electronics

Article

A Gate-Level Information Leakage Detection Framework of Sequential Circuit Using Z3

Qizhi Zhang [1], Liang Liu [2], Yidong Yuan [2], Zhe Zhang [2], Jiaji He [1,*], Ya Gao [1], Yao Li [1], Xiaolong Guo [3] and Yiqiang Zhao [1,*]

[1] School of Microelectronics, Tianjin University, Tianjin 300072, China
[2] Beijing Smart-Chip Microelectronics Technology Co., Ltd., Beijing 100192, China
[3] Department of Electrical and Computer Engineering, Kansas State University, Manhattan, KS 66506, USA
* Correspondence: dochejj@tju.edu.cn (J.H.); yq_zhao@tju.edu.cn (Y.Z.)

Abstract: Hardware intellectual property (IP) cores from untrusted vendors are widely used, raising security concerns for system designers. Although formal methods provide powerful solutions for detecting malicious behaviors in hardware, the participation of manual work prevents the methods from reaching practical applications. For example, Information Flow Tracking (IFT) represents a powerful approach to preventing leakage of sensitive information. However, existing IFT solutions either introduce hardware overheads or lack practical automatic working procedures, especially for hardware sequential logic. To alleviate these challenges, we propose a framework that fully automates information leakage detection at the gate level of hardware. This framework introduces Z3, an SMT solver, to automatically check the violation of confidentiality. On the other hand, an automatic tool is developed to remove the manual workload further. In this tool, the gate level hardware is converted to the formal model firstly, and the integrity of the model is assessed. Along with the model converting step, the property for leakage detection is generated as well. The proposed solution is tested on 25 gate-level netlist benchmarks, where sequential designs are included to validate the effectiveness. As a result, Trojans leaking information from circuit outputs can be automatically detected. The measured time consumption of the entire working procedure validates the efficiency of the proposed approach.

Keywords: information flow tracking; formal verification; information leakage detection; Z3

Citation: Zhang, Q.; Liu, L.; Yuan, Y.; Zhang, Z.; He, J.; Gao, Y.; Li, Y.; Guo, X.; Zhao, Y. A Gate-Level Information Leakage Detection Framework of Sequential Circuit Using Z3. *Electronics* **2022**, *11*, 4216. https://doi.org/10.3390/electronics11244216

Academic Editor: Yue Wu

Received: 9 November 2022
Accepted: 12 December 2022
Published: 16 December 2022

1. Introduction

The demand for intellectual property (IP) cores has significantly increased owing the changing landscape of the semiconductor industry. The proliferation of the IP market is affected by various factors such as lowered design cost, shortened time-to-market (TTM), etc. In the meantime, the credibility of third-party vendors is threatened by the hardware Trojan and design flaws, which also places high-security uncertainties on the IP end-users and customers. In a system-on-chip (SoC), a malicious IP core can bypass many existing hardware Trojan detection methods [1,2].

In detecting hardware Trojans and vulnerabilities, formal methods have been most effective among all the existing techniques [3–11]. However, very few current formal verification approaches are scalable and practical for hardware Trojan detection in the industry due to the lack of automatic and efficient tools. For instance, model checking is a popularly used malicious logic detection method for protecting third-party IP cores [10]. In the model checking, security properties are formalized as traces, and all possible traces generated by the system are checked. The system is said to satisfy the security property if all the traces pass the checking [12]. However, checking the very large system, the model checker always runs into the state space explosion issue.

Information flow tracking (IFT) [13] is a scalable approach for detecting leakage/sneaky path of sensitive information. In IFT, data or operations are assigned by labels standing

for the trust levels. The labels are propagated or updated to other data relying on the information flow policy. In general, data with labels are accessed or propagated to the trust portion in the system. Some IFT-based solutions on assuring hardware security are proposed such as SecVerilog [14,15], Caisson [16], Sapper [17], QIF-Verilog [18], CELLIFT [19], gate-level information flow tracking (GLIFT) [20], etc. However, there is a lack of IFT solutions in detecting sneaky paths in the gate-level net-list. Theorem provers such as Coq are utilized in proving the gate-level information flow property [21]. However, as a theorem proving method, a significant manual effort is required for constructing machine proofs. SecChisel [22] applies an automated formal verification checking using the Z3 solver [23], but it only provides protections during high-level synthesis.

We propose a framework for formalizing and checking gate-level hardware design for security purposes to solve those problems. In the framework, gate-level net-list data files are parsed to the formal model in the form of constraints. Then sensitive labels are introduced to denote secrets in the hardware design. If outputs are tainted by the labels, the information leakage is detected. Satisfiability modulo theories (SMT) solver is utilized as the checking engine to propagate the information flow and automatically check IFT policies.

The main contributions of this paper are as follows.

- We introduce an automated formal verification framework detecting vulnerabilities in the gate-level net-list. The net-list data is formalized into a circuit model, and then security properties are designed based on the model. Confidentiality is enforced on the input hardware design by applying an automatic checking engine.
- GLIFT is, for the first time, statically applied in the gate-level hardware with a fully automated working procedure. The information leakage is addressed and localized by tracking sensitive information.
- An automatic tool, including a parser and a SMT Solver, is designed and demonstrated based on the developed framework. The parser for translating net-list files to formal models can support multiple net-list process libraries such as generic gate Verilog and 180nm CMOS library. An algorithm is developed to support analyzing the properties in sequential circuits using an SMT solver.
- The proposed framework can accurately analyze the sequential information flow, and its effectiveness is proven in 25 benchmarks, including combinational circuits and sequential circuits.

The rest of the paper is organized as follows. In Section 2, we introduce the threat model and discuss previous work on malicious logic detection using IFT based solutions and then present a gate-level IFT model. We explain our automated framework involving the SMT solver and code parser in Section 3. Section 4 introduces the tool we designed for the automatic framework. Section 5 presents demonstrations of our approach. The limitations of this work are discussed in Section 6. Finally, conclusions are drawn in Section 7.

2. Background

2.1. Attack Model

This paper assumes that information leakage paths are created by either intended hardware Trojan or unintentional design errors. An adversary can insert malicious logic at the design or testing stage in the supply chain. We assume that the rogue agent at the third-party IP vendor can access the Register-transfer Level (RTL) design or the gate-level netlist files and then insert a hardware Trojan or backdoor to create a sneaky path in the design. Lacking security knowledge, the hardware developers could produce vulnerabilities, such as leakage paths, in the design stage. On the other hand, we assume that attackers can access inputs and outputs ports of the manufactured hardware and have knowledge of the hardware functionality. Therefore, by triggering the Trojan or observing the input-output patterns, the attacker can exploit such information leakage paths to infer the sensitive/secret information of the design.

