POLITECNICO DI TORINO
Repository ISTITUZIONALE

A Benchmark Suite of RT-level Hardware Trojansfor Pipelined Microprocessor Cores

Original
A Benchmark Suite of RT-level Hardware Trojansfor Pipelined Microprocessor Cores / Damljanovic, A., Ruospo, A., Sanchez Sanchez, E., Squillero, G.. - ELETTRONICO. - (2021). (24th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS) Vienna, Austria April 7-9).

Availability:
This version is available at: 11583/2882631 since: 2021-04-02T14:08:02Z

Publisher:
IEEE

Published
DOI:

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

17 September 2026

A Benchmark Suite of RT-level Hardware Trojans for Pipelined Microprocessor Cores

Aleksa Damljanovic, Annachiara Ruospo, Ernesto Sanchez, Giovanni Squillero
{aleksa.damljanovic, annachiara.ruospo, ernesto.sanchez, giovanni.squillero}@polito.it
Dipartimento di Automatica ed Informatica, Politecnico di Torino – Torino, Italy

***Abstract*—Recent trends in integrated circuits industry include decentralization of the production flow by involving different integration teams, third-party IP vendors and other untrusted entities. As a result, this is opening up a door to new types of attacks that may lead to devastating consequences, such as denial of service or data leakage. Therefore, the problem of ensuring hardware security has gained much attention in the last years, especially early in the design cycle, when an attacker may insert malicious circuitry at register transfer (RT) or gate level. Due to the increased complexity of modern devices, the research community is spending a lot of effort in developing more sophisticated detection methodologies and smarter attacks. However, the main problem is that they are validated on the existing benchmarks that do not reflect the real complexity. Trying to fill this gap, this paper proposes a set of RT-Level Hardware Trojan benchmarks injected in a RISC-based pipelined microprocessor core. To prove the viability, the impacts on area, power and frequency are presented and discussed. For any proposed Hardware Trojan, the functional description, the implementation details and the effects once activated are provided.**

***Index Terms*—Hardware Security, Benchmark, Hardware Trojans, RTL, Microprocessor Cores**

I. Introduction

In the last years, the growing complexity of modern devices and the fabrication costs led the Integrated Circuit (IC) industry to pursue a new global business model. Different companies are deeply involved in all phases of the IC supply chain. The outsourcing of part of the process to untrusted third-party entities makes such decentralized supply chain vulnerable to security attacks and malicious insertions. Furthermore, it raises increasing concerns about hardware security of the products, especially when dealing with strict security and safety requirements in critical applications such as avionics, communications, military, etc. In this context, Hardware Trojans (HTs) are gaining worldwide attention not only from industry and academia, but also from government bodies [1]. A HT is any malicious modification of the design whose purpose is to endanger the security of the hardware. A malicious alteration can be performed during any phase of the production cycle. A category of HTs are those inserted at the manufacturing stage. In this last scenario, an adversary could access the mask and modify it to add malicious logic. It is supposed that such logic is inserted in an intelligent manner, difficult to activate with manufacturing tests given the combination of rare internal signals values that is used to trigger it. However, more interesting are Trojans inserted earlier in the design cycle, at the register transfer level (RTL) or gate-level. Apart from superfluous complex reverse engineering, an attacker inserting a Trojan early in the design process may take advantage of the vast design space. Also, such Trojan may potentially remain hidden even in the following generations of the device.

TABLE I
Number of RTL Hardware Trojan benchmarks available on Trust-Hub [4] [5]

Design Class	AES	b19	BasicRSA	MC8051	memctrl	PIC	RS-232	wb_conmax	TOTAL
Number	21	3	4	7	1	4	10	2	52

During the last years, huge effort has been invested in developing detection methodologies as well as designing benchmark circuits to favor the advancements in research. Indeed, the research community has received a strong drive to adopt open benchmarks for validating their detection techniques. In this light, many HT models have been proposed [2], [3]. However, the growing complexity of modern devices as well as more mature and elaborate detection methodologies call for more complex benchmark circuits. Some of the authors in their studies proposed different HT taxonomies based on the insertion phase, location, abstraction level, activation mechanism, effects, etc. However, it is complicated to create a HT model given the whole spectrum of constantly evolving attacks and adversaries that are gaining access to more and more phases of the IC development.

