Graph Neural Network based Hardware Trojan Detection at Intermediate Representative for SoC Platforms

Weimin Fu
weiminf@ksu.edu
Kansas State University
Manhattan, Kansas, USA

Honggang Yu
honggang.yu@ufl.edu
University of Florida
Gainesville, Florida, USA

Orlando Arias
orlandoa@ufl.edu
University of Florida
Gainesville, Florida, USA

Kaichen Yang
bojanykc@ufl.edu
University of Florida
Gainesville, Florida, USA

Yier Jin
yier.jin@ece.ufl.edu
University of Florida
Gainesville, Florida, USA

Tuba Yavuz
tuba@ece.ufl.edu
University of Florida
Gainesville, Florida, USA

Xiaolong Guo
guoxiaolong@ksu.edu
Kansas State University
Manhattan, Kansas, USA

ABSTRACT

The rapid growth of the Internet of Things (IoT) industry has increased the demand for intellectual property (IP) cores. Increasing numbers of third-party vendors have raised security concerns for System-on-Chip (SoC) designers. With the growing complexity of SoC design, the workload is overwhelming for SoC designers to diagnose security vulnerabilities manually. Almost all existing SoC platforms are developed using SystemVerilog. However, there is a lack of reliable security static analysis tools for directly processing the SystemVerilog program. Due to its open-source, flexibility and extendability, RISC-V CPU has become an ideal platform for the IoT applications such as wearable devices, entertainment, smart thermostats, etc. As a result, assuring the trustworthiness of a given RISC-V system is highly desired. This paper proposes a graph neural network-based Trojan detection framework to protect the RISC-V SoC platform written in SystemVerilog from intruding malicious logic. The study is under-construction and planned to be validated on the Ariane RISC-V CPU with several peripheral IPs in the experimental section.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

Anomaly Detection, Hardware Trojans, RISC-V, Intermediate Representations, Graph Neural Networks

GLSVLSI '22, June 6–8, 2022, Irvine, CA, USA.
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9322-5/22/06.
https://doi.org/10.1145/3526241.3530827

ACM Reference Format:
Weimin Fu, Honggang Yu, Orlando Arias, Kaichen Yang, Yier Jin, Tuba Yavuz, and Xiaolong Guo. 2022. Graph Neural Network based Hardware Trojan Detection at Intermediate Representative for SoC Platforms. In *Proceedings of the Great Lakes Symposium on VLSI 2022 (GLSVLSI '22), June 6–8, 2022, Irvine, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3526241.3530827

1 INTRODUCTION

As an open-source instruction set architecture (ISA), RISC-V provides more robust support for low-power and high-performance processor designs. The current RISC-V standard version supports 16-bit, 32-bit, and 64-bit instructions. In addition, the RISC-V-based processor core can support flexible architecture design and help reduce hardware overhead. Therefore, it has been widely used as the platform of Internet-of-Things (IoT) applications. In the meantime, hardware Trojan, as a typical threat of 3rd-party vendors, also places high-security uncertainties on SoC end-users and customers in IoT industry. Unfortunately, although many frameworks have been developed to resist the attacks like return oriented programming (ROP) and jump oriented programming (JOP), few works have been proposed to address micro-architecture level threats in the RISC-V micro-architecture by now. The methods for checking hardware Trojan in a very large system always run into problems such as the state space explosion.

For example, speculation vulnerabilities in the modern processor core are utilized to launch a series of attacks, like branch predictor (BP), branch target buffer (BTB), and return stack buffer (RSB) [22]. Additionally, the vulnerability in out-of-order (OOO) processors is exploited to leak sensitive information by launching side-channel based attacks [24]. Furthermore, more and more hardware security vulnerabilities and Trojans are released through the database, such as the Common Weakness Enumeration [2], and published in security competitions, such as Hack@Dac[1].

