2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)

A Noval Method of Security Verification for JTAG Protection Function

Dongfang Li
Beijing Institute of Computer Technology and Applications
Beijing, China
dongfang.li@outlook.com

Wei Shen
Beijing Institute of Computer Technology and Applications
Beijing, China
shenwei_618@outlook.com

Zhihao Wang
Beijing Institute of Computer Technology and Applications
Beijing, China
wangzhihaoxjtu@163.com

***Abstract*—This paper proposed a formal verification method for JTAG security based on information flow tracking. The security property script is used to describe the security requirements. Compared with the traditional writing method of assertions, our method does not need to consider much about the design features, which not only greatly reduces the assertion writing time but also can effectively detect the security violations in the design. Based on the generated information-flow tracking model, the proposed method can generate formal constraints and System Verilog Assertions supported by formal verification tools. Experiment of JTAG security verification proves that the proposed method can effectively verify the security functions related to information flow such as access control.**

Keywords-JTAG security; security verification; formal verification; information flow

I. INTRODUCTION

Hardware security has become an important challenges in IC design , production and manufacture. Related work proves that it is possible to attack some critical systems [1]. JTAG is widely used in hardware and software debugging, download firmware, and chip testing, providing powerful functions for accessing data and circuits in chips. The OpenSPARC T2 [2], for example, has more than ten debugging functions within the JTAG, such as L2 cache r/w, memory built-in self-test (MBIST), and direct memory access. Therefore, the security of JTAG affects the security of the entire system. This article provides formal verification method for JTAG interface protection security design.

In recent years, with the continuous improvement of the formal engines performance and standardization of property description languages, model checking techniques have been widely used in SoC/FPGA functional verification. However, when facing with special features such as security, it is not affordable to solve all problems by formalization verification. High quality assertions is needed by formal verification, and verification engineers need understand the design under test completely. This puts tremendous commercial pressure on semiconductor companies that are trying to balance fully security verification and tight time-to-market.

Information flow tracking (IFT) provides a common method for the security verification of integrated circuit design, which can help identify and eliminate potential security vulnerabilities, hardware Trojans and time channels. In this paper, we propose a formal verification method based on information flow tracking. Based on the information flow tracking model, the security property description script is used to generate security properties, and the properties will be compiled into generic System Verilog Assertions by a specific compiler. The proposed method in this paper simplifies the way of writing formal assertions, and can effectively verify security requirements and information flow related functions.

We take advantage of the accurate of information flow tracking, which also only needs to involve a small part of the design, and a formal verification method for access control security is proposed. The effectiveness of the proposed method is proved by experimental results of JTAG security verification. Specifically, we make the following contributions.

- Proposing a formal verification method for access control security;
- Providing a security property description method related to information flow;
- Applying proposed method to verify a JTAG interface protection security design to show advantages of the proposed method.

II. RELATED WORK

IFT is widely used to verify security of system. Tiwari et al. proposed a gate-level information flow tracking (GLIFT) technique for accurately measuring the information flow at the logic gate level [3]. GLIFT technique can help diagnose and eliminate potential security risks in IC designs, such as time channels [4], hardware Trojans and so on [5].

Some research use formal verification method combined with information flow tracking techniques to detect security threats such as hardware Trojans in integrated circuits. Rajendran et al [6] proposed a formal method which has achieved the detection of hardware Trojans that maliciously modify 3PIP circuits. Some other similar method has been proposed to verify unauthorized information leakage in IP cores [7]. Wei et al [8] successfully detected the hardware Trojans of SoC with IFT.

III. SECURITY VERIFICATION FLOW FOR JTAG INTERFACE PROTECTION DESIGN

The gate-level information flow tracking is implemented at gate-level netlist. So this method is generic and can be

978-1-7281-3925-8/19/$31.00 ©2019 IEEE
DOI 10.1109/QRS-C.2019.00093

widely applied to any integrated circuit design in theory. In this paper, we use gate-level information flow tracking technology to verify the security of access control such as JTAG interface protection. Figure 1 shows the proposed verification flow. Based on the theory of information flow tracking, we established information flow tracking model. Then, we compile security requirements into SVA assertions for formal verification. The constraints, assertions, and information flow tracking models are compliant with standard System Verilog, Verilog syntax, so model check can be performed through formal verification tools.