A common trend is to use benchmarks released from the Trust-Hub platform [4], [5]. Considering HTs at RT-Level, only 8 typologies of benchmarks are currently available in Trust-Hub (Table I), and none of them is applied to a pipelined processor similar to the ones used in the real life, as for example the ones in the automotive applications. This is even more concerning, given a higher flexibility for implementing different kinds of malicious functions at RTL. The available HTs are injected on a small 8-bit 8051 microprocessor, and a detection technique has already been proposed in [6]. Hence, even the state-of-the-art HT detection techniques are validated on obsolete benchmarks that do not reflect the true complexity of the modern embedded devices. As stated in [3], in order

978-1-6654-3595-6/21/$31.00 ©2021 IEEE

to further support the development of appropriate detection methods, the design and implementation of practical HTs needs to be considered.

To fill this gap, this paper releases a total of 28 Hardware Trojans Benchmarks targeting a pipelined RISC microprocessor core[1]. We started from the structure of 8 HTs placed in different CPU's locations. Then, we derived additional benchmarks by modifying their trigger mechanism. Their design follows the guidelines for creating a hard-to-detect Trojan, presented in [7]. The paper is structured as follows. Section II provides a background on related work: it gives an overview of the existing design and detection methodologies. Section III describes the typology and general structure of new benchmarks. Section IV deepens the HTs design and provides implementation details together with impact such injection has on power, area and frequency. Finally, Section V concludes the paper.

II. Related Work

A HT is a malicious insertion in a circuit whose purpose is to compromise the security as well as the trustworthiness of the hardware. It may lead to unexpected behaviour. For instance, it may degrade the performance, change the circuit's functionality or even leak secret information. Based on the activation principle, a HT is classified as always-on or trigger-activated [8]. The former gets activated with its host power-on and remains active. The latter is composed of a trigger circuit and a payload circuit. The trigger usually keeps track of signal values, particular states, or events, under some internal or external conditions. Once the trigger condition is satisfied, the payload circuit gets informed and executes the malicious function. The trigger is hard to activate as it is hidden under a set of complex conditions, so the HT is dormant for most of the time and the payload, inactive. In that case, the circuit (functionally) behaves as a Trojan-free circuit. The benchmarks presented in this article belong to the trigger-activated class. Besides, a methodology to detect always-on HTs during the pre-silicon design stage is described in [9].

A. HT Design

Some works have focused on design possibilities and proposed certain methodologies to create new types of HTs. In [7], authors discuss design and implementation of RTL HT to be hard to trigger and able to evade hardware trust verification based on unused circuit identification (UCI) [6]. They rely on specific coding style and trigger input selection. Additionally, signal controllability is examined from the attacker's perspective. In [3], authors explored different implementations of HTs with different combinations of triggers, payloads, as well as unique sections of the architecture that each HT attacks. They were all designed with a varying level of sophistication, allowing the attacker to trade-off design time, ability to evade detection, and payload. They concluded that RTL designs can be quite vulnerable to hardware attacks given the vast insertion space and functional testing can often be useless in detecting them. Apart from introducing a metric for quantifying HT activation and effect, [4] introduces vulnerability analysis flow by determining hard-to-detect areas and provide public trust benchmarks. Some works proposed automatic techniques (malicious CAD tool) for HT insertion. To generate HTs using a highly configurable generation platform, authors in [10] use transition probability to identify the rarely activated internal nodes to target for HT insertion, rather than functional simulation as used in existing platforms. The platform has been tested to generate HT-infected circuits and then evaluated by the ML detection technique [11] - the Controllability and Observability for HT Detection (COTD).

B. Detection techniques

The proposed methods for detecting HTs are intended either as pre-silicon or post-silicon. They include different techniques and combine various paradigms: Machine Learning (ML), verification techniques (both formal and functional), power analysis, using data obtained from static analysis of the model, or dynamic analysis with simulation applying a set of stimuli. Some of the techniques require a golden model. Examples of post-silicon methods are those based on side-channel analysis to measure circuit parameters such as current, operating frequency, power, temperature, radiation. The added circuitry may downgrade device's performance and affect power and delay of wires and gates without fully activating the malicious circuitry. However, if the HT circuit has a small size, the effects on the side-channel parameters could be negligible and the HT could escape the detection. The authors in [12] suggest generating test patterns and combining logic tests with side-channel analysis. Generally, Machine Learning has been successfully applied in the context of HTs. In [13], [14] authors use ML as a side-channel detection methodology, as well as in reverse engineering [15] and circuit feature analysis [16]. Concerning the pre-silicon detection approaches, two main classes can be identified. Techniques in the first one exploit formal methods to prove the existence of malicious hardware [17]. The second class adopts simulation, structural analysis and functional tests generation to excite the suspicious parts of the circuit where HTs can be hidden [18]–[20].