However, while most RISC-V SoC designs are built using SystemVerilog, security verification tools that work at the HDL level are lacking [20]. One of the significant reasons is that detecting

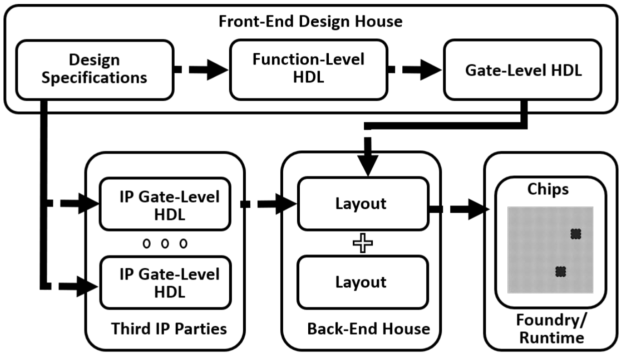

Figure 1: The IC supply chain.

a specific vulnerable structure becomes an NP-Hard problem after mapping hardware design to a graph [14]. On the other hand, designers must convert their SoCs into gate-level netlists for analysis purposes for verification [42]. Unfortunately, high-level circuit structures are lost in post-synthesis netlists and high-level software descriptions. The recovery of high-level circuit structures is also NP-hard and is an ongoing research problem, along with locating the position of threats. To solve the above problem, we developed an approach to train the hardware vulnerability model and then utilize it in threat detection on the micro-architecture level. The hardware design written in SystemVerilog is first parsed into an Abstract Syntax Tree (AST). Yosys transcribes the synthesizable parts of the AST into IR (RTLIL), and at same time, it would generate a graphviz DOT file for the specified component of the design. Following this processing, we acquire the hardware's topology and feed it into a graph neural network for analysis.

In summary, the main contributions of this paper are:

- We investigate the technical details of existing hardware intermediate representations (IRs) frameworks and the corresponding tools that process various hardware description language (HDL) programs.
- By evaluating the topology of hardware design through graph neural networks, we overcome the scalability issues and unpredictable test time associated with simulation-based methodologies.
- We use a residual learning technique to minimize the influence of particular hardware characteristics. Thus, the hardware Trojans embedded in a variety of IP Cores consisted one testbench. Our purpose is to determine the effect of inserted hardware Trojans on the hardware topology. Simultaneously, we utilize an RISC-V SoC Benchmark as our test dataset.

2 BACKGROUND

2.1 Attack Model

The IC supply chain is shown in Figure 1, which involves several stages. First, the function-level HDL is developed from the design specifications and then synthesized into the gate-level netlist files. In this stage, third-party vendors and offshore companies are included, bringing security uncertainty. Then in the back-end house, the gate-level HDL is transformed to the layout netlist by considering more physical-level parameters. Finally, the layout files are given to the foundry to fabricate the final IC products.

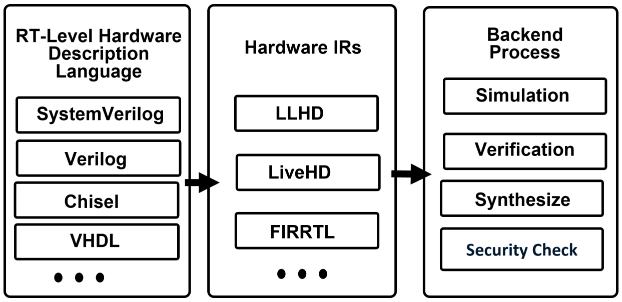

Figure 2: Examples of hardware IRs in design flow.

This paper assumes that the adversary can insert malicious logic into the design stage Functional-level HDL. We assume that adversaries may be the rogue agent at either the Front-end Design House or the third-party IP vendors who can access the hardware description language (HDL) code and insert a hardware Trojan or backdoor to manipulate critical registers of the design. Such a Trojan can be triggered by the conditions such as a counter, a predetermined time, an input vector, or certain physical conditions.

2.2 Intermediate Representations for Hardware

An abstract syntax tree (AST) is a representation of the syntactic structure of the source code. Within the AST, each node represents an operation in the source code. Tools that deal with programming languages, such as compilers, assemblers, synthesis tools, and analyzers, do not directly work on the source code. Instead, these tools often take the AST as the initial representation of the code to be processed before any transformations or analysis are performed. To generate the AST of any source code, there are three steps. First, a lexer converts elements in the source code stream into tokens. Then a grammar definition is used to process the stream of tokens ensuring that the syntactical rules are met. Lastly, the token stream is rearranged and converted into a tree structure. The final AST is generated after eliminating redundant extras. The equivalent representations of AST that is readable for human beings are the intermediate representations (IR).