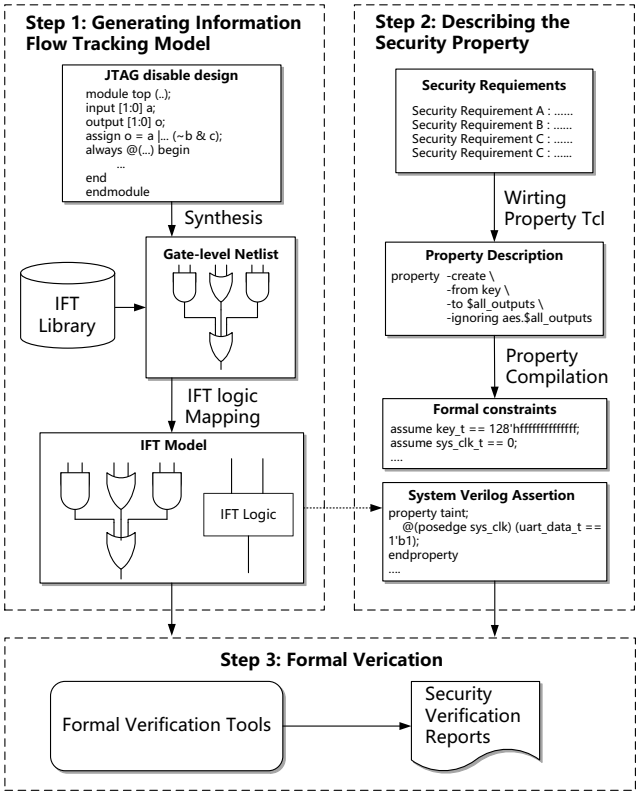

Figure 1. Security verification flow for JTAG interface protection design

After the design under test is synthesized into a gate-level netlist, information flow tracking logic is added for each logic gate. The step of adding the flow tracking logic is similar to the logical mapping. And finally, the netlist is enhanced with specific information flow tracking logic. It is worth noting that the generated information flow tracking model has both original logic and information flow tracking logic.

Generally, the security specifications of access control are written in natural language. By writing a specific security property script, the formal verification based on information flow tracking can implement the description of the security requirements related to information flow. Compared with writing assertions directly, this description method only needs to concern the start and end signals of the information flow, which greatly reduces the difficulty of assertion writing. The Formal verification of security requirements can be performed with formal constraints, assertions and information flow tracking models generated by our method.

IV. Information Flow Tracking Model Generation

Information flow tracking provides a security verification model for the information propagation path in digital circuits, which can use taint labels to tack the flow of information data in the Boolean layer. In IFT model, signals are assigned tags called taint labels that represent different levels of security, such as untrusted/trusted. These labels do not affect the original function of the circuit, but instead provides a model for understanding how data propagates based on the original function. When the integrated circuit is running, the taint labels of each signal are obtained through special logic operations. When the taint label of a signal is logic true '1', we call this signal tainted. When the taint label of a signal is logic false '0', we call this signal untainted.

Algorithm 1 IFT Generation

1: **Input :** Verilog Gate-level Netlist
2: **Output :** Information Flow Tracking Model
3: **for each** net op (a, b, c); do
4: instantiate IFT-enhanced operation:
5: op_IFT(a, a_t, b, b_t, c, c_t);
6: **end**

The information flow tracking model generation algorithm is shown in algorithm 1. At first, the RTL design under test is synthesized into a logical netlist with logic synthesis tool. Since the complete set of cells that compose the netlist is fixed, we define the taint labels and their propagation logic for all the basic logical units at the gate level, and form them into the IFT basic logic library. Then all the cells in netlist is traversed in turn, replacing them with the IFT-enhanced cells in the IFT basic logic library (Lines 5 and 6 in IFT generation algorithm). The generated IFT model can help monitor and verify the information flow in the design effectively.

