Formal Verification Approach to Detect Always-On Denial of Service Trojans in Pipelined Circuits

Kushal K. Ponugoti, Sudarshan K. Srinivasan, and Nimish Mathure
Department of Electrical & Computer Engineering
North Dakota State University (Fargo, ND, USA)
{kushalkumar.ponugoti, sudarshan.srinivasan, nimish.mathure}@ndsu.edu

2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS) | 978-1-7281-8281-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICECS53924.2021.9665617

***Abstract*—Always-On Denial of Service (DoS) Trojans with power drain payload can be disastrous in systems where on-chip power resources are limited. These Trojans are designed so that they have no impact on system behavior and hence, harder to detect. A formal verification method is presented to detect sequential always-on DoS Trojans in pipelined circuits and pipelined microprocessors. Since the method is proof-based, it provides a 100% accurate classification of sequential Trojan components. Another benefit of the approach is that it does not require a reference model, which is one of the requirements of many Trojan detection techniques (often a bottleneck to practical application). The efficiency and scalability of the proposed method have been evaluated on 36 benchmark circuits. The most complex of these benchmarks has as many as 135,898 gates. Detection times are very efficient with a 100% rate of detection, i.e., all Trojan sequential elements were detected and all non-trojan sequential elements were classified as such.**

***Index Terms*—Formal Verification, Hardware Trojans, Denial of Service, Always-On**

I. Introduction

Maliciously and intentionally induced faults in digital systems are called hardware Trojans. Usage of third-party intellectual properties and CAD tools, or compromised designers could be sources of hardware Trojans. Circuits with Trojans may leak private data over side-channels, change its intended functionality, degrade system performance over time, or completely halt a working system. These malicious events can be disastrous in security-critical applications such as defense systems, medical devices, financial systems, etc. Detection of hardware Trojans has therefore become a very critical problem in this domain.

An overview of hardware Trojans, their taxonomy, and historical progression has been extensively discussed in [1], [2]. The Denial of Service (DoS) class of Trojans causes systems to become unresponsive by placing them in sleep mode [3], prevent a microprocessor core from executing instructions, or drain power continuously [4], [5]. Some Trojans are always designed to be active (always-on), while some are activated only when a rare circuit state occurs (trigger-based). The focus of this work is the detection of Always-On DoS Trojans. Such Trojans are harder to detect as they can be designed in a way that they have no impact on the behavior of the digital circuit/system. This class of Trojans can lead to premature component aging and consume extra power. This behavior can be disastrous in embedded systems such as pacemakers and other medical implants where on-chip power resources are minimal and hence detecting them is a significant problem.

The main contribution of this work is a general formal verification method that can detect always-on DoS Trojans in pipelined circuits that satisfy the threat model described below. The method applies to register-transfer-level (RTL)/netlist versions of the design but is not applicable for Trojans embedded during fabrication. Formal verification methods are typically not applicable post-silicon. Another benefit of the method is that it does not require a reference or a golden model for detection. The need for a reference model often becomes a bottleneck to practical application.

The rest of the paper is organized as follows. The threat model and the Trojan attack are described in Section II. Section III discusses related work on detecting always-on DoS Trojans. In Section IV, a formal verification methodology to detect always-on DoS Trojans in pipelined combinational circuits is presented. A formal verification methodology to detect always-on DoS Trojans in pipelined microprocessor circuits is proposed in Section V. Verification results are shown in Section VI, and Section VII discusses future work and concludes the manuscript.

II. Threat Model and Attack Description

Fig. 1 shows two examples of always-on DoS Trojans (T1 in Fig. 1 (a) and T2 in Fig. 1 (b)) and Fig. 2 shows the embedding of the second Trojan (T2) in a five-stage pipelined processor circuit. Both Trojans shown in Fig 1. incorporate rotating shift registers that continually drain power either statically or dynamically. The Trojans hijack the ”out” wire using a Mux network. Logically, the ”Out” wire is always connected to the

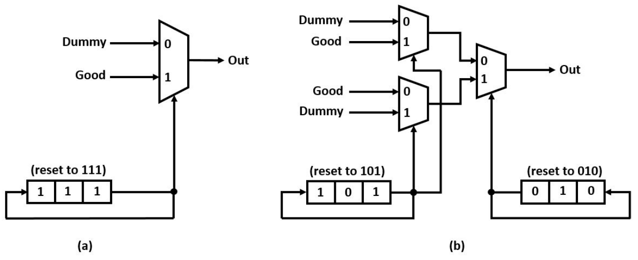

Fig. 1. (a) Trojan, T1, using a reset to 111 register (b) Trojan, T2, using a reset to 101 and a reset to 010 register