2.2. Related Work

Formal methods have been proven that it is the most effective and complete method in ensuring the security of integrated circuits. However, because of the lack of automated and security-oriented tools, very few formal methods are applied to hardware security verification. Moreover, some of these works are manual and dynamic, making the verification process time-consuming. The related works are summarized in Table 1.

Table 1. Summary of formal methods in hardware security.

Work	Target	Static or Dynamic	Automatic
[24]	Gate level HW design	Dynamic	Y
[25]	RTL HW design	Dynamic	Y
[26]	Gate level HW design	Dynamic	Y
[27]	RTL HW design	Dynamic	N
[28]	Gate level HW design	Dynamic	Y
[29]	HW/SW interactions	Dynamic	Y
This work	Gate level HW design	Static	Y

Recently, IFT based security approaches for protecting confidentiality are delivered in the form of a language-based solution. Caisson [16] and Sapper [17] realize IFT isolation and separation properties and in the synthesized secure circuits. In Caisson or Sapper, wires and registers are duplicated in generated hardware, which introduce considerable hardware overheads at the circuit level. SecVerilog avoids the hardware overheads by detecting information leakage in the compilation stage [15]. It extends the type system of standard Verilog to enforce noninterference in the design. However, a complex security label system is needed by SecVerilog to increase precision. Only with sufficient knowledge of security, the circuit designers can specify information flow policies in SecVerilog. In contrast, QIF-Verilog only extends one simple security label from the standard Verilog to reduce the cost of learning from the developers' side [18]. It quantifies the information leakage by applying the quantitative information flow tracking in the design stage. However, the QIF-Verilog is not capable of supporting IFT analysis in the gate-level netlist. CELLIFT [30] provides a dynamic information flow tracking method for hardware. It leverages the logical macrocell abstraction to achieve scalability, precision and completeness in RTL design. However, its performance is limited by the amount of logic cell types.

In [31], GLIFT is proposed to detect malicious logic by tracking the information flow in the runtime hardware. It models logic gates and labels individual bit at the gate-level. The information flow propagation logic is realized in hardware along with the original functional circuit, though with high hardware overheads [19]. A static GLIFT approach is proposed in [21] which checks security property in the gate-level netlist. It translates the property and the netlist to theorems and formal circuits, respectively. The theorem proving is utilized to prove the satisfaction of the property against the formal circuit. Using an interactive proving approach, developers manually construct the proofs, which increases the time required for certifying large hardware design. SecChisel is proposed in [22] to check the confidentiality and integrity of hardware design automatically using the SMT solver. Based on the Chisel hardware construction language, the SecChisel verification framework converts a higher level hardware description to the intermediate representations, FIRRTL representations, and then parses them to Z3 inputs for the information flow checking. Although the framework checks the IFT property automatically, it focuses on the high-level synthesis procedure rather than the gate-level netlist, not to mention that Chisel has not been widely adopted in industry. Refs. [24,32] propose a unified formal model which combines IFT Taint-propagation and X-propagation to verify the security and integrity of the hardware design. This work realize efficient model building for multiple property verification. However, it causes a large simulation overhead because of the extra tracking logic in RTL code.

In our preliminary work [33], GLIFT approach is used to detect information leakage combined with an SMT solver Z3. It translates the original circuit and extra IFT logic from the net-list file to a static formal model. The property is designed based on the privacy of information propagation. Specifically, the security labels of sensitive input and output ports are set high to evaluate whether the sensitive information can be propagated to output. Then the model and property are input into the Z3 SMT solver for tracking information flow. This work realized an automatic framework for GLIFT model translating, property generating, and leakage path solving. However, the framework can only handle the combinational circuit. There is a demand for supporting sequential logic as the design of hardware becomes more and more complex.

2.3. Modeling Gate-Level IFT

An advantage of GLIFT is that each data bit is associated with a security label, which propagates labels more precise and reduces false-positive rates [20]. As an example, in Equation (1), we perform *and* operation between the secret signal and a 32-bits zero vector, then output the result.

$$Output := \text{AND-2}(Secret, 0x00000000) \tag{1}$$

where **AND-2** function performs as a 32 bits two-inputs AND operation and *Secret* has been labelled as high sensitive. In the traditional IFT approach, the sensitive label would be propagated to the output port and then detected as information leakage. However, as the other signal involved in the *and* operation is zero, no secret is actually leaked through this operation, which causes a false-positive.

In the GLIFT, both signal value and security labels are taken into consideration during the label propagation. Rather than tracking the data flow in the original design only, how the output is influenced by input values must also be accounted for. To achieve this goal, extra logic gates are created to represent the influence along with the original circuit. We use a two-input AND logic gate as the example. For each two-input AND gate, the extra logic gates are inserted as shown in Figure 1. The A and B are 1-bit input while O is the 1-bit output. Accordingly, labels for A, B and O are denoted as A_t, B_t and O_t. Following the structure, once the low sensitive input is 0, the output label O_t keeps 0 no matter what the other high sensitive input value is. Only in the case that the low sensitive input is 1, the O_t is influenced by the high sensitive input, which means that the highly sensitive label has the potential to propagate to the output.

For sequential circuits, when the clock edge comes, the signal in circuit can be propagated through sequential logic gate. The sensitive label should also be propagated at the meanwhile. However, previous sequential information flow researches don't always obey the theory elaborated above. Take D-flip-flop as an example, Figure 2 shows the logic and sensitive propagation rule of DFF cell in previous research. When the clock positive edge comes, the signal of port D propagates to port Q. The sensitive label D_t is propagated to port Q_t without any sequential constraint. There is no problem to apply this DFF model in dynamic information flow tracking methods. Because in dynamic information flow tracking methods, extra circuits for IFT logic are added. The circuit implementation of IFT logic guarantees the correctness of information propagation. However, in static information flow tracking method, static model of IFT logic is generated directly. There is no additional practical circuit implementation. Thus, the sequential synchronization of information propagation must be ensured in the formal model so that the complete information leakage path with sequential property can be detected.