V. Security Property Description

A. Security Property

Corresponding to functional properties, integrated circuit design also has security properties. The security and functional properties of the system can interact are inseparable. Compared with the functional properties of the system, the security properties are usually more abstract. The five basic elements of information security are confidentiality, authenticity, integrity, availability, and non-repudiation. The proposed verification method based on IFT in this paper is aimed to solve the security verification of confidentiality, integrity and availability in IC design.

- Confidentiality: Confidentiality refers to the feature of not leaking security assets to unauthorized users. For example, for cryptographic module circuits, we do not want to expose the encryption key by security

threats such as hardware Trojans, side channels, and time channels.

- Integrity: Integrity refers to the feature of keeping the information from being destroyed, modified or lost during information transmission. For example, we do not want the critical registers that control core functionality of system to be accessed by untrusted inputs.
- Availability: Availability refers to the feature of providing service for an authorized entity, which means the required information can be accessed duly when needed. For example, denial of service in a network environment, disruption of the network and system are all attacks on availability.

B. Description Method

In this paper, we use the security property script to implement the description of requirements related to the information flow. The syntax of the property description i as follows.

The requirements related to the information flow can be described in code by using security property script. Through a specific compiler, the properties eventually compiled into formal constraints and System Verilog Assertion based on IFT model. Compared with manually writing security property assertions, security engineer only needs to pay attention to the security-related information behavior in access control by using security property scripts. Therefore, the whole complex features of the design do not need to be considered completely, which significantly reduces the difficulty of assertion writing.

```
1.  d_prop -create
2.   -from <start_signal> -to <end_signal>
3.    [ -unreachable ]
4.    [  -condition <expression> ]
5.    [ -ignoring <signal> ]
```

Figure 2. The security property script

Users can use the command '-from <start_signal> -to <end_signal>' to write assertions which check the existence of an information flow. The parameter '-from' can be used to set the start signal of the information flow, and the parameter '-to' can be used to set the end signal of the information flow. In the verification of confidentiality, the parameter '-from' can be set as the security asset signal, and the parameter '-to' can be set as the untrusted port. Similarly, we can verify the integrity by reversing the position of these two parameters.

Users can set the parameter '-unreachable' to write assertion which check the nonexistence of an information flow. In the verification of availability, this command can be used to determine if two signals can affect each other.

The command '-condition <expression>' can be used to add check condition of the assertion. By using this command, a whitelist of behaviors allowed by the security requirements is created. The assertion check will be disabled under the behaviors in whitelist. By setting security conditions, we can not only allow the operation of some special functions in the system, but also clarify the scope of security property verification, which reduces the false positive rate of security verification.

The command '-ignoring <signal>' can be used to block the specific information flow during assertion check, because the specific information flow that meet the security requirements are allowed and should not be a counterexample reported by assertions. By setting the parameter '-ignoring', we can not only block the information flow that meets the security requirements, but also compress the formal verification state space, which improves the verification efficiency.

VI. Experimental Results

In this section, we use the proposed method to perform formal verification of the security of a JTAG protection design.

A. Introduction of the JTAG Protection Design

Both DSP and FPGA are debugged through the JTAG interface. In the debug mode, the memory data and the flash data can be read through the JTAG interface of DSP, and the configuration file can also be obtained through the JTAG interface of FPGA. To prevent sensitive data from leaking, the JTAG interface needs to be blocked under some conditions. The way to disable JTAG is to encapsulate it inside the chip, not for external users.

For the JTAG interface, it needs to be disabled by hardware logic after finishing application debugging before the chip is delivered. At the same time, in order to facilitate the user's upgrade, the disable JTAG interface shall be enabled under certain conditions. The JTAG protection design is shown in Figure 3. In this design, the JTAG of the DSP is disabled by the internal logic of the FPGA, and the JTAG of the FPGA is disabled by an encapsulated combinational logic circuit. The functions of the design are as follows.