Apart from techniques used during the design cycle for the IPs such as processor core or memory, some of the methods are to be applied run-time (to detect anomalies in communication transfers). A real-time online learning approach for Securing many-core design was proposed in [21] training on hardware feature analysis and HT insertion effects.

III. Hardware Trojans

The proposed benchmarks are intellectual property (IP) level Hardware Trojans conceived for a pipelined Central Processing Unit (CPU). Such Trojans are implanted into an individual IP core of the SoC and can affect only the specific IP in which they are embedded [22]. The benchmarks comply with the taxonomy and the classification scheme outlined in [1],

[1]The presented HT Benchmarks will be made available for the research community and will be uploaded on Trust-Hub platform.

[4], [5]. Furthermore, the following attributes are outlined for each benchmark: abstraction level, insertion phase, location, activation mechanism, trigger, payload, effect. For the sake of completeness, the insertion phase of the HTs is the Design phase, while the abstraction level is the Register-Transfer level for all of the introduced benchmarks. Concerning the effects, the benchmarks might prove to be disastrous or introduce minor damage. Three different categories have been identified:

1) **Degrade Performance (DP)**: The availability of the system under attack might not be affected, remaining fully operational. However, the HT might damage the performance of an IC and, in a worst-case, cause it to fail.
2) **Denial Of Service (DoS)**: The HT when activated stops all the activities of the system.
3) **Change the Functionality (CF)**: The HT alters the functionalities of the system, causing it to perform malicious, unauthorized operations. The CF might also lead to a DP or DoS.

TABLE II
TROJAN BENCHMARKS DESCRIPTION

Name	Location	Trigger	Payload	Cat
OR1K-**T1**00	Decode Unit	Sequence of instructions	Periodically forcing signal values	DP
OR1K-**T2**00	Control Unit	Counters monitoring read accesses to SPRs	Entering the supervisor mode	DoS
OR1K-**T3**00	PIC Unit[2]	Counters for mask and status reg. write access	Disabling external interrupts	CF
OR1K-**T4**00	Control Unit	3 counters for monitoring instructions	Disabling control flag bit	CF
OR1K-**T5**00	Decode Unit	A specific sequence of instructions	Introducing "bubbles" to stall the pipeline	DP
OR1K-**T6**00	Data Cache	Counters monitoring Data Cache Final State Machine (FSM) transitions	Invalidating dcache content	DP
OR1K-**T7**00	Load & Store Unit	Instruction type, order and number	Exception on the data bus	DoS
OR1K-**T8**00	Instr. Cache	Counters monitoring Instr. Cache FSM transitions	Invalidating icache content	DoS

Regarding the trigger part in the introduced Trojan benchmarks, they can be grouped into two main categories. The first category is represented by a sequence of events that, when triggered, enable the payload. Such events can be related to different signals in the model, for instance an exact sequence of instructions, or a set of consecutive values observed on a given bus. There are different possibilities for implementing it; however, two main parts can be identified: a set of conditions that activate or deactivate a targeted flag, and the second one for registering that flag with some auxiliary combinational or sequential logic. Given the complexity of the condition, this type of trigger may be difficult to activate, and therefore may escape to standard verification approaches. The second category of triggers is used to create and check sub-conditions. Once all of them are satisfied, the payload is activated. They can be implemented by monitoring different processor resources, for example, by observing certain values on the bus, the order and/or the number of certain instructions, the read/write access to the registers, or tracking the value of control signals between different stages of the pipeline. Sub-conditions may also check the state of counters in charge of monitoring different activities in the processor. The implemented counters may be the part of a separate process observing the aforementioned activities or be hidden, for instance, in an already existing state machine. In fact, this type of trigger gives the possibility to create a wide-range of complex conditions. A HT would generally be expected to be as much controllable as possible from the attacker's perspective. However, working with a microcontroller, i.e., a System-on-chip (SoC) that integrate additional components such as peripherals, memories, etc. renders such access more difficult. Given that all of the benchmarks are developed for a processor core, and that there are no mechanisms relying on the user input, i.e., component output, such as switches, keyboards or keywords/phrases in the input data stream to activate a Trojan, all of the HTs in our set are considered internally triggered. Moreover, they are activated either depending on the time-based events or on the instructions that are being executed. Table II reports all the essential details related to the newly developed benchmarks: their name, location, trigger and payload brief description and their category.