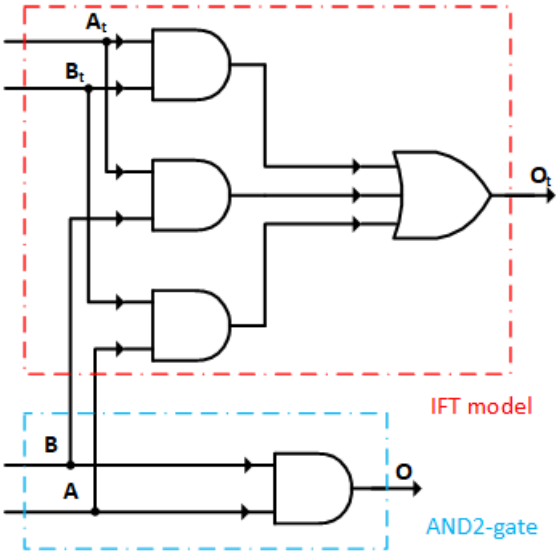

Figure 1. The AND-2 gate-level IFT model [20].

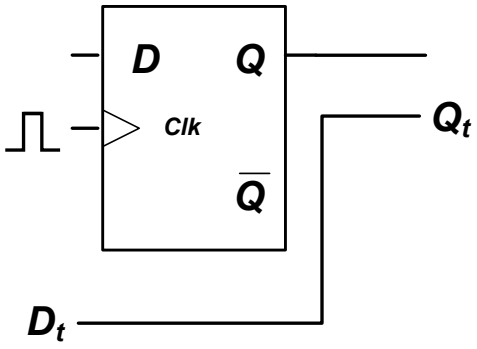

Figure 2. The DFF IFT model.

2.4. SMT Solver

Satisfiability (SAT) solvers have been used in many electronic design automation (EDA) fields such as logic synthesis, verification, and testing. The SAT solvers are originally designed to solve the well-known Boolean Satisfiability problem, which decides whether a propositional logic formula can be satisfied given value assignments of the variables in the formula. Based on SAT solver, SMT solver is derived by including several first-order theories, such as arithmetic, bit-vectors and quantifiers [23]. However, due to the high computational complexity, there is no hardware implementation for SMT solvers, and the software-based SMT solver is not scalable to large designs. Z3 is a popular used SMT solver providing efficient verification and analysis applications [23]. It is assembled in the Python environment as Z3PY, which is a convenience for developing practical tools [34].

3. Methodology

The proposed framework automates the formal verification by realizing IFT in the gate-level net-list design. Following our preliminary work in [33], it converts the whole hardware design to Z3 constraints and adds extra logic to track security labels. Label checking will be performed in an SMT solver. In this paper, Z3 is utilized as the solver. Therefore, a parser is developed to translate the net-list to its equivalent Z3 constraints along with the extra GLIFT logic generations.

3.1. Framework Overview

The working procedure of the proposed formal framework is shown in Figure 3. The gate-level net-list data is input to a parser, where the original hardware design is parsed to its formal equivalent representations, called the functional circuit representations F. In the meantime, the parser further generates extra logic gates to introduce and track security labels. Those logic gates are denoted as IFT circuit representations I, which compose the GLIFT logic. Both representations F and I are in the form of Z3 constraints. Hence we define the formal model M as Equation (2).

$$M := F \wedge I \tag{2}$$

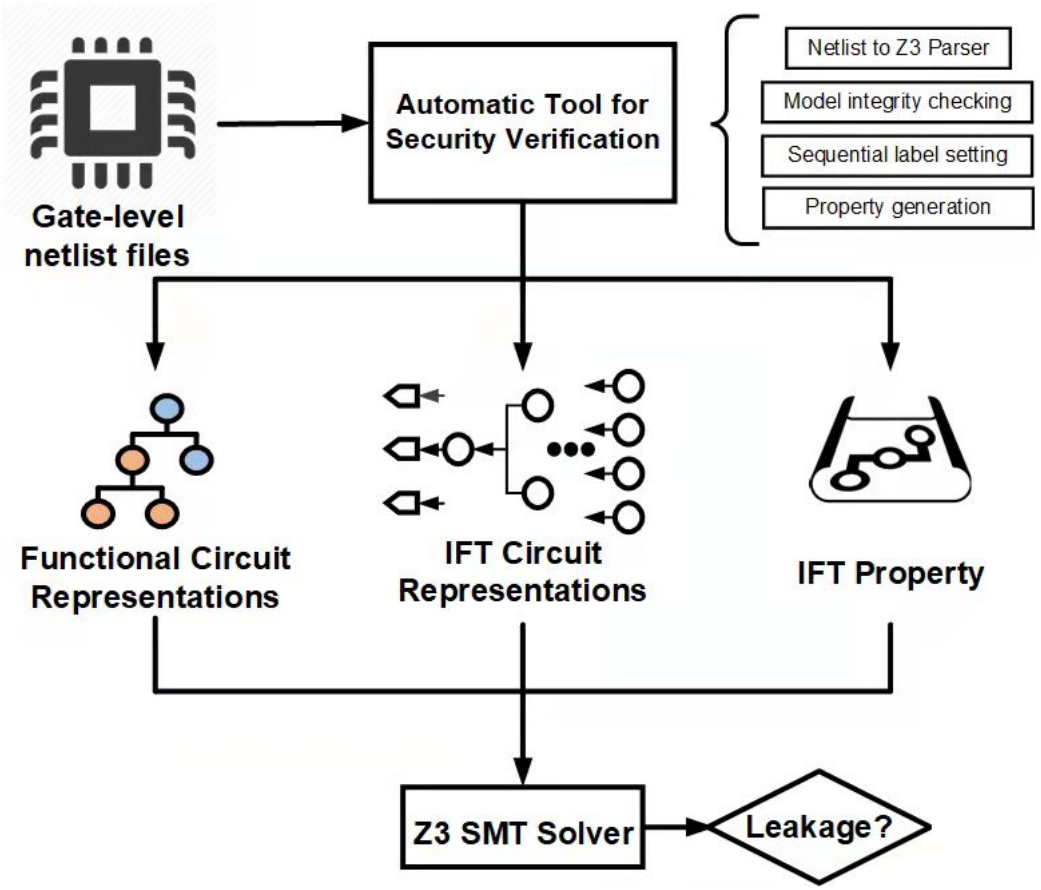

Figure 3. Working procedure of the proposed formal framework.