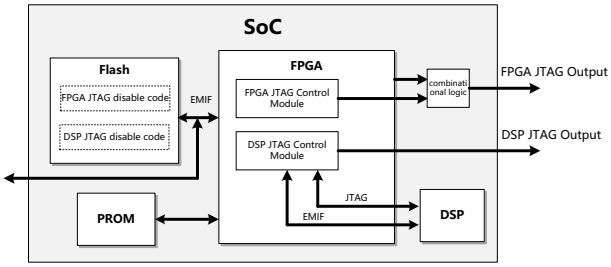

Figure 3. The JTAG protection design

1) Disable JTAG interface of DSP: The JTAG signal of DSP can be accessed through FPGA internal logic. After finishing debugging, the special command code is written to the internal flash address 0xFFFF2. The FPGA will acquire the command code after each power-on loading. If the disable condition is met, the FPGA disables the DSP JTAG signal, so that users cannot access the DSP's JTAG interface.

2) Disable JTAG interface of FPGA: After the debugging is completed, the special command code is

written to the internal flash address 0x1FFFF2. The FPGA acquires the command code after power-on loading, and outputs a disable signal if the disable condition is satisfied. Then, the output signal is performed an AND logic operation with the JTAG signal of the FPGA, making it impossible for the user to access the JTAG interface of the FPGA.

3) Unlock the JTAG interface: An FPGA input pin 'unblock_jtag' is reserved as the input port for the unlock function. During the operation of the chip, when the JTAG needs to be unlocked, a 256-bit unlock signal inputs to the pin at 500Hz clock frequency. FPGA verifies this signal in real time. If the 256-bit signal is equal to the unlock key, the FPGA undo the JTAG disable signal for DSP and FPGA, which makes the JTAG interface available. The procedure of unlocking the JTAG is shown in Figure 4.

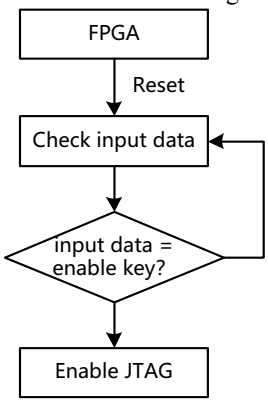

Figure 4. The Procedure of Unlocking the JTAG

B. Information flow tracking model generation for the JTAG protection design

In this experiment, we use Synopsys Design Compiler to synthesize the JTAG protection design into a gate-level netlist. After that, the equivalence check tool is used to verify whether the generated netlist is exactly the same as the original RTL design. Then, each gate-level cell in the netlist is mapped with an IFT logic cell. When it is completed, an information flow tracking model which has both the original logic and the enhanced information flow tracking logic is generated.

C. Security Properties of the JTAG Protection Design

The security requirements for JTAG blocking control are a set of access control rules as follows.

- Security requirement a: When the DSP JTAG disable control signal 'DJTAG_disable_data' is equal to a specific disable code, the JTAG signal 'dspjtag_dout' of the DSP is independent of the JTAG signal 'dspjtag_protect_dout' that actually outputted by the SoC.
- Security requirement b: When the FPGA JTAG disable control signal 'FJTAG_disable_data' is equal to a specific blocking code, the JTAG signal 'fpgajtag_dout' of the FPGA is independent of the JTAG signal 'fpgajtag_protect_dout' actually outputted by the SoC.