2.3 Hardware IRs in Design Flow

Nowadays, high-level synthesis technologies and SystemVerilog are utilized to rapidly develop large-scale SoCs. In addition, many IR frameworks have been developed to improve the flexibility in verifying and testing these hardware designs. The latest examples of hardware-oriented IRs include FIRRTL [19], LLHD [32], LiveHD [39–41], CoreIR [26], and uIR [33]. These IRs aim to become the critical infrastructure of the hardware design flow and present an ecosystem, as summarized in Figure 2.

In the ecosystem, developers use hardware description language to design the circuit in RT-Level, such as SystemVerilog, Chisel, or higher abstractions like C++. Then the RT-Level descriptions are translated to these hardware IRs, that is shareable across multiple designers and EDA tools. Based on the IRs, simulations, formal verifications, synthesizing and security checking can be carried out. Open-source tools are involved in these design flows, like the LLVM compiler [23], Yosys [42], Verilator simulator [34], ABC [28], and SAT/SMT solvers [11].

As an example, developed in [32], LLHD is a hardware IR to represent a finished hardware RT-Level design. It is claimed to support a wide range of languages, including Chisel, SpinalHDL, MyHDL, SystemVerilog, and VHDL. It covers most back-end processing such as simulation, test-benches/formal verification, behavioral and structural modeling, gate-level netlist synthesis. However, most of the optimization and transformation functions are still in development. Therefore, only a small subset of its ecosystem has been implemented. LiveHD is another open-source framework developed to build scalable hardware designs [40]. It mainly applies two hardware IRs: LNAST [41] and LGraph [39]. So far, the LiveHD is still under construction. Due to the ambitious objective, these hardware IRs still need time to connect with other RTL languages and existing electronic design automation (EDA) tools.

2.4 Security Applications of Hardware IRs using Lightweight Parsers

Meanwhile, unlike the above "ambitious" hardware IR frameworks, some lightweight frontends are developed to pre-process the HDL program, such as Veriable [8], Surelog [9], Slang [29], as well as PyVerilog [36]. These tools parse either Verilog or SystemVerilog design to their self-defined IRs. Accordingly, more and more studies are proposed to analyze or formally verify hardware designs using these lightweight parsers to finish the security check at a specific domain. These methods include QIF-Verilog [13], RTSEC [5], If-Tracker [25], QFlow [30], HW2VEC [44], GNN4TJ [43], Hardware Fuzzing [37], Coppelia [45], etc. We summarize the workflow of these security applications as shown in Figure 3.

As the most popular hardware description language, Verilog is utilized as the source language for many well-developed EDA parsers. For instance, PyVerilog is a Python-based parser that translates Verilog designs to IRs using a self-defined syntax and then provides data flow analysis between two associated/connected signals. Then the data flow graph (DFG) of the Verilog design can be generated for further analysis and verification. In QIF-Verilog and QFlow, the quantitative information flow metrics are designed and added as a feature to the DFG to evaluate the information leakage. In terms of HW2VEC and GNN4TJ, the DFG is utilized for training the model and then detecting security vulnerabilities. Although DFG is employed effectively in the above methods, it is only a data structure subset extracted from IRs. IF-Tracker and RTSEC perform analysis based on the IRs directly to detect more sophisticated behaviors. For example, the malicious IPs' interactions over the bus in an SoC can be detected using IF-Tracker, while the trustworthy enhanced hardware is generated by adding watermarking and logic locking in RTSEC.

Very recently, methods have been proposed to translate Verilog codes into high-level descriptions in C++ or similar languages [18, 45], which allows for the use of existing software verification tools such as KLEE or Verilator [6, 34]. Although the Verilator is famous as an open-source simulator, it has become a popular Verilog parser. By applying Verilator to convert a hardware design to C++ code, Coppelia performs security checking on the equivalent C++ program [27]. HW-Fuzzing uses Verilator to acquire the equivalent model of the C++ language for Verilog code, based on which the traditional software fuzzing is performed [37]. In addition, symbolic execution engines such as KLEE or Fuzz tools such as AFL are utilized in these approaches, respectively.