IV. TROJAN IMPLEMENTATION AND ANALYSIS

The proposed RTL Hardware Trojans are implemented in the mor1kx CPU, whose architecture and HTs' respective faulty location being depicted in Fig. 1. The mor1kx is an open-source core provided by the OpenRISC community; it is a configurable 32/64-bit load and store RISC architecture, written in Verilog Hardware Description Language (HDL).
Due to the high design flexibility, it is possible to customize the core by choosing the best trade-off between area and performance. The version selected in this work (*Cappuccino*) has a pipeline with 4 stages, supports delay slot and is tightly coupled with the caches. It also integrates a Programmable Interrupt Controller (PIC), a Tick Timer (TT) and Debug units. In this work, HTs are injected in the original HDL design, one at a time, by directly modifying the RTL code. On top of 8 primary HT designs, detailed in Table II, we performed modifications concerning the complexity of trigger conditions

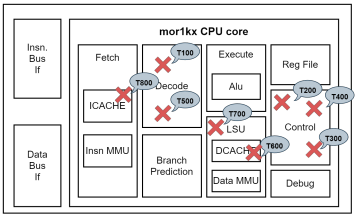

Fig. 1. Proposed RTL Hardware Trojans in the *Cappuccino* configuration of the mor1kx CPU.

```
assign trojan_en = (counter1 == 10665 ) && (counter2 == 12122) && (counter3 == 4323) && (counter4 == 123) && (counter5 == 543) &&
                   (counter6 == 2112) && (counter7 == 5533) && (counter8 == 10101) && (counter9 == 9888) && (counter10 == 3424) &&
                   (counter11 == 7321)? 1 : 0;


always @(posedge clk `OR_ASYNC_RST)
  if (rst)
    trojan_en_r <= 0;
  else if (trojan_en & !trojan_en_r)
    trojan_en_r <= 1;

assign trojan_edge = trojan_en & !trojan_en_r;
```

Fig. 2. Trigger T200 condition

```
always @(posedge clk `OR_ASYNC_RST)
  if (rst) begin
    counter1 = 0;
    counter2 = 0;
    counter3 = 0;
    counter4 = 0;
    counter5 = 0;
    counter6 = 0;
    counter7 = 0;
    counter8 = 0;
    counter9 = 0;
    counter10 = 0;
    counter11 = 0;
  end
  else if (spr_we | spr_read) begin
     if (spr_access[`OR1K_SPR_SYS_BASE]) begin
     if (`SPR_OFFSET(spr_addr)==`SPR_OFFSET(`OR1K_SPR_CPUCFGR_ADDR))
            counter1 = counter1 + 1;
         else if (`SPR_OFFSET(spr_addr)==`SPR_OFFSET(`OR1K_SPR_EPCR0_ADDR))
            counter2 = counter2 + 1;
         else if (`SPR_OFFSET(spr_addr)==`SPR_OFFSET(`OR1K_SPR_SR_ADDR))
            counter3 = counter3 + 1;
     end
     else if (spr_access[`OR1K_SPR_DC_BASE]) begin
    if (`SPR_OFFSET(spr_addr)==`SPR_OFFSET(`OR1K_SPR_DCCR_ADDR))
          counter4 = counter4 + 1;
     end
     else if (spr_access[`OR1K_SPR_PC_BASE]) begin
    if (`SPR_OFFSET(spr_addr)==`SPR_OFFSET(`OR1K_SPR_PCCR0_ADDR))
          counter5 = counter5 + 1;
       else if (`SPR_OFFSET(spr_addr)==`SPR_OFFSET(`OR1K_SPR_PCMR0_ADDR))
          counter6 = counter6 + 1;
     end
     else if (spr_access[`OR1K_SPR_PM_BASE]) begin
        if (`SPR_OFFSET(spr_addr)==`SPR_OFFSET(`OR1K_SPR_PMR_ADDR))
          counter7 = counter7 + 1;
     end
     else if (spr_access[`OR1K_SPR_PIC_BASE]) begin
    if (`SPR_OFFSET(spr_addr)==`SPR_OFFSET(`OR1K_SPR_PICMR_ADDR))
```

Fig. 3. Trigger T200 counters

and coding style to expand our benchmark library and to obtain additional 20 HT designs.