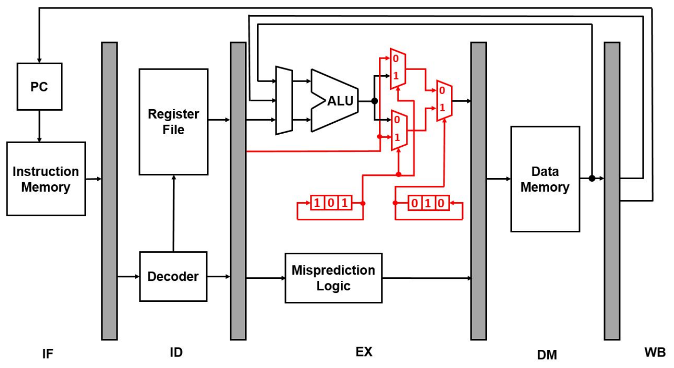

Fig. 2. Example of an always-on DoS Trojan circuit embedded in a 5-stage microprocessor host circuit

"Good" wire, which is the correct output. The "Dummy" wire is never propagated to "Out." The Mux network ensures that the Trojan output is connected to the host without impacting host functionality, ensuring the Trojan will evade isolation during synthesis. The post-synthesis netlist of Trojan T1 is shown in Fig. 3, demonstrating that the Trojan has not been simplified during synthesis. We applied two synthesis tools, Intel Quartus (version 20.1) and Synopsys Design Compiler (version 2018), to both Trojans T1 and T2 and found that both the tools do not simplify or detect the presence of these Trojans. What is shown in Fig. 1 is only an example, and there are numerous ways to design such Trojans in a more complex, convoluted, and intricate manner to evade reduction by synthesis. Also, such Trojans can be embedded in the host circuit post-synthesis.

There are three properties that characterize the attack model that the proposed method is designed to detect. First, inputs/outputs of the Trojan are connected to the host circuit and therefore the Trojan cannot be detected by methods that rely on the isolation of the Trojan module during synthesis (which is a standard approach used for always-on Trojans). Second, the Trojan has no impact on circuit functionality (intended or unintended) and therefore cannot be detected by methods that rely on circuit valuation for various scenarios. Third, the attack is typically a power drain over time.

III. Related Work

Detecting always-on DoS Trojans embedded during the fabrication process has been extensively discussed in [6]–[8] which is not in the scope of this work. Their methods are applicable post-fabrication and do not apply to the RTL. Our method focuses on detecting Trojans in the RTL, before fabrication. It is much less expensive to detect Trojans embedded in the RTL before fabrication.

```
t1 : reg1_0 port map (d => regout2, clock => clk, reset => rst, q => regout0);
t2 : reg1_1 port map (d => regout0, clock => clk, reset => rst, q => regout1);
t3 : reg1_2 port map (d => regout1, clock => clk, reset => rst, q => regout2);
t4 : mux2 port map( a => dummy, b => good, mux_sel => regout2, f => z);
```

Fig. 3. Figure shows the netlist generated by the synthesis tool, indicating the presence of the shift register

Unused Circuit Identification (UCI) [9], FANCI [10] and VeriTrust [11] are focused on detecting trigger-based Trojans, where the idea is that they look for circuit components that are rarely activated. However, always-on Trojans are always activated without using a trigger, and therefore these methods do not apply. Sturton et al. [12] and Zhang et al. [13] have also shown that the aforementioned techniques can be circumvented. Also, it is unclear if they are applicable for detecting Trojans in pipelined circuits. It should also be noted that malicious designers or attackers always try to upgrade their approaches to implanting these Trojans to circumvent existing detection techniques. Hence, formal methods are becoming more necessary than before as they can formally prove whether a design is Trojan-free or not.