Compared with Verilog, it is more challenging to develop an open-source parser for SystemVerilog because of its more complex syntax. Slang and Surelog are two of the stable maintained SystemVerilog parser. Specifically, Surelog is capable of transforming SystemVerilog design to IRs that can be readable by Yosys. In this paper, the proposed NN-based Trojan detection framework is based on the AST graph mapped from Surelog IRs by Yosys.

2.5 Learning based Trojan Detection Approach

Over the last several decades, advances in machine learning (ML) have reshaped established methodologies and models for various design applications. Several machine learning detection algorithms effectively detect HT in hardware designs, either at the register transfer level (RTL)[16] or at the gate-level netlisting (GLN)[17].

While typical machine learning approaches provide very desired results applied to Euclidean data, they are often severely performed for non-Euclidean data. The graphs defining the hardware design, on the other hand, are not Euclidean[7], which challenges utilizing these approaches. The notion of graph deep learning has been developed to address NP-complete issues in graph matching. Furthermore, other graph learning techniques, including Graph Convolutional Networks (GCN)[21], Graph Neural Networks (GNN)[35], and Graph Autoencoder (GAE)[38], partially solve the challenge of evaluating non-Euclidean data. These works simultaneously provide tools for the future development of machine learning approaches in hardware security.

As an example, HW2VEC[44] is proposed as a graph deep learning method to analyze the flow of hardware data on this foundation. This work is on the netlist layer that detects the existence of hardware Trojans. On the other hand, a more extensive data scale and a lower abstraction level in the netlist layer increase the computation complexity of applying HW2VEC. Additionally, the PyVerilog component is utilized in HW2VEC as a front-end with severe restrictions – it cannot take SystemVerilog files and is inefficient when accepting large instances. Furthermore, this study does not solve the issue of insufficient data to support network learning. Therefore, we suggest our novel approach in an attempt to solve these three constraints.

3 METHODOLOGY

The overall architecture of the proposed approach is demonstrated in the Fig. 4. Beginning with hardware RTL design in SystemVerilog, Surelog parses the design in one UHDM model. With the help of UHDM Yosys integration, Yosys transforms the UHDM to IR and generate a dot file for hardware structure. The Networkx python package reads the dot file and provides an API to manipulate the graph and export a graph. Graph Neural Network Framework Spektral can be employed to construct the model. Specifically, the approach is split into five components as follows.

(1) **Parse the SystemVerilog design:** Surelog, an ANTLR-based parser, is used as a front-end to parse the SystemVerilog hardware design into a Universal Hardware Data Model

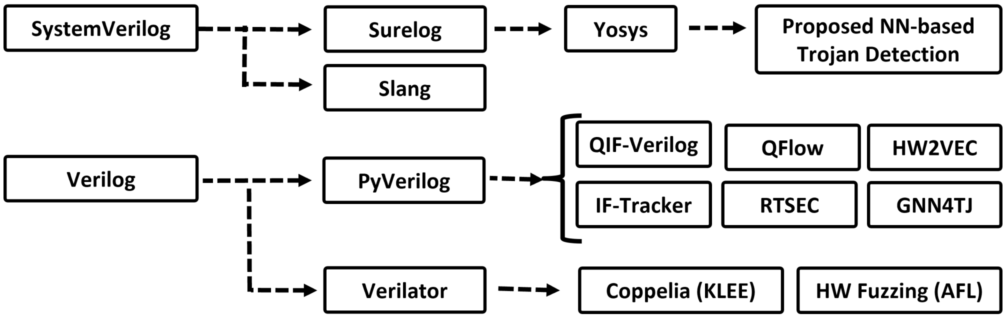

Figure 3: Applications of lightweight parsers in design flow.