In register transfer level, access control rules are very difficult for verification engineers to interpret or verify. Engineers was leveraging traditional property description language and standard verification environment to check the security of access control but struggled crafting conventional System Verilog Assertions that adequately addressed the problem, which consumed a lot of verification time. In order to address this problem, the following security properties can be designed by using the property description script proposed in this paper are as follows.

```
1.  d_prop -create
2.      -unreachable -from dspjtag_dout -to
    dspjtag_protect_dout
3.      -condition { DJTAG_disable_data ==
    16'h1CAE }
```

Figure 5. Security property 'a'

```
1.  d_prop -create
2.      -unreachable -from fpgajtag_dout -to
    fpgajtag_protect_dout
3.      -condition { FJTAG_disable_data ==
    16'h02C5 }
```

Figure 6. Security property 'b'

The property 'a' indicates that 'dspjtag_dout' never reaches 'dspjtag_protect_dout' under the condition of 'DJTAG_disable_data == 16'h1CAE', and property 'b' indicates that 'fpgajtag_dout' never reaches 'fpgajtag_protect_dout' under the condition of 'FJTAG_disable_data == 16'h02C5'.

Based on the information flow tracking model, the security properties are compiled into constraints and assertions supported by the formal tool through the property compiler. The generated formal constraints and System Verilog Assertions are as follows.

```
1.  assume   { dspjtag_dout_t == 1'b1 }
2.  assume   { fpgajtag_dout_t == 0 }
3.  assume   { clk_t ==0 }
4.  assume   { rst_t ==0 }
5.  ……
```

Figure 7. Constraint 'a'

```
1.  property assertion_a;
2.      @(posedge clk) ( (DJTAG_disable_data ==
    16'h1CAE) |=> (dspjtag_protect_dout_t ==
    0) );
3.  endproperty
```

Figure 8. Assertion 'a'

```
1.  assume    { dspjtag_dout_t == 0 }
2.  assume    { fpgajtag_dout_t == 1'b1 }
3.  assume    { clk_t ==0 }
4.  assume    { rst_t ==0 }
5.  ……
```

Figure 9. Constraint 'b'

```
1.  property assertion_b;
2.      @(posedge clk) ( (FJTAG_disable_data ==
    16'h02C5) |=> (fpgajtag_protect_dout_t ==
    0) );
3.  endproperty
```

Figure 10. Assertion 'b'

In the IFT model, 'dspjtag_dout_t' and 'fpgajtag_dout_t' represent the taint label of 'dspjtag_dout' and 'fpgajtag_dout'. In the formal constraints, 'dspjtag_dout_t' and 'fpgajtag_dout_t' are respectively set to logic true '1', and taint labels of other input signals are set to logic false '0'. Assertion 'a' and 'b' are used to check whether 'dspjtag_protect_dout_t' and 'fpgajtag_protect_dout_t' are logic false '0'. If assertions are proven, it means that the information of 'dspjtag_dout' and 'fpgajtag_dout' cannot flow to 'dspjtag_protect_dout' and 'fpgajtag_protect_dout', which is in line with the security requirements of JTAG design.

D. Formal Verification Analysis

In this paper, we use Cadence Jasper Gold FPV to perform formal verification of security property. We establish a formal verification project, put the generated information flow tracking model, formal constraints and assertions into the specified path, and use the tcl command to perform formal verification.

Since the formal constraints corresponding to the two security properties are inconsistent, formal verification is performed separately. The formal verification result is shown in Figure 11 and Figure 12. The assertion statement is displayed in the embedded task list. The execution status of all the properties is displayed in the Property Table window. The red cross indicates that the assertion failed, and the green check indicates that the assertion is proven. The final result shows that the assertion_a is failed, the assertion check time is 0.048s, and the coverage time is 0.168s. The security assertion_b is proven and the time cost is 113s. This is because the design under test has a long timing depth. All the cases should be traversed to prove the assertion, which takes a long execution time.