Taking the logic gates in Figure 1 as an example, signal $\{A, B, O\}$ are composed following constraints in F while signal $\{A_t,B_t,O_t\}$ are composed based on constraints in I. The corresponding procedure of deriving formal model M is shown as follows.

$$F := (O == A\&B)$$

$$I := (O_t == (A_t\&B_t)|(A_t\&B)|(A\&B_t))$$

$$\begin{aligned} M := F \wedge I := (O == A\&B) \wedge \\ (O_t == (A_t\&B_t)|(A_t\&B)|(A\&B_t)) \end{aligned}$$

where & stands for the *and* operation and | stands for the *or* operation. M is the model, which input to the Z3 platform. IFT properties are denoted as P, indicating sensitive data bits. Input to the Z3 solver, P is in the form of Z3 constraints as well. The constraints C, which need to be checked in the end, are conjunctions of M and P. Taking the circuit in Figure 1 as as example, we assume that B is of high sensitivity while A is in low sensitivity.

It leads to label value 1 in B_t and label value 0 in A_t. If the output O leaks sensitive information, then we will have O_t of label 1. We can derive the C as follows.

$$P := (A_t == 0) \wedge (B_t == 1) \wedge (O_t == 1)$$

$$\begin{aligned} C := P \wedge M := & (A_t == 0) \wedge (B_t == 1) \wedge (O_t == 1) \wedge \\ & (O == A \& B) \wedge \\ & (O_t == (A_t \& B_t) | (A_t \& B) | (A \& B_t)) \end{aligned}$$

Z3 SMT solver is then utilized to check C. If there is no solution, whatever the inputs are, there is no path to propagate the high sensitive label to the output O_t. The design is highly secure regarding the confidentiality property. Otherwise, the high sensitive label can be propagated to the output port and observed by the attacker by giving the solution as input. In this example, the solutions $\{A = 1, B = 0\}$ and $\{A = 1, B = 1\}$ are obtained by the Z3. Therefore, the design in Figure 1 has information leakage paths.

3.2. Sequential Split Strategy

We propose a sequential split strategy to solve the timing synchronization problem of the original circuit information flow and the extra circuit information flow, as mentioned in Section 2.3. First, we analyze the RTL hardware program, named RTL code, before circuit synthesis. Figure 4 shows an example of the result of the sequential logic synthesis (right side) in the RTL code (left side). When the structure shown in Figure 4 appears in the code, there will be a series of DFFs in the net-list after synthesis. To avoid the sequential synchronization problem mentioned in Section 2.3, we split the RTL code into two parts—before and after the sequential statement code. Each part will only contain combinational logic code statements. After the split, every individual part will be synthesized into net-list data. That is, there is no net-list sequential code block or logic cell inside an individual part, as shown in Figure 4. Such individual Net-list file can be translated to the formal model in Z3 by using our automatic parser.

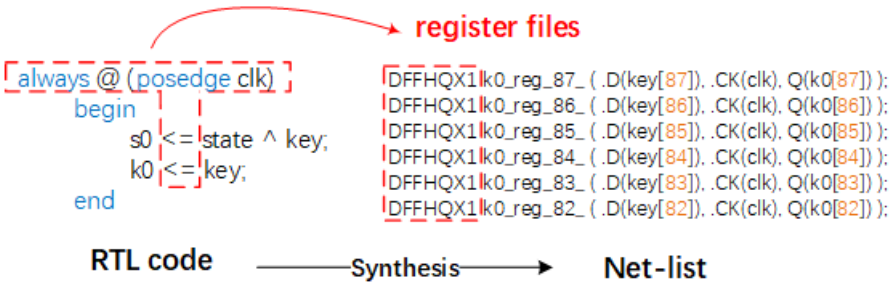

Figure 4. Synthesis results of sequential block in RTL code.

Then these individual net-list files are associated together in a cascaded way. As a pipeline, each circuit part's input and output signals are extracted as the connection of two cascaded modules in the adjacent clock cycles. The logical relationship between the output of modules in the previous clock cycle and the modules' input in the next clock cycle is declared. At the same time, the shadow logic of the above relationship is declared either. Then, all this connection information is added into the model to formal the model of the whole circuit. In the end, the complete model contains several parts, where every part represents the circuit design logic and GLIFT logic of each clock cycle.

4. Tool Design

We developed an automatic tool for security verification. It first translates the net-list to Z3 model/constraints, then checks the model's integrity. After that, the time label is added to every submodel represented as individual parts in Section 3.2. Along with the model establishment step, the property based on GLIFT theory is generated as well. The

tool is written in Python, and the structure is shown in Figure 5. Every block in the tool structure is introduced as follows.

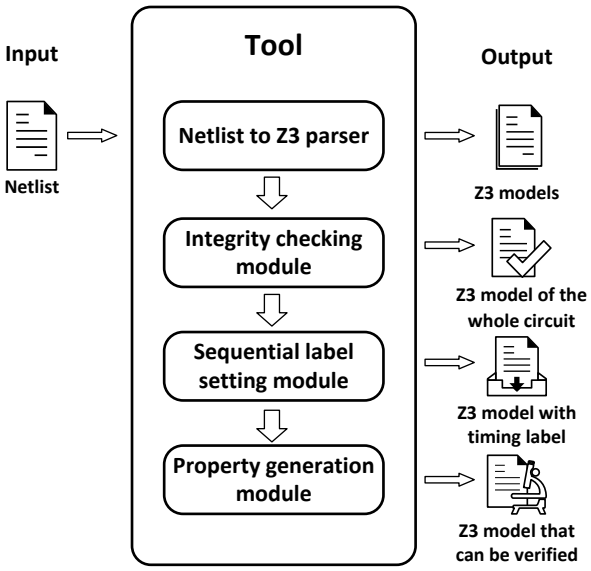

Figure 5. Structure of the designed tool.

4.1. Net-list to Z3 Parser

We developed an automatic parser for converting and generating Z3 constraints from the gate-level net-list. The parser is written in Python and has the structure shown in Figure 6. There are two parts in the parser—code analysis and code generation. The code analysis part interprets the net-list file. It generates wires and registers that are utilized in the functional circuit. Especially, the input and output signals are extracted from those wires/registers, which are assistant to the following property design and model integration. Then in the code generation, the functional circuit representations F and IFT circuit representations I are produced. As a result, rely on the extracted inputs and outputs, F and I are integrated for the Z3 solver.