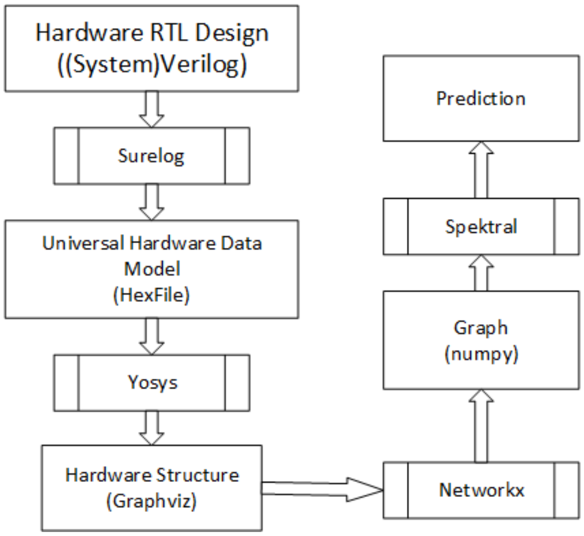
Hardware RTL Design ((System)Verilog)
Surelog
Universal Hardware Data Model (HexFile)
Yosys
Hardware Structure (Graphviz)
Networkx
Graph (numpy)
Spektral
Prediction

Figure 4: Overall architecture of the proposed approach.

(UHDM). This file includes AST consisting of the Verilog Procedural Interface (VPI) [10] .

(2) **Create a schematic of the hardware structure:** Yosys[42] receives the UHDM file and automatically translates it to the AST called RTLIL. Then RTLIL is converted to a hardware structure diagram by yosys.
(3) **Pre-processing the schematic of the hardware structure:** A merge script is created to combine several Yosys diagrams inside the hardware design.
(4) **Transform:** Graphviz graphs are transformed into a structure that the neural network can handle.
(5) **Classify:** Classification via Graph Residual Network.

3.1 Parse the Hardware Design

The raw data from the whole hardware design is parsed to a UHDM model in the first step. Surelog, a multi-threaded SystemVerilog preprocessor and parser, is the front-end of our approach, supporting the full SystemVerilog2017 standard. Due to the compiler's Antrl-based architecture, the recursive-descent parse technique makes it one of the finest open-source tools for SystemVerilog. At the moment, the compiler can compile any synthesizable subset of SystemVerilog. Before Surelog preprocesses the hardware, we need to create a flist that contains the entire hardware design. Surelog generated UHDM is very similar to SystemVerilog's Object Model.

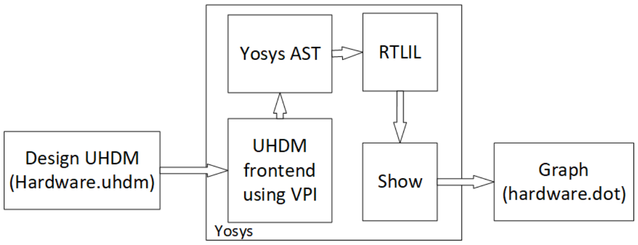

Figure 5: The UHDM-YOSYS flow. The UHDM integration[4] read the UHDM model via VPI, the AST is immediately rewritten into Yosys' internal AST format, and Yosys generates its own internal IR from the portions that can be synthesized. Meanwhile, the show command could be used to get a graphviz representation of the hardware topology.

Therefore, the simulator and other tools inside the toolchain can access the data using the Verilog Procedural Interface(VPI).

3.2 Hardware Schematic Creation

In contrast to the conventional hardware security approach, neither data-flow nor control-flow graph are employed directly. As in Fig.5, Yosys' function is utilized to generate schematics directly from the synthesizable section of the IR (RTLIL) Yosys relied on, which contains the hardware's topology. We presume that the hardware Trojan is inserted in the hardware design. Unlike other simulation-based techniques, we aim to utilize a static method to reduce the amount of data in the approach to avoid scalability concerns. Classification results could be acquired straight from static inspection. Moreover, the proposed work is at the RT level, which further eliminates the scalability issues. Yosys creates a graph for each module and the nodes are named evenly. Accordingly, a script is developed to combine all the diagrams for the whole hardware design into a giant diagram containing all hardware topological information.

3.3 Graphviz Graph Transformation

After acquiring the hardware design in Graphviz format, we have to transform it into a graph to further simplify the computing work. We accomplish this goal by utilizing Networkx [15], a Python library for constructing, manipulating, and analyzing complicated networks. A variety of well-known geometric representation learning libraries (PyTorch-Geometric, Deep Graph Library, and Spektral) could accept networkx-generated graphs as input. However, after being transformed to networkx graphs, the graphviz files generated by Yosys preserve additional information. This additional information is omitted at this stage because the processed data retains the hardware's structure and could be put into the graph neural network pipeline.

