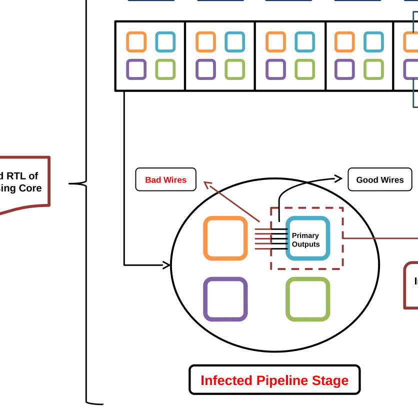

d RTL of
ing Core
Bad Wires
Good Wires
Primary
Outputs
In
Infected Pipeline Stage

low on area, power, and delay, that prove the resiliency of our attack to side-channel attack detection techniques.

Metric	Healthy Processor	Processor Under BP's Attack		Processor Under I-Cache's Attack	
		Magnitude	Overhead	Magnitude	Overhead
Area (μm^2)	127370.326453	127388.672052	+0.0144%	127552.018451	+0.1426%
Power (mW)	396.561916	396.575816	+0.0035%	396.588516	+0.0067%
Delay (ns)	48.5764	48.5726	-0.0078%	48.5770	+0.0012%

Table I: The Footprint of Designed Trojans

B. Analysis of Performance Degradation

In this part, we show the performance degradation and ruinous effects caused by designed hardware Trojans on a processor. Regarding this issue, an out-of-order processor – which has a 32 KB/2-Way instruction cache and a branch target buffer with 4K entries – is used in GEM5 architectural simulator. Therefore, by running ten SPEC2006 benchmarks for ten million instructions in Syscall Emulation mode on the healthy processor and the infected processors, the required results for analysis are achieved.

Based on the collected data, there is a significant reduction in instruction per cycles (IPCs) for the ten benchmarks, that are approximately in range 23 % - 84 % and 0.1 % - 90 % for both the processor under BP's attack and the processor under I-Cache's attack – as it is shown in Figure 2.

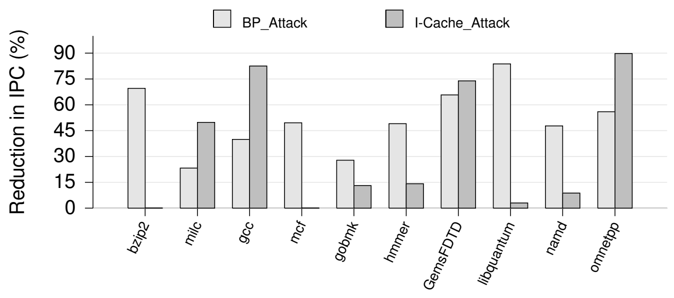


Figure 2: Comparison between the IPCs of Infected BP and Infected I-Cache

The Trojan that is injected in the output's interface of branch prediction causes a considerable increase in the number of incorrectly predicted branches for all of the ten benchmarks – shown in Figure 3, which is in range 2X - 506X.

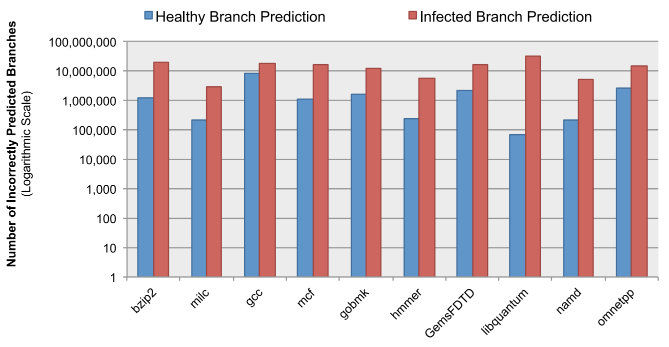


Figure 3: Comparison between the prediction behaviors

As it can be seen from Figure 4, the infected instruction cache suffers from a substantial miss rate growth which is enclosed in range 8X - 454X. Suprisingly, the miss rates of three benchmarks namely bzip2, mcf, and gobmk are not under the Trojan's payload. This issue might be a serious problem if an IP consumer relies on standard benchmarks for performance evaluation of a received processing core. Meanwhile, a malicious IP vendor could design a hardware Trojan – belongs to the aforementioned class of attacks – in a way to be resilient to standard benchmarks.