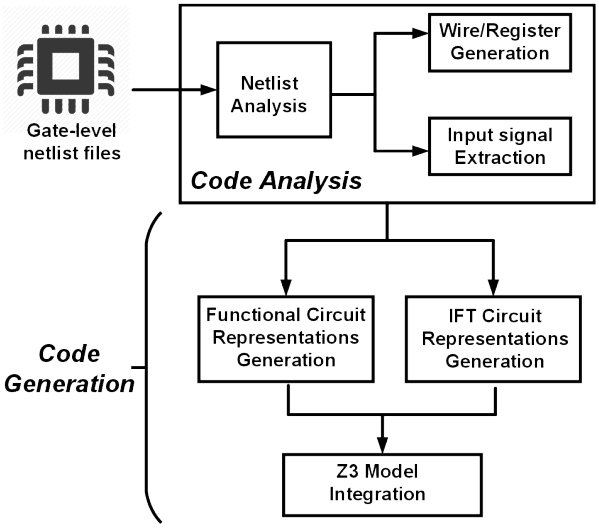

Figure 6. Block structure of the developed parser.

4.2. Integrity Checking Module

An integrity-checking module is designed to ensure the integrity of the model. To handle sequential circuits, we split the circuit as discussed in Section 3.2. Accordingly, the

models are composed of separated sub-models. Therefore, the connection signals between cascaded parts of circuits and corresponding IFT signals are declared especially. In addition, the types of logic gates that make up the net-list are shown up. The result is compared with the logic gate library utilized in the Net-list to Z3 parser code generation. If any type of logic gate in net-list does not appear in the preset logic gate library, errors will be reported, guaranteeing the model's integrity.

4.3. Sequential Label Setting Module

We label the model with timing tags to make the model more specific and the leakage path clearer in the sequential aspect. An ergodic algorithm is applied to label every variation in the model timing. For example, the variation $N3$ at the first clock cycle is transformed to $N3_T1$ after this procedure. The functional circuit representations F and IFT circuit representations I are transformed to F_T and I_T as the output of this module.

4.4. Property Generation Module

As one of the essential parts of formal verification, we set two property generation methods in this part according to the GLIFT theory. The premise is that the IFT logic of security sensitive signals is set as high. One of the two theories is to set an OR gate for all the IFT logic of the output signals. Then it checks if a solution can result in a high logic at the output of the OR gate. The other is to set the logic value of the IFT logic of the suspect output port as high and then find if a solution can satisfy this condition. We choose one of these two methods to generate property according to the characteristics of the experimental circuit.

5. Experiments

This section demonstrates the proposed information flow tracking based automated formal verification. The experiment is set up in Python environment and evaluates IFT property in Verilog net-list benchmarks. Trojans are inserted into the genuine benchmarks, while properties are designed as adding labels in IFT circuit representations.

5.1. Experimental Setup

To use the proposed framework in practical applications, a developer/user only needs to indicate high-sensitive bits in the IFT circuit representations' input signals and those outputs observable by attackers. In the experiment, some specific data bits are treated as secrets, and confidentiality is checked for those labeled secrets.

The main tool utilized for experimentation is Z3 SMT Solver. API of Z3 has been assembled in the Python environment as Z3PY. The Z3 solver is in the same environment as the net-list to Z3 parser, which makes the toolchain be integrated easily. We employ the Z3 to check if the tainted label of secret information can be delivered to the IFT circuit's output. All the demonstrations are executed in Windows 10 on a computing machine with Core(TM) i3-9100 CPU(manufactured in Intel Corporation, Santa Clara, CA, USA) @3.60 GHz and 8 GB memory.

To demonstrate the practicality of our proposed framework, we evaluate 22 ISCAS'85 gate-level net-list benchmarks [35,36]. Those benchmarks are written in Verilog and have been synthesized using Cadence Genus. They provide combinational logic circuits to let users test different mythologies. To fit the attack model in this paper, we insert the leakage paths to simulate hardware Trojans into the design. In addition, we choose one-round AES circuit as the benchmark to prove our framework's practicality in sequential circuits. Furthermore, hardware trojans, customized based on Trojans in Trust-Hub, are inserted to establish sneaky information-leaking paths. Then, the net-list to Z3 parser translates the Trojan inserted benchmarks to models in Z3, while the IFT logic of the benchmark is generated simultaneously. After that, we establish the solver and add constraints standing for IFT properties. The model and properties are finally checked together in the Z3 platform.

5.2. Leakage Paths Checking

In Figure 7, we show a template of inserted hardware Trojan design. All Trojans in our ISCAS85 benchmarks follow this structure and will leak information about the circuit. Specifically, the inserted hardware Trojans are combinational circuits composed of AND, NAND, and NOR gates. The trigger of the Trojan is connected to the input ports and would be activated by a specific input pattern. The Trojan payload enables an AND gate and passes the sensitive information to the output ports.

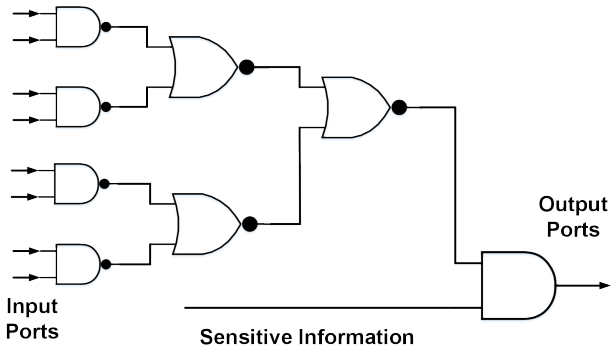

Figure 7. Design of inserted sneaky paths in ISCAS85 benchmarks.

Figure 8 shows the trojan we design for AES-T3. Once the input matches the preset value, the Trojan trigger outputs a high signal value. Then the Trojan payload is activated to leak the secret information, the Key of AES.