3.4 Handling of Hardware Topology

Due to the lack of datasets in hardware security, deep learning approaches have yet to be ideal trained. Different IP core has naturally distinct design structures, and it is difficult to migrate Trojan features from one IP core to another. While some netlist-level work could extract Trojan characteristics due to device homogeneity, falling into the scalability issue at the netlist level would use much more time. This stage introduces the notion of graph spectral domain and residual learning. The graph model obtained from the hardware design is also known as a spatial domain; however, since the number of node neighbors in this domain is not always equal, we transform it into a spectral domain to complete the convolution process. Specifically, the hardware represented in the spatial domain is converted to the spectral domain by utilizing the Fourier normal transform of the graph.

In order to determine the impact of the Trojan's implantation on the hardware topology spectrum, the proposed method performs subtraction between the Trojan-in and Trojan-free spectral matrices. We intend to exploit the graph neural network in this approach to extract Trojan's features, enabling it to be used on various hardware architectures. Furthermore, the proposed method generates datasets that span many different IP cores, which solves the lack of datasets.

3.5 Graph Neural Network Model

We employ spektral[12], a framework for graph deep learning, and refer to previous work for molecular protein prediction.

We are now building a three-layer convolutional graph network. Each layer of the model generates a new node representation by aggregating neighbor information, by selecting the average of a graph's node features. The training loop repeatedly trains and computes gradients using graph loader objects, similar to image classification or language modeling.

This work is still ongoing.

4 EXPERIMENTAL RESULTS

The overall purpose of this paper is to provide Trojan detection for a SystemVerilog-based SoC. As a result, our testbench[3] is an SoC equipped with an Ibex core, an open-source 32-bit RISC-V CPU core written in SystemVerilog. Additionally, we link an AES 128 module[31] via the AXI bus. The experiment is on a computer with an Intel(R) Core(TM) i7-4770 CPU, 16GB RAM, and Ubuntu 20.04.1 as the operating system. The compilation procedure took 20 seconds in total.

4.1 From IR to Graph

We develop code to process these graphs efficiently. In the transformation, superfluous attributes are deleted from the dot file. While in the current approach, conversion from the directed graph to an undirected graph is done in the next step due to the graph's Laplace transform's validity. At this stage, we maintain the orientation of the edges. Fig.6 indicates the hardware architecture of AES-T100 from Trust-hub after the Trojan's insertion.

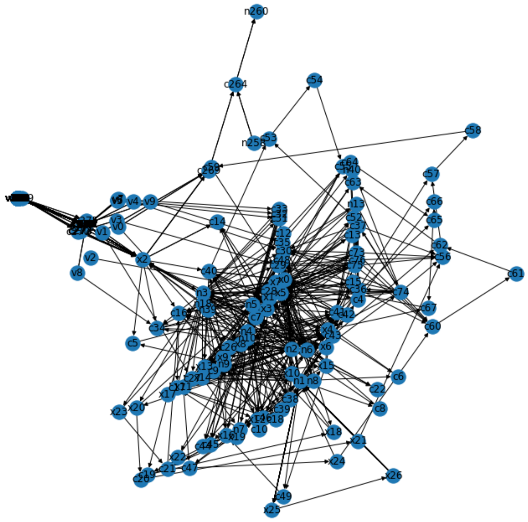

Figure 6: Diagram showing the hardware architecture of the AES-T100 from the trust-hub after Trojan injection.

Following that, the data are converted to spectral space. Then a graph neural network is used to analyze the data. The results will be demonstrated in our future work.

5 DISCUSSIONS

We plan to complete the experiments with the following aspects in future work. First of all, the UHDM model generated by Surelog has the potential to cause core dumps in Yosys and Verilator, which are the only applications that presently support the UHDM model. In subsequent work, we may bypass these two software and directly acquire the hardware topology via UHDM. Second, we tried to refer to the molecular prediction network. Molecules, on the other hand, are very distinct from hardware. For example, hardware always includes more types of nodes, more edges from single nodes, and a far larger overall design size than molecules, necessitating the reworking of the layer and network structure. Finally, the proper Fourier basis has to be investigated experimentally.