Trust-Hub [4], [5] has a wide suite of RTL hardware Trojan benchmarks of which 3 are based on always-on DoS. Kumar et al. [14] provided a solution to detect DoS Trojans from Trust-Hub by exploiting the fact that the Trojan shift register has unconnected output ports, which is detected by the synthesis tool. However, as shown in Fig. 1, DoS Trojans can be designed to circumvent such detection, by connecting the Trojan register output to the host without impacting host functionality. Fern et al. [15], proposed a formal verification approach that can be used to detect always-on DoS Trojans, however, this is not their intended target. The basic idea is to constrain a flip-flop to 0 in symbolic state w and step the circuit. In another copy of w, the same flip-flop is constrained to 1 and the circuit is stepped again. If the outputs are equal for both steps, it can be concluded that the flip-flop has no impact on the circuit and is, therefore, a Trojan. Note that this approach is insufficient for pipelined circuits such as microprocessors, as only some flip-flops will impact the circuit outputs when the circuit is stepped just once. The method proposed in this paper extends the aforementioned approach to apply to pipelined circuits and pipelined microprocessor circuits.

IV. Formal DoS Detection For Pipelined Circuits

Consider a pipelined circuit with four stages as shown in Fig. 4. If the Fern et al. [15] approach is applied, a flip-flop located in the pipeline registers between c3 and c4 will impact the circuit output when the circuit is stepped once. However, flip-flops in pipeline registers of other stages will

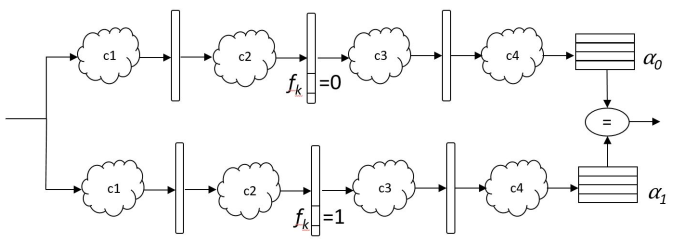

Fig. 4. Figure shows a generic 4-stage pipelined circuit and depicts Property 1 for always-on DoS Trojan detection

not impact the output after one step. Therefore, all the flip-flops in pipeline registers between stages c1 and c2 and stages c2 and c3 will be flagged as Trojans and the method will not be able to differentiate Trojan flip-flops and non-trojan flip-flops in these stages. If the same approach above is used but with two steps of the circuit instead of one step, the impact of the flip-flops between stages c3 and c4 will be on the outputs, but not the impact from pipeline registers in other stages. The proposed approach is to augment the circuit with an array with n rows as a history variable, where n is the number of stages in the pipeline. History variables do not impact the circuit functionality and are used only for verification purposes. Rows 1, 2, .., n of the array will record the outputs of the circuit after one, two, ..., n steps respectively. The array will thus record the impact of the values of all the pipeline registers on the outputs and this structure is used for always-on DoS Trojan detection. The formal property that exploits this idea is described below and is also depicted in Fig. 4.

Without loss of generality, consider a pipelined circuit with p inputs $i_1, \ldots, i_p$; q outputs $o_1, \ldots, o_q$; r flip-flops $f_1, \ldots, f_r$; and n stages. The flip-flops $f_1, \ldots, f_r$ correspond to the pipeline registers in the circuit. $\xi()$ is the function that corresponds to one step of the pipelined circuit. i.e., given the current state and inputs, $\xi()$ gives the next state and outputs: $\langle o_1, \ldots, o_q, f_1^1, \ldots, f_r^1 \rangle = \xi(i_1, \ldots, i_p, f_1, \ldots, f_r)$, where $f_1^1, \ldots, f_r^1$ correspond to the next state of the flip-flops. α is a register array with n q-bit registers $\alpha[0], \ldots, \alpha[n\text{-}1]$.

Property 1 *(DoS Detection For Pipelines)* Flip-flop k is a DoS Trojan if register arrays α_0 and α_1 are equal *for all* $i_1, \ldots, i_p \in$ [0,1] and *for all* $f_1, \ldots, f_r \in$ [0,1], under the following four constraints:

1. $\langle \alpha_0[0][0], \ldots, \alpha_0[0][q\text{-}1], f_{1_0}^0, \ldots, f_{r_0}^0 \rangle = \xi(i_1, \ldots, i_p, f_1, \ldots, f_k in, \ldots, f_r)$
2. $\bigwedge_{j=1}^{n-1} \langle \alpha_0[j][0], \ldots, \alpha_0[j][q\text{-}1], f_{1_0}^j, \ldots, f_{r_0}^j \rangle = \xi(0, \ldots, 0, f_{1_0}^{j-1}, \ldots, f_{r_0}^{j-1})$
3. $\langle \alpha_1[0][0], \ldots, \alpha_1[0][q\text{-}1], f_{1_1}^0, \ldots, f_{r_1}^0 \rangle = \xi(i_1, \ldots, i_p, f_1, \ldots, f_k\text{=}1, \ldots, f_r)$
4. $\bigwedge_{j=1}^{n-1} \langle \alpha_1[j][0], \ldots, \alpha_1[j][q\text{-}1], f_{1_1}^j, \ldots, f_{r_1}^j \rangle = \xi(0, \ldots, 0, f_{1_1}^{j-1}, \ldots, f_{r_1}^{j-1})$

In the property above, in constraint 1, flip-flop f_k is forced to 0 and the pipelined circuit is stepped. The outputs of the circuit are recorded in $\alpha_0[0]$. Constraint 2 corresponds to n-1 flushing steps of the pipeline circuit. A flushing step is when the pipeline is stepped forward with all the inputs set to 0. The outputs of each of these flushing steps are recorded in $\alpha_0[j]$. Constraints 3 and 4 are similar to constraints 1 and 2, with the only difference being that f_k is initially forced to 1 instead of 0 and the circuit outputs are stored in α_1. In the end if, α_0 and α_1 are equal, this implies that under all possible pipeline circuit states and inputs, flip-flop f_k has no impact on the circuit outputs and functionality and can therefore be classified as a DoS Trojan. The above property should be checked for every flip-flop in the circuit.

V. Formal DoS Detection For Microprocessor Circuits

Property 1 is insufficient for detection of DoS Trojans in microprocessor pipelines, as these circuits have programmer visible state components such as the program counter, register file, memory, and other special registers and flags in addition to pipeline registers. Also, these circuits have feedback and stalling behaviors that influence the pipeline operation. Another property is provided for DoS Trojan detection in microprocessor circuits. Note that the property can easily be adapted for other circuits with similar characteristics.

Consider a processor circuit with three programmer visible components including a program counter (ρ), register file (ζ), and and memory (μ). Other flags and special registers if present can be treated similar to the program counter. Also, the processor has r flip-flops $f_1, \ldots, f_r$, that correspond to pipeline registers and any other state in the processor that is not a programmer visible component. $\xi()$ is the function that corresponds to one step of the circuit: $\langle \rho^1, \zeta^1, \mu^1, f_1^1, \ldots, f_r^1 \rangle$ = $\xi(\rho, \zeta, \mu, f_1, \ldots, f_r)$, where $f_1^1, \ldots, f_r^1$ correspond to the next state of the flip-flops and ρ^1, ζ^1, and μ^1 correspond to the next state of the programmer visible components.

ξ_{flush} is a flushing function of the processor and is a variation of ξ. ξ_{flush} steps the processor without fetching any new instructions and was first proposed by Burch and Dill [16] in the context of safety verification of pipelined processors. The flushing function is implemented by modifying ξ. The program counter is stalled and bubbles are inserted into the fetch stage. In-flight branch instructions are allowed to still update the program counter. A sufficient number of applications of the flush function will complete all in-flight instructions in a processor state and empty the pipeline. The number of flushing steps required to guarantee an empty pipeline starting from any state of a processor (denoted as n for Property 2) can be computed based on the design of the processor. The results computed by the in-flight instructions will be updated onto the programmer visible components.

Property 2 *(DoS Detection For Microprocessor)* Flip-flop k is a DoS Trojan if ρ_0^n=ρ_1^n and ζ_0^n=ζ_1^n and μ_0^n=μ_1^n *for all* values of ρ, ζ, and μ and *for all* $f_1, \ldots, f_r \in$ [0,1], under the following four constraints:

1. $\langle \rho_0^0, \zeta_0^0, \mu_0^0, f_{1_0}^0, \ldots, f_{r_0}^0 \rangle = \xi(\rho, \zeta, \mu, f_1, \ldots, f_k\text{=}0, \ldots, f_r)$
2. $\bigwedge_{j=1}^{n} \langle \rho_0^j, \zeta_0^j, \mu_0^j, f_{1_0}^j, \ldots, f_{r_0}^j \rangle = \xi_{flush}(\rho_0^{j-1}, \zeta_0^{j-1}, \mu_0^{j-1}, f_{1_0}^{j-1}, \ldots, f_{r_0}^{j-1})$
3. $\langle \rho_1^0, \zeta_1^0, \mu_1^0, f_{1_1}^0, \ldots, f_{r_1}^0 \rangle = \xi(\rho, \zeta, \mu, f_1, \ldots, f_k\text{=}1, \ldots, f_r)$
4. $\bigwedge_{j=1}^{n} \langle \rho_1^j, \zeta_1^j, \mu_1^j, f_{1_1}^j, \ldots, f_{r_1}^j \rangle = \xi_{flush}(\rho_1^{j-1}, \zeta_1^{j-1}, \mu_0^{j-1}, f_{1_1}^{j-1}, \ldots, f_{r_1}^{j-1})$

Table 1. Verification Results for Pipelined Circuits (Property 1)

Circuit Benchmarks	Gates #	Flip-Flops #	Verification Time [secs]		Average Verification Time per Flip-Flop [secs]	
			Non-Trojan Flip-Flops	Trojan Flip-Flops	Non Trojan Flip-Flops	Trojan Flip-Flops
mul-4-p-T1	135	53	0.42	6.63	0.01	0.60
mul-8-p-T1	451	163	1.53	23.58	0.01	2.14
mul-16-p-T1	1,639	556	5.45	82.86	0.01	7.53
mul-20-p-T1	2,664	859	16.96	139.93	0.02	12.72
mul-24-p-T1	3,855	1,226	36.45	199.25	0.03	18.11
mul-28-p-T1	5,271	1,583	78.60	280.10	0.05	25.46
mul-32-p-T1	12,751	1,952	135.87	396.13	0.07	36.01
i85-c432-p-T1	223	152	1.41	7.48	0.01	0.68
i85-c1355-p-T1	609	155	1.44	20.03	0.01	1.82
i85-c1908-p-T1	943	201	3.84	41.83	0.02	3.80
i85-c2670-p-T1	1,256	559	16.44	106.81	0.03	9.71
mul-4-p-T2	147	56	0.16	8.73	0.01	0.62
mul-8-p-T2	463	166	1.56	30.70	0.01	2.19
mul-16-p-T2	1,641	559	5.46	106.27	0.01	7.59
mul-20-p-T2	2,675	862	17.00	179.52	0.02	12.82
mul-24-p-T2	3,867	1,229	48.82	256.64	0.04	18.33
mul-28-p-T2	5,283	1,586	78.73	359.11	0.05	25.65
mul-32-p-T2	12,763	1,955	155.30	510.90	0.08	36.49
i85-c432-p-T2	235	155	1.43	10.25	0.01	0.73
i85-c1355-p-T2	621	158	1.47	26.65	0.01	1.90
i85-c1908-p-T2	955	204	5.71	54.90	0.03	3.92
i85-c2670-p-T2	1,268	562	21.95	136.40	0.04	9.74

Table 2. Verification Results for Pipelined Processor Circuits (Property 2)

Processor Benchmarks	Gates #	Flip-Flops #	Non-Programmer Visible Flip-Flops #	Verification Time [secs]		Average Verification Time per Flip-Flop [secs]	
				Non-Trojan Flip-Flops	Trojan Flip-Flops	Non Trojan Flip-Flops	Trojan Flip-Flops
m-16-bit-T1	2,961	1,256	370	3.60	43.14	0.01	3.92
m-32-bit-T1	10,171	2,840	640	12.60	176.22	0.02	14.71
m-64-bit-T1	36,122	7,576	1,120	44.39	860.79	0.04	78.25
m-128-bit-T1	135,773	23,192	2,080	206.94	14,424.21	0.10	1311.29
r-v-32-bit-T1	10,284	2,840	640	18.88	215.75	0.03	19.61
r-v-64-bit-T1	36,236	7,576	1,120	77.67	1,102.86	0.07	100.26
r-v-128-bit-T1	135,886	23,192	2,080	269.00	15,511.76	0.13	1410.16
m-16-bit-T2	2,973	1,259	373	3.63	57.03	0.01	4.07
m-32-bit-T2	10,183	2,843	643	18.94	213.40	0.03	15.24
m-64-bit-T2	36,134	7,579	1,123	55.49	1,186.08	0.05	84.72
m-128-bit-T2	135,785	23,195	2,083	248.28	19,892.61	0.12	1420.90
r-v-32-bit-T2	10,296	2,843	643	25.16	333.65	0.04	23.83
r-v-64-bit-T2	36,248	7,579	1,123	99.84	1,735.20	0.09	123.94
r-v-128-bit-T2	135,898	23,195	2,083	310.35	26,210.26	0.15	1872.16