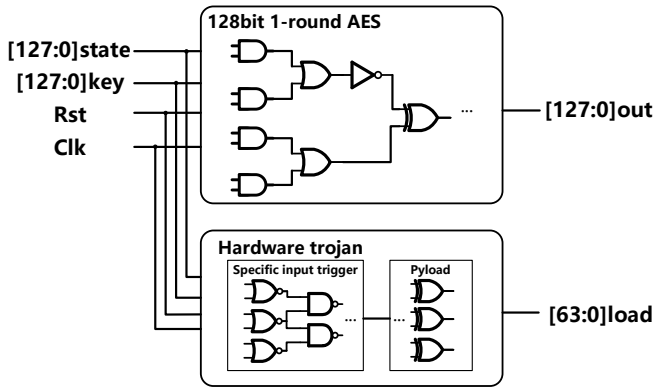

Figure 8. Design of inserted sneaky paths in benchmark AES-T3.

For each combinational benchmark, We denote one data bit in a specific input as the secret and set its label as high. For sequential benchmark, we consider the 128-bit key signals as the secret information and label all the 128-bit signals as high. Output bits that the Trojan influences are defined as vulnerable output ports. The security property is represented as "Assigning the high sensitive label to a secret and low sensitive labels to the rest signals, whether there exists at least one solution causing high sensitive label appeared on vulnerable output ports". In other words, if the Z3 finds a solution, then the Trojan is detected. As a counter-example, the solution is the input vector that propagates secrets to outputs.

5.3. Results and Analysis

Table 2 shows the results of our experiments. Again, Trojans are inserted into all the benchmarks for leaking information. Among those benchmarks, the Trojans in "memetrl" and "div" are always on, while the others are triggered by a signal. We account for the number of logic gates from the net-list data design files in the column of the functional

gate, and the number of GLIFT gates from the formal model in the column of IFT gate. The gate number in IFT logic is $3-10\times$ more than functional gates. It indicates the huge area overheads would be caused if we implement the GLIFT logic in real hardware circuits.

Table 2. Tests on Trojan insertion benchmarks.

Benchmarks	Format	Trigger Mode	Functional Gate	IFT Gate	Model Time (ms)	Detection Time (ms)	Total (ms)	Detected
c17	generic gate verilog	signal trigger	6	24	70	24	94	Yes
c432	generic gate verilog	signal trigger	160	775	77	143	220	Yes
c499	generic gate verilog	signal trigger	202	888	65	130	195	Yes
c880	generic gate verilog	signal trigger	383	1455	63	210	273	Yes
c1355	generic gate verilog	signal trigger	546	3315	74	275	349	Yes
c1908	generic gate verilog	signal trigger	880	2168	74	250	324	Yes
c2670	generic gate verilog	signal trigger	1193	3290	81	509	590	Yes
c3540	generic gate verilog	signal trigger	1669	4638	91	663	754	Yes
c5315	generic gate verilog	signal trigger	2307	7995	116	1112	1228	Yes
c6288	generic gate verilog	signal trigger	2416	9364	134	1085	1219	Yes
c7552	generic gate verilog	signal trigger	3512	9809	138	1361	1499	Yes
bar	generic gate verilog	signal trigger	2960	15,271	1391	2553	3944	Yes
max	generic gate verilog	signal trigger	5065	14,216	4486	4486	8972	Yes
sin	generic gate verilog	signal trigger	7656	25,970	2634	2323	4957	Yes
arbiter	generic gate verilog	signal trigger	23,189	59,600	4605	7087	11,692	Yes
voter	generic gate verilog	signal trigger	25,993	69,436	9437	8805	18,242	Yes
square	generic gate verilog	signal trigger	35,264	91,406	7976	12,017	19,993	Yes
sqrt	generic gate verilog	signal trigger	36,787	122,891	34,187	38,399	72,586	Yes
multiplier	generic gate verilog	signal trigger	42,974	131,690	8664	19,698	28,562	Yes
log2	generic gate verilog	signal trigger	46,746	155,981	27,165	26,367	53,532	Yes
memctrl	generic gate verilog	always on	81,588	224,428	24,932	32,611	57,543	Yes
div	generic gate verilog	always on	100,985	274,228	27,908	264,972	320,788	Yes
AES1-T1	180 nm CMOS library	signal trigger	6956	82,182	3,185,023	25,703	3,210,726	Yes
AES1-T2	180 nm CMOS library	always on	6861	81,055	3,184,493	25,608	3,210,101	Yes
AES1-T3	180 nm CMOS library	signal trigger	6905	82,286	3,188,966	25,676	3,214,642	Yes

The time consumption of parsing Verilog net-list to Z3 constraints is listed as model time. Time cost in Z3 solving is listed as the detection time. The column of total time indicates the time consumption from taking in benchmarks to detecting hardware Trojans. Taking the benchmark c6288 as an example, the c6288 includes 2416 gates, from which 9364 GLIFT logic gates are generated by the parser. The time consumption of code parsing and generation is 134 ms. The Z3 solving takes 1085 ms to detect the hardware Trojan. Assuming that the security property has already been designed, the total time cost for detecting Trojan in c6288 is 1219 ms.

We can see that the model time of sequential benchmarks (AES1-T1, AES1-T2, and AES1-T3) are disproportionately more than other benchmarks. Because of the difference in process library, the functional gates of sequential benchmarks are more complex than combinational benchmarks. Thus, the number of functional gate of original circuits is proportionally less than that of combinational benchmarks. Moreover, for sequential circuits, we label every variation in net-list with timing tags to observe which clock the variation is in at the whole sneaky path. The timing tag labeling leads to much more model time consumption in sequential circuits than in combinational circuits. As a result, all Trojans in those benchmarks are detected successfully.

The largest benchmark in this experiment is div which includes $100,985$ functional gates. The total time spent on the security verification is $320,778$ ms or around 5 min. From the results, the evaluation can be finished in minutes. The proposed formal framework is efficient for protecting the confidentiality of the gate-level net-list.

Further, leakage paths can be obtained by analyzing results, which can help developers improve their designs. Figure 9 demonstrates the leakage paths detected in the benchmark c432. The signal N17 is the secret input signal that is tainted and the signal Tj-payload is the output of the Trojan. In this example, we detected 167 leakage paths in the gate-level net-list while 3 of them are shown in the figure. Developers could improve the secure level by adding obfuscation on those paths.