6 CONCLUSIONS

To secure a RISC-V based SoC design, we present an NN-based hardware vulnerability detection framework. We extend the tool's functionality by employing Surelog as a SystemVerilog front-end, and then evaluate the hardware architecture using graph neural networks to perform a static analysis of hardware security. Simultaneously, we use residual learning to minimize the influence of hardware functions on the analysis, hence increasing the size of the training database. The experimental section of this work is still under construction, and more completed demonstrations will be given in our future work.

ACKNOWLEDGMENTS

Portions of this work were supported by the National Science Foundation (CCF-2019310, CCF-2028910, and CCF-2019283.).

REFERENCES

[1] 2021. HACK at DAC. Retrieved December 5-9, 2021 from https://hackatevent.org/hackdac21/

[2] 2022. Common Weakness Enumeration. https://cwe.mitre.org/

[3] lowRISC . 2022. GitHub - lowRISC/ibex: Ibex is a small 32 bit RISC-V CPU core, previously known as zero-riscy. [Online; accessed 2022-04-10].

[4] antmicro. 2022. GitHub - antmicro/yosys-uhdm-plugin-integration. https://github.com/antmicro/yosys-uhdm-plugin-integration

[5] Orlando Arias, Zhaoxiang Liu, Xiaolong Guo, Yier Jin, and Shuo Wang. 2022. RTSec: Automated RTL Code Augmentation for Hardware Security Enhancement. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE.

[6] Cristian Cadar, Daniel Dunbar, Dawson R Engler, et al. 2008. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs.. In *OSDI*, Vol. 8. 209–224.

[7] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1616–1637.

[8] Chipsalliance. 2019. Verible project. Retrieved April 5, 2022 from https://github.com/chipsalliance/verible/

[9] chipsalliance. 2022. GitHub - chipsalliance/Surelog: SystemVerilog 2017 Pre-processor, Parser, Elaborator, UHDM Compiler. Provides IEEE Design/TB C/C++ VPI and Python AST API. Compiles on Linux gcc, Windows msys2-gcc & msvc, OsX. https://github.com/chipsalliance/Surelog

[10] C. Dawson, S.K. Pattanam, and D. Roberts. 1996. The Verilog Procedural Interface for the Verilog Hardware Description Language. In *Proceedings. IEEE International Verilog HDL Conference*. 17–23. https://doi.org/10.1109/IVC.1996.496013

[11] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.

[12] Daniele Grattarola and Cesare Alippi. 2020. Graph Neural Networks in TensorFlow and Keras with Spektral. *CoRR* abs/2006.12138 (2020). arXiv:2006.12138 https://arxiv.org/abs/2006.12138

[13] Xiaolong Guo, Raj Gautam Dutta, Jiaji He, Mark M Tehranipoor, and Yier Jin. 2019. 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, 91–100.

[14] Xiaolong Guo, Huifeng Zhu, Yier Jin, and Xuan Zhang. 2019. When capacitors attack: Formal method driven design and detection of charge-domain trojans. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1727–1732.

[15] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX*. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[16] Tao Han, Yuze Wang, and Peng Liu. 2019. Hardware Trojans detection at register transfer level based on machine learning. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[17] Kento Hasegawa, Youhua Shi, and Nozomu Togawa. 2018. Hardware Trojan detection utilizing machine learning approaches. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 1891–1896.

[18] Jiaji He, Xiaolong Guo, Travis Meade, Raj Gautam Dutta, Yiqiang Zhao, and Yier Jin. 2019. SoC interconnection protection through formal verification. *Integration* 64 (2019), 143–151.

[19] Adam Izraelevitz, Jack Koenig, Patrick Li, Richard Lin, Angie Wang, Albert Magyar, Donggyu Kim, Colin Schmidt, Chick Markley, Jim Lawson, et al. 2017. Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 209–216.

[20] Himanshu Jain, Daniel Kroening, Natasha Sharygina, and Edmund M Clarke. 2008. Word-level predicate-abstraction and refinement techniques for verifying RTL Verilog. *IEEE transactions on computer-aided design of integrated circuits and systems* 27, 2 (2008), 366–379.

[21] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[22] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. 2019. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1–19.

[23] Chris Lattner and Vikram Adve. 2004. LLVM: A compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004*. IEEE, 75–86.