In Property 2, f_k is the flip-flop that is being checked to see if it is a Trojan. Constraint 1 corresponds to a step of the processor circuit with f_k set to 0. Let w_0 correspond to the initial state with f_k=0 and let v_0 be the state obtained by stepping the processor from w_0. All other flip-flops and state (program counter, register file, and memory) in w_0 are unconstrained. Therefore, the property will be verified for all possible values of the other flip-flops and state. Constraint 2 corresponds to applying n flushing steps to v_0. Let v_0^f correspond to the resulting flushed state. Constraint 3 is similar to constraint 1, except that f_k set to 1 instead of 0. Let w_1 correspond to the initial state with f_k=1 and let v_1 be the state obtained by stepping the processor from w_1. Constraint 4 is similar to constraint 2 in that it corresponds to applying n flushing steps to v_1. Let v_1^f correspond to the resulting flushed state. Property 2 checks if the programmer visible components in v_0^f and v_1^f are equal. Since in both w_0 and w_1, all other state elements were unconstrained, if Property 2 is satisfied, this implies that f_k has no impact on the programmer visible components for any state of the processor and can therefore be classified as a Trojan flip-flop. The flushing steps of constraints 2 and 4 ensure that if f_k does impact circuit functionality, the impact will be propagated to the programmer visible components. The above property should be checked for every flip-flop in the circuit that is not a programmer visible component. Since programmer visible components are essential to the circuit functionality, if they are infected, this will distort circuit behavior and be flagged during testing and verification for functional correctness.

VI. Experimental Results

Table 1 and Table 2 show verification results for the benchmarks by employing Properties 1 and 2, respectively. The tables also give the complexity of the benchmark circuits to be evaluated, in terms of the total gate and flip-flop count. The properties were checked using version 4.3.2 of the Z3 SMT solver [17]. The verification experiments were performed on an Intel(R) Core(TM) i7 - 8700 CPU @ 3.2 GHz with 32GB of RAM and 64-bit operating system.

In Table 1, *mul-n-p-t* are pipelined multipliers, where n is the width of the multiplier and *-p* indicates that the circuit is pipelined. Four ISCAS-85 circuits [18] are also used which are labelled *i85-c-p-t*, where c is the ISCAS-85 circuit name. The original ISCAS-85 benchmarks are not pipelined. Pipelined versions (*-p*) of these benchmarks were created for evaluating the efficacy of Property 1. The benchmarks are also marked with T1 or T2 indicating which Trojan was injected into the circuit. All the multipliers and ISCAS-85 benchmarks are pipelined into four stages each. The benchmarks with Trojan T1 had 3 Trojan flip-flops while the benchmarks with Trojan T2 had 6 Trojan flip-flops. Also, an additional 8 Trojan flip-flops were randomly embedded in the pipelined registers. These additional Trojan flip-flops were not connected with the host circuit. Property 1 was checked on all the flip-flops in the circuit. All the 11 Trojan flip-flops satisfied the property, i.e., they had no impact on the circuit functionality, whereas all non-trojan flip-flops did not satisfy the property and instead resulted in a counterexample. Hence, Property 1 provided a 100% rate of detection for Trojan flip-flops and correctly classified all flip-flops as either Trojan or non-trojan. Table 1 also shows the verification times for checking Property 1 on Trojan and non-trojan flip-flops separately and also shows the average time/flip-flop for Trojan and non-trojan flip-flops. As can be seen from the table, the time for detecting Trojan flip-flops is much higher than the time required for checking non-trojan flip-flops. The reason for this discrepancy is that non-trojan flip-flops lead to a counterexample, whereas detection of trojan-flops requires showing that Property 1 is satisfied for all possible input values and flip-flop values. Overall, this is a benefit for large circuits, as the number of Trojan flip-flops will be minuscule (if any) compared to the number of non-trojan flip-flops and should lead to efficient detection times.