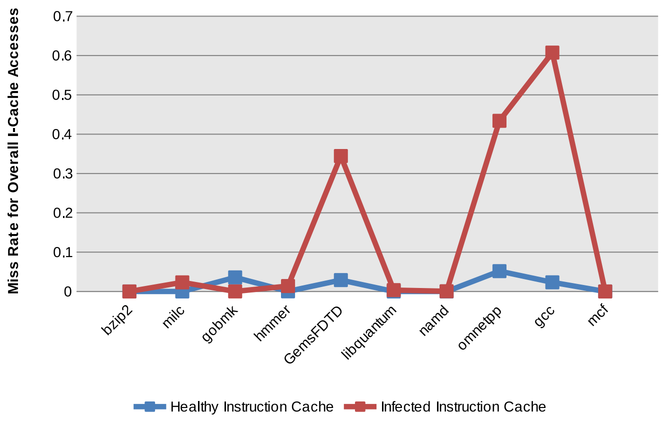


Figure 4: Comparison between the miss rates of the processors

C. Detectability of The Attack

While the previous parts demonstrate the attack's potency, it is critical to investigate whether it can be manifested by existing detection techniques. For this goal, we have implemented FANCI (Functional Analysis for Nearly-unused Circuit Identification) tool – a recently proposed technique by A. Waksman et al. [8] – which tries to reveal an attack by analyzing all of the circuit's wires based on determination of the effect's strength of input wires on output wires (considering both endogenous and exogenous input or output wires, not merely primary inputs or primary outputs). Meanwhile, in order to analyze the effect's strength of related input wires for each output wire and determines whether the wire is suspicious, FANCI uses four different methods namely Average, Median, Hybrid (consists of both average and median), and Triviality.

To evaluate the detection ability of FANCI and also perusing the furtive temperament of the hardware Trojans, the infected netlists of branch prediction and instruction cache are examined via it. According to our analysis on the outcomes of FANCI, the I-Cache's attack is detectable by applying a simple tracing method (or even manually), because of having the Miss wire – which comes from the output's interface – in the suspicious wires list. However, due to the lack of existence of the Prediction wires in the list, an advanced tracing algorithm is required to detect the Trojan. The number of suspicious wires for Fetch stage under BP's attack and I-Cache's attack are shown in Figure 5.

The FANCI tool might be really precarious to an infected netlist. The reason of this statement is that if a malicious IP vendor implements the circuit of a Trojan (or some critical zone(s) of it) by primitive two-input gates, then based on the configuration of FANCI most of the truly listed suspicious wires are eliminated and also the total number of circuit's

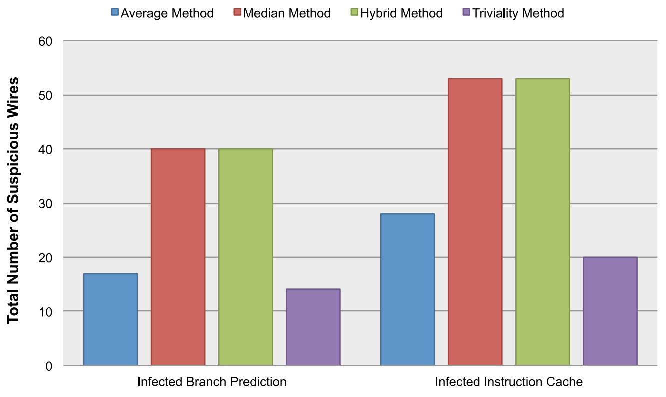


Figure 5: Suspicious wires of the circuits based on the four analysis methods

wires is increased, that result in leaving the Trojan undetected. According to Figure 6 which shows the point of this issue, the final control value of the component is 0.325 based on the average method; however, when it is implemented by primitive two-input gates, the final control value would be 0.5, which makes the OUT wire to be considered as a trustworthy wire – because it is a high amount for the control value of a suspicious wire. Meanwhile, the total number of output wires would become six times of the regular status, which makes the process of tracing more challenging.