[24] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown. *arXiv preprint arXiv:1801.01207* (2018).

[25] Zhaoxiang Liu, Orlando Arias, Weimin Fu, and Yier Jin. 2022. Inter-IP Malicious Modification Detection through Static Information Flow Tracking. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE.

[26] Cristian Mattarei, Makai Mann, Clark Barrett, Ross G Daly, Dillon Huff, and Pat Hanrahan. 2018. CoSA: Integrated verification for agile hardware design. In *2018 Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 1–5.

[27] Xingyu Meng, Shamik Kundu, Arun K Kanuparthi, and Kanad Basu. 2021. RTL-ConTest: Concolic Testing on RTL for Detecting Security Vulnerabilities. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2021).

[28] Alan Mishchenko et al. 2007. ABC: A system for sequential synthesis and verification. *URL http://www. eecs. berkeley. edu/alanmi/abc* 17 (2007).

[29] Michael Popoloski. 2019. Slang - SystemVerilog Language Services. Retrieved April 3, 2022 from https://github.com/MikePopoloski/slang/

[30] Lennart M Reimann, Luca Hanel, Dominik Sisejkovic, Farhad Merchant, and Rainer Leupers. 2021. QFlow: Quantitative Information Flow for Security-Aware Hardware Design in Verilog. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*. IEEE, 603–607.

[31] Hassan Salmani, Mohammad Tehranipoor, and Ramesh Karri. 2013. On design vulnerability analysis and trust benchmarks development. In *2013 IEEE 31st international conference on computer design (ICCD)*. IEEE, 471–474.

[32] Fabian Schuiki, Andreas Kurth, Tobias Grosser, and Luca Benini. 2020. LLHD: A multi-level intermediate representation for hardware description languages. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 258–271.

[33] Amirali Sharifian, Reza Hojabr, Navid Rahimi, Sihao Liu, Apala Guha, Tony Nowatzki, and Arrvindh Shriraman. 2019. μir-an intermediate representation for transforming and optimizing the microarchitecture of application accelerators. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 940–953.

[34] Wilson Snyder. 2013. Verilator: Open simulation-growing up. *DVClub Bristol* (2013).

[35] Alessandro Sperduti and Antonina Starita. 1997. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks* 8, 3 (1997), 714–735.

[36] Shinya Takamaeda-Yamazaki. 2015. Pyverilog: A python-based hardware design processing toolkit for verilog hdl. In *International Symposium on Applied Reconfigurable Computing*. Springer, 451–460.

[37] Timothy Trippel, Kang G Shin, Alex Chernyakhovsky, Garret Kelly, Dominic Rizzo, and Matthew Hicks. 2021. Fuzzing hardware like software. *arXiv preprint arXiv:2102.02308* (2021).

[38] Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang. 2017. Mgae: Marginalized graph autoencoder for graph clustering. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 889–898.

[39] Sheng-Hong Wang, Rafael Trapani Possignolo, Qian Chen, Rohan Ganpati, and Jose Renau. 2019. LGraph: A unified data model and API for productive open-source hardware design. In *Proc. 2nd Workshop Open-Source EDA Technol.*

[40] Sheng-Hong Wang and Jose Renau. 2021. Design Decisions in LiveHD for HDLs Compilation. (2021).

[41] Sheng-Hong Wang, Akash Sridhar, and Jose Renau. 2019. LNAST: A language neutral intermediate representation for hardware description languages. In *Proc. 2nd Workshop Open-Source EDA Technol.*

[42] Clifford Wolf. 2016. Yosys open synthesis suite.

[43] Rozhin Yasaei, Shih-Yuan Yu, and Mohammad Abdullah Al Faruque. 2021. Gnn4tj: Graph neural networks for hardware trojan detection at register transfer level. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1504–1509.

[44] Shih-Yuan Yu, Rozhin Yasaei, Qingrong Zhou, Tommy Nguyen, and Mohammad Abdullah Al Faruque. 2021. HW2VEC: A Graph Learning Tool for Automating Hardware Security. *arXiv preprint arXiv:2107.12328* (2021).

[45] Rui Zhang, Calvin Deutschbein, Peng Huang, and Cynthia Sturton. 2018. End-to-end automated exploit generation for validating the security of processor designs. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 815–827.