Trojan T100: This Trojan is located in the processor's decode-execute unit (decode to execute signal stage passing). A new process has been added to monitor the instructions being executed. An if-then-else nested structure controls the op-code value originating from the decode unit. Each time an instruction gets decoded, if the sequence is correct, a counter is incremented; if the sequence is interrupted, the counter is reset. The sequence of instructions is ORI-ADDI-AND-ORI-SUB-XOR-AND-XORI-ADD-OR. Once the counter reaches the value 10, i.e., consecutive instructions correspond to the above sequence, payload gets activated. In this case, pipeline is stalled indefinitely, thus disrupting the service.

Trojan T200: This implementation is located in the control unit of the processor. Eleven counters in the newly added process monitor read and write access of special purpose registers (CPUCFGR, EPCR0, SR, DCCR, PCCR0, PCMR0, PMR, PICMR, PICSR, TTMR, TTCR) (Fig. 2). With each access, a corresponding counter is incremented. When all of the counters reach pre-defined values, a trigger is activated (Fig. 3). The payload in this case is integrated into existing code by adding a single OR condition to go from user to supervisor mode. Such behaviour is typical when an exception occurs. The effect is interrupts and timer exceptions being disabled, as well as Data and Instruction MMU. Additionally, a device that is in the supervisor mode enables access to some sensitive registers.

Trojan T300: This HT is located in the programmable interrupt controller. Two counters are inserted to count write accesses to *picmr* (PIC mask) and *picsr* (PIC status) special-purpose supervisor-level registers. Once the trigger part is activated and there are no pending interrupts, payload gets to perform its role by masking all maskable interrupts, which may result in disastrous consequences in safety-critical systems. Reset needs to be performed to unmask such interrupts and disable the HT.

Trojan T400: Malicious trigger-part of this HT consists of three counters counting the number of 3 instructions in the control stage (rfe – return from exception, mfspr – move from special purpose register, mtspr – move to special purpose register). When all three counters count up to a predefined value, the payload is activated. Once activated, the malicious function is designed to prevent the first succeeding setting of the compare-conditional branch flag by adding a simple condition in the assign statement. However, the effect can be severe, given that often a processor when dealing with some instructions uses exactly this flag to calculate the address or/and choose the operand, which may disrupt the desired flow and cause serious problems depending on the application. Once the request for setting the flag arrives Trojan performs its malicious function and gets deactivated. Additionally, a reset signal resets the counters and deactivates the Trojan.

Trojan T500: In the decode to execute unit, a Trojan is implanted to monitor the consecutive instructions. Once the sequence of instructions corresponds to the sequence of 14 pre-defined instructions a trigger is activated. The difference with respect to some of the others HTs introduced in this paper is that this HT introduces two processes for registering the activation signal and producing a pulse. In that manner, the payload gets activated periodically. The payload is added to the condition to form the *decode_bubble_o* signal and insert periodically a bubble into the pipeline. The effect is no change in functionality of the processor. However, due to the stalls it becomes slower, thus, degrading the performance.

Trojan T600: This HT has been inserted into the data cache module. The trigger part consists of 3 counters inserted in the state machine. The Cache FSM has five states: IDLE, WRITE, READ, REFILL, INVALIDATE. The counters have been inserted to count the transitions between the states: IDLE to INVALIDATE, READ to REFILL, WRITE to READ. Once