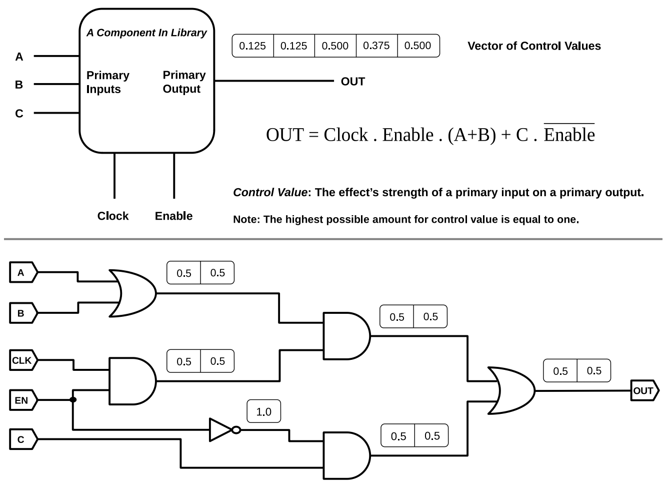

A Component In Library
A
B
C
Primary Inputs
Primary Output
0.125
0.125
0.500
0.375
0.500
Vector of Control Values
OUT
OUT = Clock . Enable . (A+B) + C . $\overline{\text{Enable}}$
Control Value: The effect's strength of a primary input on a primary output.
Note: The highest possible amount for control value is equal to one.
Clock
Enable
A
B
CLK
EN
C
0.5
0.5
1.0
OUT

Figure 6: FANCI Weakness Description

III. A New HTH Manifestation Technique

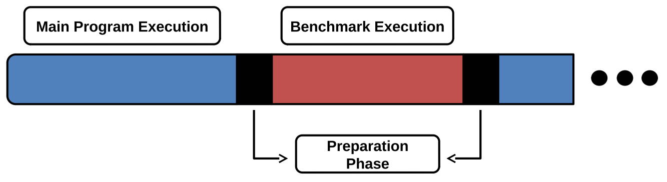
The best way to face every attack is analyzing its purpose and obligations to design a countermeasure according to them, because the attack can be recessive to other confronting techniques. As an example, since our proposed attack does not target functionality and has a cover nature, it cannot be detected logically using functionality based or wire analysis based techniques – which was demonstrated according to our analysis on FANCI's results. In this regard, we decided to create an HTH detection technique to security check a processing core based on evaluating its performance.

A. Inclusion of HTH Detection Module in SoC

According to our technique, a module for analyzing processing core(s) of an SoC is included in the chip, which consists of: (a) Purposeful benchmarks for investigating the healthiness of the processor; (b) A memory unit for maintaining the outcomes of the golden model; and (c) A unit for evaluating the performance results of the processor-under-test (PUT) and determining the existence of a possible Trojan. The module can be configured in a way to be activated whether in run-time or in test mode of the chip.

B. Description of Run-Time and Test Mode Configurations

In run-time configuration, after a certain number of instructions of the main program were executed, the whole or a portion of a purposeful benchmark is run on the processing core in order to evaluate the performance of the processor and detect a possible hardware Trojan basen on that. Meanwhile, the processor needs to be flushed and the content of instruction cache and branch history table require to be cleared at the start of each execution phase – whether it is for main program or it is for purposeful benchmark. The major advantage of this configuration over the other one is its reliability in detecting sequential hardware Trojans, because of having multiple phases for performance evaluation. However, it causes latency overhead in addition to area and power overhead. The configuration is explained abstractively in Figure 7.



***Preparation Phase*: It is for flushing pipeline and clearing the content of instruction cache and branch history table.**

Figure 7: Run-Time Configuration

With configuring the module for test mode, a designed benchmark is simply run on the PUT and its performance is evaluated. The only overhead of this configuration is in area of the chip; but its reliability and efficiency is possibly lower than the run-time configuration.

C. Design of Purposeful Benchmark

A purposeful benchmark is designed by an SoC designer to be embedded in a chip for testing or verifying all or some zone(s) of the processor's architecture. It can be data processing based and/or instruction executing based according to the SoC designer's objectives, and can be written in any high-level or low-level programming language to strengthen the process of performance evaluation. The HTH detection module can be threated if a malicious person in a factory finds and manipulates the benchmarks of the chip for the purpose of changing the module's decision, which leads to leave the Trojan undetected. To tackle this possible problem, the functionality of the processing core needs to be checked frequently and assessed pursuent to the correct functionality of different phases of a running benchmark. Finally, in order to quest the detectability of our designed attack and efficiency of proposed HTH detection module, the architectural behavior of instruction cache and branch prediction are taken into account to develop our purposeful benchmarks.