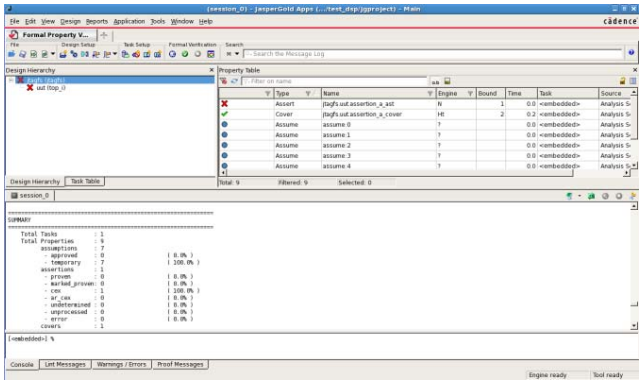

Figure 11. The formal verification result

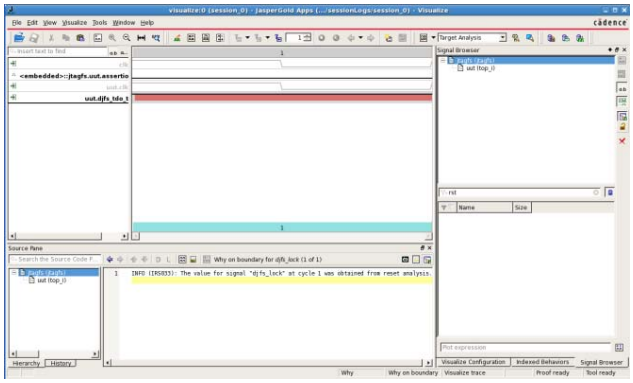

Figure 12. The violation of assertion a

By analyzing the violation of assertion a, it is found that the JTAG disable signal is set to logic true '1' after reset, in which case JTAG can work properly before FPGA receives the JTAG disable command. It may be a security threat if an unauthorized user accesses the SoC through JTAG during this time. The experiment result proves that the security verification of JTAG can be effectively implemented by the proposed method in this paper.

VII. CONCLUSION

In this paper, we proposed a formal verification method for the security access control of integrated circuits. We defined the parameters of the security property script, and generate formal constraints and security assertions based on the information flow tracking model. Compared to the manual assertion writing method, it is not necessary to consider much about the whole functional implementation of the design. Through the experiment of JTAG protection verification, it is proved that the proposed method can effectively help verify security behaviors such as security access control.

ACKNOWLEDGMENT

This work is supported by the Defense Industrial Technology Development Program of China (No.XX2017204C062 and No.XX2016204C001).

REFERENCES

[1] S. Skorobogatov and C. Woods, “Breakthrough Silicon Scanning DiscoversBackdoor in Military Chip,” Tech. Rep., 2012.I. Breeuwsma, “Forensic Imaging of Embedded Systems Using JTAG(Boundary-Scan),” Digital Investigation, vol. 3, no. 1, pp. 32–42, 2006.

[2] “OpenSPARC T2,” http://www.oracle.com/technetwork/systems/opensparc/, Oracle.

[3] Mohit Tiwari, Hassan M.G.Wassel, BitaMazloom, Shashidhar Mysore, Frederic T.Chong, and Timothy Sherwood. 2009. Complete Information Flow Tracking fromthe Gates Up. In the 14th International Conference on Architectural Support forProgramming Languages and Operating Systems. 109–120.

[4] Armaiti Ardeshiricham, Wei Hu, and Ryan Kastner. 2017. Clepsydra: Modeling timing flows in hardware designs. In International Conference on Computer-Aided Design. 147–154. https://doi.org/10.1109/ICCAD.2017.8203772

[5] Yier Jin and Yiorgos Makris. 2012. Proof carrying-based information flow tracking for data secrecy protection and hardware trust. In the 30th IEEE VLSI Test Symposium (VTS). 252 − 257. https://doi.org/10.1109/VTS.2012.6231062

[6] J. Rajendran, V. Vedula and R. Karri, “Detecting malicious modifications of data in third-party intellectual property cores,” DAC 2015.

[7] J. Rajendran, A. M. Dhandayuthapany, V. Vedula and R. Karri, “Formal Security Verification of Third Party Intellectual Property Cores for Information Leakage,” International Conference on VLSI Design, 2016.

[8] Hu, Wei, et al. ”Detecting hardware trojans with gate-level information flow tracking.” Computer 49.8 (2016): 44-52