TABLE III
SYNTHESIS RESULTS

Design	Size					δ**Area**[%]	Power mW				δ**Power**[%]
	Ports	Nets	Cells	Comb./Seq.	Area		Intern.	Switch.	Leak.	Total	
Orig.	9,679	931,538	924,619	601,116 323,252	4,777,062.18	–	257.62	4.93	69.42	331.99	–
$T100$	9,679	931,567	924,648	601,145 323,252	4,777,100.66	0.1×10^{-2}	257.62	4.93	69.42	331.99	-1.81×10^{-4}
$T200$	9.679	932,716	925,797	602,038 323,508	4,781,729.10	9.8×10^{-2}	257.82	4.94	69.48	332.24	7.60×10^{-2}
$T300$	9,679	931,899	924,980	601,412 323,317	4,778,437.10	2.9×10^{-2}	257.67	4.93	69.44	332.06	2.03×10^{-2}
$T400$	9,679	932,033	925,114	601,515 323,348	4,779,015.82	4.1×10^{-2}	257.70	4.93	69.45	332.09	3.07×10^{-2}
$T500$	9,679	931,793	924,874	601,333 323,290	4,777,998.70	2.0×10^{-2}	257.65	4.93	69.43	332.03	1.35×10^{-2}
$T600$	9,679	932,056	925,137	601,535 323,351	4,779,066.78	4.2×10^{-2}	257.69	4.93	69.43	332.06	2.11×10^{-2}
$T700$	9,679	931,787	924,867	601,330 323,286	4,777,932.66	1.8×10^{-2}	257.65	4.93	69.43	332.03	1.16×10^{-2}
$T800$	9,697	932,052	925,043	601,441 323,351	4,779,032.98	4.1×10^{-2}	257.70	4.94	69.45	332.09	3.18×10^{-2}

all of the three counters reach certain values, cache invalidation is forced.

Trojan T700: This HT is located in the load-store unit. Trigger part consists of nested if-else examining the sequence of consecutive multiple load i.e., store operations with 3 different types of access: byte (8), half-word (16) and word (32). Once the complex condition gets satisfied, a pulse signal is generated to activate the payload. The payload in this case is integrated into the process dealing with the data bus exceptions. In this regard, once the payload becomes activated, it will execute its malicious function by simulating a data bus exception and stepping into the exception routine. As a result, the processor proceeds to the next instruction in the pipeline skipping the current one at the moment when the exception occurred. Such event may definitely disrupt the normal operation of the processor.

Trojan T800: This HT is implanted into the instruction cache unit. Its trigger part is incorporated within the FSM with counters following FSM state transitions. Once all the counters get set to predefined values, a payload is activated: the internal hit signal is tied to zero, therefore, every time a request is sent, the instruction cache reports a miss, i.e., not found in cache memory. Consequently, a refill operation is performed, thus significantly slowing down processor's performance.

To demonstrate the feasibility of performing the proposed modifications and inserting malicious code, we synthesized all of our 8 HT designs, including the original one, with a 65nm industrial technology. Successively, we collected reports regarding area, power and frequency. The results given in Table III clearly show that such insertions are negligible in terms of area and power overhead. The relative area difference is below 9.8×10^{-4}, while the total power relative difference is below 7.6×10^{-4}. Furthermore, we have confirmed that the critical path in the design does not change by introducing the proposed modifications.

Starting from the structure of these original 8 HTs, 20 additional benchmarks have been derived by making changes mainly on the trigger part (complexity of trigger conditions, changing the comparison values, and changing them structurally). For instance, if the trigger looks for a particular instructions sequence, this has been shortened or extended. Additional wire signals for controlling the conditions are introduced, and the position and number of counters is changed together with comparison values. Furthermore, if the trigger sequence was hosted in a single RTL process, it has been split up to use two or more processes, clearly maintaining the same sequence. For example, Trojan T200, originally uses the value of 11 counters to control the trigger condition. A modified version of this Trojan uses 14 counters for its activation. Their values are incremented within two separate processes (10 + 4). The aforementioned changes are especially useful for evading detection by some methodologies that rely on one particular coding style. On the whole, the benchmark set finally contains a total of 28 HT.

Functional testing is quite unlikely to detect malicious circuitry based on instruction or access sequences as the input space is too large. The number of instructions in 32-bit version of the processor is 96 (including custom ones). Therefore, the probability of activating Trojan T100 is 10×10^{-20} order of magnitude. Moreover, functional verification/testing is statistically useless trying to detect HTs observing multiple counter values. It is not only because of the large number of conditions but also given the large comparison values and limited time required to run the simulations. All of the listed HTs can get excited and are not completely dormant/silent in terms of activity. Nevertheless, the probability of activating the payload is extremely low without the knowledge of HT's structure inserted by the attacker. UCI detection technique has certain limitations. UCI can be avoided by inserting malicious circuits that affect unchecked outputs. Unchecked outputs could arise from incomplete test cases or from unspecified output states. Additionally, an attacker might exploit implementation-specific behavior and hide a HT in a module such as cache. Such affected outputs might be difficult for a testing program to check deterministically, thus causing malicious circuits to affect outputs and avoid UCI analysis.