IV. Experimental Methodology

In this section, the infrastructure of required experiments for analysis of the attack's strength and evaluation of the efficiency and reliability of our manifestation technique is presented.

A. Injection of Hardware Trojans

We have employed one of the generated cores (Core-1) of FabScalar tool in order to transfuse our designed hardware Trojans to consequently achieve the footprint of them on main processor characteristics and also provide the essential input (netlist) for FANCI tool. Due to the fact that the processing core was written in Verilog HDL, we have implemented the hardware Trojans using the same language in FetchStage1 component of the processor. The healthy processor along with the infected processors have been synthesized by Synopsys Design Compiler using TSMC 45 nm technology library.

B. FANCI Tool Implementation

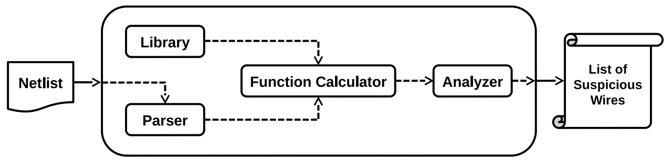


Figure 8: FANCI Tool Implementation

Our implementation of FANCI tool consists of four major parts namely Library, Parser, Function Calculator, and Analyzer – shown in Figure 8 – and has been performed using C++ programming language. Library part defines and implements the functionality of needed gates and components. Parser is for parsing the netlist and building the required containers for all of the gates/components and wires of the design. Calculation of the value of an output wire (whether endogenous or exogenous) based on its functionality and also construction of its vector of control values are accomplished by Function Calculator. Finally, Analyzer scrutinizes the vector of control values of a wire based on the specified threshold value (which is for control values) using the chosen analysis method to determine whether it is suspicious or not.

C. Architectural Simulation

According to the architectural behavior of branch prediction and instruction cache, two hardware Trojans – similar to the designed ones at RTL – have been injected in respective components of an out-of-order processor in GEM5 architectural simulator. Also, we have developed three purposeful benchmarks namely IE (if-else statement based), FC (function call based), and Crossbred using C++ programming language. IE benchmark is a program, which includes many if-else statements and arithmetic operations. To build FC benchmark, several user-defined functions and built-in functions are recruited and finally by implementing myriad nested if-else statements and employing various user-defined functions and built-in functions, Crossbred benchmark is constructed. By injection of hardware Trojans and creation of required benchmarks, the architectural simulations are performed on the healthy processor and the infected processors with considering the preparation phases of run-time configuration using GEM5 simulator.

V. Evaluation

UNDER CONSTRUCTION

VI. Related Work

Catastrophic effects of morbid intellectual property (IP) inclusion in SoC design has motivated researches in the world to decipher and analyze different varieties of attacks of this scenario and also incited them to discover and provide effective and applicable countermeasures for them. Unfortunately, this malicious scenario wasn't focused properly by researchers in compare to others and consistent solutions can hardly be found for it.

Two different sets of Trojans were presented in [9] and [11] regarding the aforementioned attacks, but the authors did not provide any solution for detecting or correcting them. In [3], an approach for detecting hardware Trojans in processor using Processor Protection Unit (PPU) was proposed, which can distinguish the correct behavior of the processor from the incorrect one. However, due to the creation of advanced HTHs frequently and ever increase in demand for having a system with appropriate security level, their technique requires significant area and consumes more power in compare to the initial design.

Emulation of Anti-Virus and Anti-Trojan softwares at hardware level has opened a new area for detection of the attacks [2] and [5]. Although, with aggressive formation of both software and hardware attacks and the possibility of finding the correspondences of software bugs in hardware designs, these techniques might be tottery. The authors in [7] and [4], presented a new procedure for increasing the security level of designed circuit by implementing Dynamic Information Flow Tracking at gate level (GLIFT). As regards the IP vendors are reluctant to release the proprietary information of their products and IP consumers may not access to them easily, GLIFT might not be applicable for detection of these attacks.