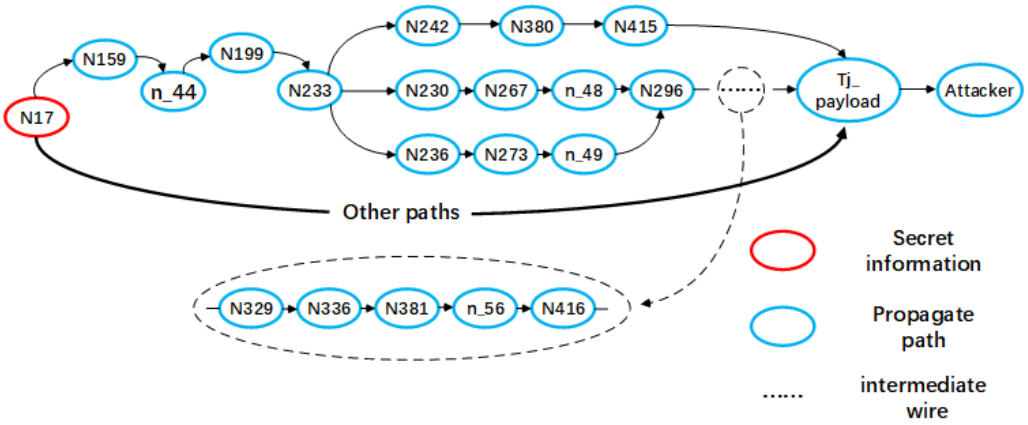

Figure 9. Detected leakage paths of benchmark c432.

6. Limitations and Discussion

Although the proposed framework demonstrates excellent performance in detecting sneaky paths of information leakage, there are some limitations, such as proof of a very large-scale circuit and the need for a fully automatic sequential logic process algorithm. SMT solving is often efficient in obtaining a result where a solution exists. However, it becomes an NP-hard problem once there exists no solution to the given problem/constraints. Mapping to our framework, it demonstrates a significant performance in detecting sneaky paths in the condition that a Trojan or vulnerability exists. The solution searching strategy can be optimized to improve efficiency further.

In contrast, if there are no such paths leaking secrets, the solver must check all possible cases before termination, which leads to intense computation complexity. To address this issue, we will set a threshold according to the size of the net-list file. The SMT solving would be terminated in the threshold and report a compromised secure checking to users.

Our parser can currently support combinational and sequential circuit logic parsing and generation. However, the sequential logic needs to first be manually handled and then transformed into a formal model and verified. In the future, we will perfect our framework to be fully automatic and support large-scale circuit verification.

7. Conclusions

This paper proposes a formal framework to protect the confidentiality of hardware design at the gate-level. By designing a parser, the formal model is generated and composed of a functional circuit and GLIFT logic circuit. The Z3 solver validates the model with an IFT property in the end. Moreover, a sequential split algorithm is proposed to guarantee the beingness of the verification result. The framework provides a fully automatic static formal verification from the input net-list file to the IFT property checking. In the future, an automatic sequential circuit-processing module will be added to the framework. Accordingly, larger scale benchmarks with hardware Trojans will be tested. Furthermore, we will extend the framework to cover more properties and features. The integrity property will be considered to identify malicious modifications.

Author Contributions: Methodology, Q.Z. and X.G.; software, Q.Z.; validation, Y.G.; data curation, Z.Z.; writing—original draft preparation, Q.Z.; writing—review and editing, J.H.; visualization, Y.L.; project administration, L.L. and Y.Z.; funding acquisition, Y.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by The Laboratory Open Fund of Beijing Smart-chip Microelectronics Technology Co., Ltd.