V. Conclusions and Future Works

In this paper we proposed a set of 8 new principle HTs and their 20 modifications for a pipelined processor core. The proposed HTs have been injected in very different parts of the processor design. They differ in the trigger and payload. The synthesis reports show the negligible impact that the introduced modifications have on area, power and frequency. We believe that the set of benchmarks could be extremely useful for validating dynamic HT detection methodologies since the core is open-source and in the near future the HTs will be also publicly available. The proposed set of HTs are easily modifiable and allow to create even more complex set of trigger conditions, while the space for inserting payloads is quite vast and allows to execute different type of malicious functions. That is why the future work will be focused on diversifying and developing the HT Benchmarks Library even further. Although most of the detection techniques work at the gate level, shifting the detection of HTs inserted at RTL to the gate level would result in increased design and verification costs. Therefore, our future work will also focus on developing a new, fast and efficient method for detecting such Trojans, based on both static and dynamic analyses of the circuit.

Acknowledgements

The work has been partially supported by the European Commission through the Horizon 2020 RESCUE-ITN project under the agreement No. 722325.

References

[1] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware trojans: Lessons learned after one decade of research," *ACM Trans. Des. Autom. Electron. Syst.*, 2016.

[2] S. King *et al.*, "Designing and implementing malicious hardware.," Jan. 2008.

[3] Y. Jin, N. Kupp, and Y. Makris, "Experiences in hardware trojan design and implementation," in *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, 2009, pp. 50–57.

[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 *et al.*, "Benchmarking of hardware trojans and maliciously affected circuits," *Journal of Hardware and Systems Security*, 2017.

[6] M. Hicks *et al.*, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 159–172.

[7] J. Zhang and Q. Xu, "On hardware trojan design and implementation at register-transfer level," 2013.

[8] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *Design & Test of Computers, IEEE*, 2010.

[9] A. Ruospo and E. Sanchez, "On the detection of always-on hardware trojans supported by a pre-silicon verification methodology," in *2019 20th International Workshop on Microprocessor/SoC Test, Security and Verification (MTV)*, 2019, pp. 25–30. DOI: 10.1109/MTV48867.2019.00013.

[10] S. Yu, W. Liu, and M. O'Neill, "An improved automatic hardware trojan generation platform," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2019, pp. 302–307. DOI: 10.1109/ISVLSI.2019.00062.

[11] H. Salmani, "Cotd: Reference-free hardware trojan detection and recovery based on controllability and observability in gate-level netlist," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 2, pp. 338–350, 2017. DOI: 10.1109/TIFS.2016.2613842.

[12] S. Bhunia, M. Hsiao, M. Banga, and S. Narasimhan, "Hardware trojan attacks: Threat analysis and countermeasures," Aug. 2014, pp. 1229–1247.

[13] S. Wang *et al.*, "Hardware trojan detection based on elm neural network," in *2016 First IEEE International Conference on Computer Communication and the Internet (ICCCI)*, 2016, pp. 400–403.

[14] 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*, 2017.

[15] C. Bao, D. Forte, and A. Srivastava, "On application of one-class svm to reverse engineering-based hardware trojan detection," in *Fifteenth International Symposium on Quality Electronic Design*, 2014, pp. 47–54.

[16] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Trojan-feature extraction at gate-level netlists and its application to hardware-trojan detection using random forest classifier," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.

[17] M. Rathmair, F. Schupfer, and C. Krieg, "Applied formal methods for hardware trojan detection," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 169–172.

[18] X. Zhang and M. Tehranipoor, "Case study: Detecting hardware trojans in third-party digital ip cores," in *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*.

[19] A. Waksman, M. Suozzo, and S. Sethumadhavan, "Fanci: Identification of stealthy malicious logic using boolean functional analysis," 2013.

[20] 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.

[21] A. Kulkarni, Y. Pino, and T. Mohsenin, "Adaptive real-time trojan detection framework through machine learning," in *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2016.

[22] R. Elnaggar and K. Chakrabarty, "Machine learning for hardware security: Opportunities and risks," *Journal of Electronic Testing*, 2018.