In Table 2, two types of processor benchmarks were used (MIPS, indicated by m, and RISC-V indicated by $r - v$). Both types of processor benchmarks implemented a 5-stage pipeline. The data path width of the pipelined processor circuits was varied from 16 to 128. Apart from the 3 Trojan flip-flops in T1 and 6 Trojan flip-flops in T2, an additional 16 Trojan flip-flops were embedded in the pipelined registers in each of the processor benchmarks. As shown in Table 2, property 2 was checked on all the non-programmer visible flip-flops i.e. any flip-flop which is not a part of the program counter, register file, instruction memory, and data memory. Property 2 also correctly classified all the Trojan and non-trojan flip-flops in the processor benchmarks and provided a 100% rate of detection for Trojan flip-flops. For Property 2 as well, the verification time required to identify Trojan flip-flops was much higher than what was required for non-trojan flip-flops.

VII. Conclusion

Two properties are proposed to detect DoS Trojans in pipelined circuits. Property 1 can be used for any combinational circuit that is pipelined, i.e., the assumption is that the original circuit has no state. Property 2 can be used for pipelined circuits, where the original circuit has a state. Microprocessors are examples of such circuits, where the programmer visible components are state components inherent to the circuit. Property 2 was specifically designed for microprocessor circuits and can be extended to other pipelined circuits with the inherent state by treating these state components similar to how the program counter was treated in Property 2. The key takeaway with both properties is that the time required to identify non-trojan flip-flops is minuscule compared to that required for the detection of Trojan flip-flops and this is a big benefit because a majority of the flip-flops will be non-trojan. Abstractions are typically used to improve the efficiency of verification. For future work, abstraction techniques will be explored to significantly increase the size of circuits that can be verified.

VIII. Acknowledgment

This paper is based upon work supported by the National Science Foundation under Grant No. CNS-2117190.

References

[1] 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, 2010.

[2] M. Xue, C. Gu, W. Liu, S. Yu, and M. O’Neill, “Ten years of hardware trojans: a survey from the attacker’s perspective,” *IET Computers & Digital Techniques*, vol. 14, no. 6, pp. 231–246, 2020.

[3] R. Elnaggar, K. Chakrabarty, and M. B. Tahoori, “Hardware trojan detection using changepoint-based anomaly detection techniques,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2706–2719, 2019.

[4] H. Salmani, M. Tehranipoor, and R. Karri, “On design vulnerability analysis and trust benchmarks development,” in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, 2013, pp. 471–474.

[5] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, “Benchmarking of hardware trojans and maliciously affected circuits,” *Journal of Hardware and Systems Security*, vol. 1, no. 1, pp. 85–102, 2017.

[6] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, “Stealthy dopant-level hardware trojans,” in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2013, pp. 197–214.

[7] Y. Liu, Y. Jin, A. Nosratinia, and Y. Makris, “Silicon demonstration of hardware trojan design and detection in wireless cryptographic ics,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 4, pp. 1506–1519, 2017.

[8] Y. Shiyanovskii, F. Wolff, A. Rajendran, C. Papachristou, D. Weyer, and W. Clay, “Process reliability based trojans through nbti and hci effects,” in *2010 NASA/ESA Conference on Adaptive Hardware and Systems*, 2010, pp. 215–222.

[9] 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 *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 159–172.

[10] A. Waksman, M. Suozzo, and S. Sethumadhavan, “Fanci: identification of stealthy malicious logic using boolean functional analysis,” in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 697–708.

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

[12] 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*, 2011, pp. 64–77.

[13] J. Zhang, F. Yuan, and Q. Xu, “Detrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans,” in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 153–166.

[14] K. S. Kumar, R. Chanamala, S. R. Sahoo, and K. K. Mahapatra, “An improved aes hardware trojan benchmark to validate trojan detection schemes in an asic design flow,” in *2015 19th International Symposium on VLSI Design and Test*, 2015, pp. 1–6.

[15] N. Fern, I. San, and K.-T. T. Cheng, “Detecting hardware trojans in unspecified functionality through solving satisfiability problems,” in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 598–504.

[16] J. R. Burch and D. L. Dill, “Automatic verification of pipelined microprocessor control,” in *International Conference on Computer Aided Verification*. Springer, 1994, pp. 68–80.

[17] L. De Moura and N. Bjørner, “Z3: An efficient smt solver,” in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[18] M. Hansen, H. Yalcin, and J. Hayes, “Unveiling the iscas-85 benchmarks: a case study in reverse engineering,” *IEEE Design Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.