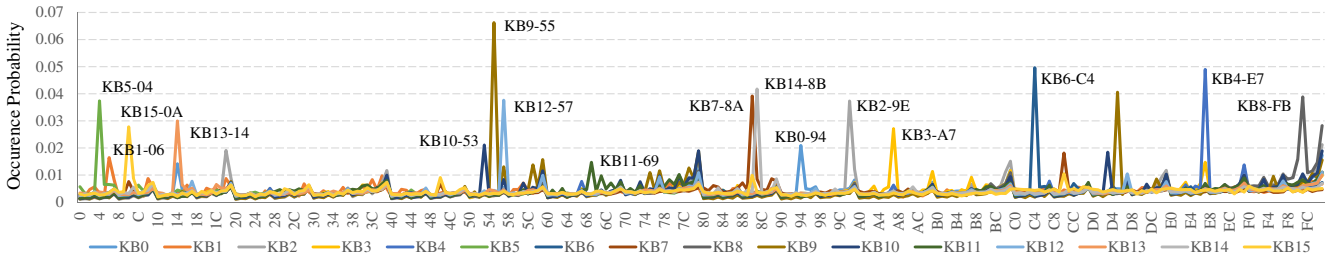


Fig. 7. Key guess results using fault attack.

addition, techniques such as security verification, switching probability analysis and circuit frequency sweeping all provide possible countermeasures to mitigate the new security threat.

VI. Conclusions

We demonstrate the possibility of exploiting internal don't care conditions for inserting a new type of hard-to-detect hardware Trojan. We show that such hardware Trojans may survive from logic optimizations under a number of synthesis tools using different options. When deployed, such hardware Trojan can open up a door for fault attacks to leak secret information or downgrade system performance by increasing power consumption. We present detectability analysis, suggest synthesis optimizations and discuss possible defense techniques to mitigate such a security threat.

VII. Acknowledgments

This work was supported in part by the National Science Foundation under grant NSF CNS-1527631 and by the National Natural Science Foundation of China under grant 61303224.

References

[1] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware trojans," in *the 15th International Conference on Cryptographic Hardware and Embedded Systems*, ser. CHES'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 197–214.

[2] R. Waugh, "Could a vulnerable computer chip allow hackers to down a boeing 787? 'back door' could allow cyber-criminals a way in," May 2012, http://www.dailymail.co.uk/sciencetech/article-2152284/.

[3] S. Adee, "The hunt for the kill switch," *Spectrum, IEEE*, vol. 45, no. 5, pp. 34–39, May 2008.

[4] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 10–25, Jan 2010.

[5] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware trojans," *Computer*, vol. 43, no. 10, pp. 39–46, Oct 2010.

[6] A. Baumgarten, M. Steffen, M. Clausman, and J. Zambreno, "A case study in hardware trojan design and implementation," *International Journal of Information Security*, vol. 10, no. 1, pp. 1–14, 2011.

[7] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *IEEE Symposium on Security and Privacy*, ser. SP'10, 2010, pp. 159–172.

[8] J. Zhang, F. Yuan, L. Wei, Z. Sun, and Q. Xu, "VeriTrust: Verification for hardware trust," in *Design Automation Conference (DAC'13)*, 2013, pp. 61:1–8.

[9] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using boolean functional analysis," in *ACM SIGSAC Conference on Computer; Communications Security*, ser. CCS '13, 2013, pp. 697–708.

[10] C. Sturton, M. Hicks, D. Wagner, and S. T. King, "Defeating uci: Building stealthy and malicious hardware," in *2011 IEEE Symposium on Security and Privacy*, ser. SP '11, 2011, pp. 64–77.

[11] J. Zhang, F. Yuan, and Q. Xu, "DeTrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans," in *ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 153–166.

[12] S. K. Haider, C. Jin, M. Ahmad, D. M. Shila, O. Khan, and M. van Dijk, "Advancing the state-of-the-art in hardware trojans detection," Cryptology ePrint Archive, Report 2014/943, 2014, http://eprint.iacr.org/2014/943.

[13] J. Zhang, F. Yuan, L. Wei, Y. Liu, and Q. Xu, "VeriTrust: Verification for hardware trust," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 7, pp. 1148–1161, July 2015.

[14] C. Dunbar and G. Qu, "Designing trusted embedded systems from finite state machines," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 5s, pp. 153:1–153:20, Oct. 2014.

[15] N. Fern, S. Kulkarni, and K. T. T. Cheng, "Hardware trojans hidden in rtl don't cares - automated insertion and prevention methodologies," in *Test Conference (ITC), 2015 IEEE International*, Oct 2015, pp. 1–8.

[16] N. Fern and K.-T. T. Cheng, "Detecting hardware trojans in unspecified functionality using mutation testing," in *IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 560–566.

[17] A. Nahiyan, K. Xiao, K. Yang, Y. Jin, D. Forte, and M. Tehranipoor, "AVFSM: A framework for identifying and mitigating vulnerabilities in fsms," in *Design Automation Conference (DAC'16)*, 2016, pp. 89:1–6.

[18] N. Fern, I. San, and K. T. T. Cheng, "Detecting hardware trojans in unspecified functionality through solving satisfiability problems," in *Asia and South Pacific Design Automation Conference*, 2017, pp. 598–504.

[19] S. Iman and M. Pedram, *Logic Synthesis for Low Power VLSI Designs*. Norwell, MA, USA: Kluwer Academic Publishers, 1998.

[20] IWLS, "IWLS Benchmarks Ver. 3.0," 2005, http://iwls.org/iwls2005/benchmarks.html.

[21] Berkeley Logic Synthesis and Verification Group, "ABC: A system for sequential synthesis and verification, release 10216." http://www.eecs.berkeley.edu/~alanmi/abc.

[22] C. Wolf, "Yosys open synthesis suite," http://www.clifford.at/yosys/.

[23] H. Salmani, M. Tehranipoor, and J. Plusquellic, "A novel technique for improving hardware trojan detection and reducing trojan activation time," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 1, pp. 112–125, Jan 2012.

[24] S. Narasimhan, D. Du, R. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, and S. Bhunia, "Hardware trojan detection by multiple-parameter side-channel analysis," *IEEE Transactions on Computers*, vol. 62, no. 11, pp. 2183–2195, Nov 2013.

[25] D. Zhang, Y. Wang, G. E. Suh, and A. C. Myers, "A hardware design language for timing-sensitive information-flow security," in *Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15, 2015, pp. 503–516.

[26] A. Ardeshiricham, W. Hu, J. Marxen, and R. Kastner, "Register transfer level information flow tracking for provably secure hardware design," in *Design, Automation & Test in Europe Conference & Exhibition*, 2017, pp. 1691–1696.

[27] Y. Jin and Y. Makris, "Proof carrying-based information flow tracking for data secrecy protection and hardware trust," in *2012 IEEE 30th VLSI Test Symposium (VTS)*, April 2012, pp. 252–257.

[28] W. Hu, B. Mao, J. Oberg, and R. Kastner, "Detecting hardware trojans with gate-level information-flow tracking," *Computer*, vol. 49, no. 8, pp. 44–52, Aug 2016.

[29] K. Xiao, X. Zhang, and M. Tehranipoor, "A clock sweeping technique for detecting hardware trojans impacting circuits delay," *IEEE Design & Test*, vol. 30, no. 2, pp. 26–34, 2013.