S. Millican et al. proposed a method for performing logic simulation and fault simulation on a rendered IP core by its consumer without need to know the detail of its implementation [6]. Although, the description of their work is inadequate and there exist fundamental questions regarding the implementation of logic simulator and fault simulator by the consumer without knowing the encryption scheme. Also, it can be beneficial if they specify the applicability of their method on complex designs with significant number of gates and components. Meanwhile, a survey has been done on Trust-Hub benchmarks by our group, which shows none of the provided benchmarks is about injection of hardware Trojans in processors at architectural level. However, there exist some benchmarks related to injection of HTH in micro-controller for manipulating and altering the functionality and behavior of the chip; but they can be detected by existing reliable RTL verification techniques. Therefore, with lack of existence a consistent countermeasure for analyzing and detecting the

	Healthy Processor			Infected Processor		
Parameter	IE-Bench.	FC-Bench.	Crossbred-Bench.	IE-Bench.	FC-Bench.	Crossbred-Bench.
Instruction Per Cycle	1.197297	1.143638	0.799871	1.017181	0.861789	0.693044
Number of Cycles	84,757,419	138,796,677	159,044,131	99,765,722	184,190,221	183,559,438
Miss Rate for I-Cache Accesses	0.000032	0.000117	0.006842	0.008640	0.009150	0.018530
Miss Rate for L2 Cache Accesses	0.967376	0.286161	0.005044	0.971081	0.973315	0.493721

Table II: Comparison between Healthy Processor and Infected Processor under I-Cache's Destructive Wrapper

	Healthy Processor			Infected Processor		
Parameter	IE-Bench.	FC-Bench.	Crossbred-Bench.	IE-Bench.	FC-Bench.	Crossbred-Bench.
Instruction Per Cycle	1.197297	1.143638	0.799871	0.844236	0.750891	0.652756
Number of Cycles	84,757,419	138,796,677	159,044,131	120,203,129	211,393,023	194,888,591
Number of Fetched Branches Per Cycle	0.470021	0.206094	0.251049	0.443065	0.195381	0.253393
Number of Predicted Branches	39,837,808	28,605,193	39,927,841	53,257,789	41,302,098	49,383,490
Number of Incorrectly Predicted Branches	2,066,069	2,399,860	3,275,599	4,437,737	7,118,909	5,577,104
Number of Executed Branches	33,575,925	23,739,060	32,485,284	36,446,658	29,512,920	34,531,657

Table III: Comparison between Healthy Processor and Infected Processor under BP's Destructive Wrapper

aforesaid attacks, our proposed purposeful benchmark creation technique is able to confront them with high degree of trustiness.

VII. Conclusion

UNDER CONSTRUCTION

References

[1] Chakraborty, R. S. and others MERO: A Statistical Approach for Hardware Trojan Detection. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems* (2009), pp. 396–410.

[2] Demme, J. and others On the feasibility of online malware detection with performance counters. In *ISCA* (2013), pp. 559–570.

[3] Dubeuf, J. and others Run-time detection of hardware Trojans: The processor protection unit. In *IEEE European Test Symposium (ETS)* (2013), pp. 1–6.

[4] Hu, W. and others Theoretical Fundamentals of Gate Level Information Flow Tracking. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2011), pp. 1128–1140.

[5] Jin, Y. and others Cycle-Accurate Information Assurance by Proof-Carrying Based Signal Sensitivity Tracing. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)* (2013), pp. 99–106.

[6] Millican, S. and others CryptIP: An Approach for Encrypting Intellectual Property Cores with Simulation Capabilities. In *27th International Conference on VLSI Design and 13th International Conference on Embedded Systems* (2014), pp. 92–97.

[7] Tiwari, M. and others Complete Information Flow Tracking from the Gates Up. In *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems* (2009), pp. 109–120.

[8] Waksman, A. and others FANCI: identification of stealthy malicious logic using boolean functional analysis. In *ACM Conference on Computer and Communications Security* (2013), pp. 697–708.

[9] Wang, X. and others Software exploitable hardware Trojans in embedded processor. In *Proc. of DFT* (2012), pp. 55–58.

[10] Wolff, F. and others Towards Trojan-Free Trusted ICs: Problem Analysis and Detection Scheme. In *Design, Automation and Test in Europe (DATE)* (2008), pp. 1362–1365.

[11] Zhen, L. and others Designing and Implementing Hardware Trojans in ARM9 IP. In *International Conference on Communications, Circuits and Systems (ICCCAS)* (2013), pp. 524–527.