Acknowledgments: All of the acknowledgement have been covered in Funding and Author contribution.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Agrawal, D.; Baktir, S.; Karakoyunlu, D.; Rohatgi, P.; Sunar, B. Trojan detection using IC fingerprinting. In Proceedings of the 2007 IEEE Symposium on Security and Privacy (SP '07), Oakland, CA, USA, 20–23 May 2007; pp. 296–310. [CrossRef]
2. Jin, Y.; Makris, Y. Hardware Trojan detection using path delay fingerprint. In Proceedings of the 2008 IEEE International Workshop on Hardware-Oriented Security and Trust, Anaheim, CA, USA, 9 June 2008; pp. 51–57. [CrossRef]
3. Zhang, X.; Tehranipoor, M. Case study: Detecting hardware trojans in third-party digital ip cores. In Proceedings of the 2011 IEEE International Symposium on Hardware-Oriented Security and Trust, San Diego, CA, USA, 5–6 June 2011; pp. 67–70.
4. Love, E.; Jin, Y.; Makris, Y. Proof-carrying hardware intellectual property: A pathway to trusted module acquisition. *IEEE Trans. Inf. Forensics Secur.* **2012**, *7*, 25–40. [CrossRef]
5. Jin, Y.; Yang, B.; Makris, Y. Cycle-accurate information assurance by proof-carrying based signal sensitivity tracing. *IEEE Int. Symp. Hardw.-Oriented Secur. Trust (HOST)* **2013**, 99–106.
6. Jin, Y. Design-for-security vs. design-for-testability: A case study on dft chain in cryptographic circuits. In Proceedings of the 2014 IEEE Computer Society Annual Symposium on VLSI, Tampa, FL, USA, 9–11 July 2014. [CrossRef]
7. De Paula, F.M.; Gort, M.; Hu, A.J.; Wilton, S.J.; Yang, J. Backspace: Formal analysis for post-silicon debug. In *Proceedings of the 2008 International Conference on Formal Methods in Computer-Aided Design*; IEEE Press, Portland, Oregon, USA, 2008; p. 5.
8. Guo, X.; Dutta, R.G.; Jin, Y.; Farahmandi, F.; Mishra, P. Pre-silicon security verification and validation: A formal perspective. In *Proceedings of the 52nd Annual Design Automation Conference*; ACM, San Francisco, CA, USA, 2015; p. 145.
9. Drzevitzky, S. Proof-carrying hardware: Runtime formal verification for secure dynamic reconfiguration. In Proceedings of the 2010 International Conference on Field Programmable Logic and Applications, Milan, Italy, 31 August–2 September 2010; pp. 255–258.
10. Rajendran, J.; Vedula, V.; Karri, R. *Detecting Malicious Modifications of Data in Third-Party Intellectual Property Cores. Ser.*; DAC '15: New York, NY, USA, 2015; pp. 112:1–112:6.
11. Henzinger, T.A.; Jhala, R.; Majumdar, R.; Sutre, G. Software verification with blast. In *Model Checking Software*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 235–239.
12. Guo, X.; Dutta, R.G.; Mishra, P.; Jin, Y. Automatic code converter enhanced pch framework for soc trust verification. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2017**, *25*, 3390–3400. [CrossRef]
13. Myers, A.C.; Liskov, B. A decentralized model for information flow control. In Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), Saint-Malo, France, 5–8 October 1997; pp. 129–142.
14. Zhang, D.; Askarov, A.; Myers, A.C. Language-based control and mitigation of timing channels. *ACM SIGPLAN Not.* **2012**, *47*, 99–110. [CrossRef]
15. Zhang, D.; Wang, Y.; Suh, G.E.; Myers, A.C. A hardware design language for timing-sensitive information-flow security. *ACM SIGPLAN Not.* **2015**, *50*, 503–516. [CrossRef]
16. Li, X.; Tiwari, M.; Oberg, J.K.; Kashyap, V.; Chong, F.T.; Sherwood, T. Hardekopf, B. Caisson: A hardware description language for secure information flow. *ACM SIGPLAN Not.* **2011**, *46*, 109–120. [CrossRef]
17. Li, X.; Kashyap, V.; Oberg, J.K.; Tiwari, M.; Rajarathinam, V.R.; Kastner, R.; Sherwood, T.; Hardekopf, B.; Chong, F.T. Sapper: A language for hardware-level security policy enforcement. In *ACM SIGARCH Computer Architecture News*; ACM: New York, USA, 2014; Volume 42, pp. 97–112.
18. Guo, X.; Dutta, R.G.; He, J.; Tehranipoor, M.M.; Jin, Y. Qif-verilog: Quantitative information-flow based hardware description languages for pre-silicon security assessment. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*; IEEE: McLean, VA, USA, 2019; pp. 91–100.
19. Hu, W.; Oberg, J.; Irturk, A.; Tiwari, M.; Sherwood, T.; Mu, D.; Kastner, R. On the complexity of generating gate level information flow tracking logic. *IEEE Trans. Inf. Forensics Secur.* **2012**, *7*, 1067–1080. [CrossRef]
20. Hu, W.; Mao, B.; Oberg, J.; Kastner, R. Detecting hardware trojans with gate-level information-flow tracking. *Computer* **2016**, *49*, 44–52. [CrossRef]
21. Qin, M.; Hu, W.; Wang, X.; Mu, D.; Mao, B. Theorem proof based gate level information flow tracking for hardware security verification. *Comput. Secur.* **2019**, *85*, 225–239. [CrossRef]
22. Deng, S.; Gümüsoglu, D.; Xiong, W.; Gener, Y.S.; Demir, O.; Szefer, J. Secchisel: Language and tool for practical and scalable security verification of security-aware hardware architectures. *IACR Cryptol. ePrint Arch.* **2017**, *2017*, 193.
23. De Moura, L.; Bjørner, N. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*; Springer: Budapest, Hungary, 2008; pp. 337–340.
24. Hu, W.; Wu, L.; Tai, Y.; Tan, J.; Zhang, J. A unified formal model for proving security and reliability properties. In Proceedings of the 2020 IEEE 29th Asian Test Symposium (ATS), Penang, Malaysia, 23–26 November 2020; pp. 1–6.
25. Kumar, B.; Jaiswal, A.K.; Vineesh, V.S.; Shinde, R. Analyzing hardware security properties of processors through model checking. In Proceedings of the 2020 33rd International Conference on VLSI Design and 2020 19th International Conference on Embedded Systems (VLSID), Bangalore, India, 4–5 January 2020; pp. 107–112.
26. Khalid, F.; Abbassi, I.H.; Rehman, S.; Kamboh, A.M.; Hasan, O.; Shafique, M. Forasec: Formal analysis of hardware trojan-based security vulnerabilities in sequential circuits. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2021**, *41*, 1167–1180. [CrossRef]

27. Bouzafour, A.; Renaudin, M.; Garavel, H.; Mateescu, R.; Serwe, W. Model-checking synthesizable systemverilog descriptions of asynchronous circuits. In Proceedings of the 2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), Vienna, Austria, 13–16 May 2018; pp. 34–42.
28. He, J.; Guo, X.; Meade, T.; Dutta, R.G.; Zhao, Y.; Jin, Y. Soc interconnection protection through formal verification. *Integration* **2019**, *64*, 143–151. [CrossRef]
29. Pieper, P.; Herdt, V.; Große, D.; Drechsler, R. Dynamic information flow tracking for embedded binaries using systemc-based virtual prototypes. In Proceedings of the 2020 57th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 1–5 June 2020; pp. 1–6.
30. Solt, F.; Gras, B.; Razavi, K. Cellift: Leveraging cells for scalable and precise dynamic information flow tracking in RTL. In *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, 10–12 August 2022*; Butler K.R.B., Thomas, K., Eds.; USENIX Association, 2022; pp. 2549–2566. Available online: https://www.usenix.org/conference/usenixsecurity22/presentation/solt(accessed on 14 December 2022).
31. Tiwari, M.; Wassel, H.M.; Mazloom, B.; Mysore, S.; Chong, F.T.; Sherwood, T. Complete information flow tracking from the gates up. *ACM Sigplan Not.* **2009**, *44*, 109–120. [CrossRef]
32. Blackstone, J.; Hu, W.; Althoff, A.; Ardeshiricham, A.; Zhang, L.; Kastner, R. A Unified Model for Gate Level Propagation Analysis. 2020. Available online: https://arxiv.org/abs/2012.02791 (accessed on 11 December 2022).
33. Zhang, Q.; He, J.; Zhao, Y.; Guo, X. A formal framework for gate-level information leakage using z3. In *2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*; IEEE: Shanghai, China, 2020; pp. 1–6.
34. Microsoft Research. Z3 api in Python. Available online: https://ericpony.github.io/z3py-tutorial/guide-examples.htm (accessed on 11 December 2022).
35. Hansen, M.C.; Yalcin, H.; Hayes, J.P. Unveiling the iscas-85 benchmarks: A case study in reverse engineering. *IEEE Des. Test Comput.* **1999**, *16*, 72–80. [CrossRef]
36. EPFL and ISCAS85. Epfl and iscas85 Combinational Benchmark Circuits in Generic Gate Verilog. Available online: https://github.com/jpsety/verilog_benchmark_circuits (accessed on 11 December 2022).