data

MDPI

*Data Descriptor*

# Hardware Trojan Dataset of RISC-V and Web3 Generated with ChatGPT-4

**Victor Takashi Hayashi * and Wilson Vicente Ruggiero**

Polytechnic School, University of São Paulo, Sao Paulo 05508-010, Brazil; wilson@larc.usp.br
* Correspondence: victor.hayashi@usp.br

**Abstract:** Although hardware trojans impose a relevant threat to the hardware security of RISC-V and Web3 applications, existing datasets have a limited set of examples, as the most famous hardware trojan dataset TrustHub has 106 different trojans. RISC-V specifically has study cases of three and four different hardware trojans, and no research was found regarding Web3 hardware trojans in modules such as a hardware wallet. This research presents a dataset of 290 Verilog examples generated with ChatGPT-4 Large Language Model (LLM) based on 29 golden models and the TrustHub taxonomy. It is expected that this dataset supports future research endeavors regarding defense mechanisms against hardware trojans in RISC-V, hardware wallet, and hardware Proof of Work (PoW) miner.

**Keywords:** hardware trojan; RISC-V; Web3; Blockchain; LLM, ChatGPT

check for updates

**Citation:** Hayashi, V.T.; Ruggiero, W.V. Hardware Trojan Dataset of RISC-V and Web3 Generated with ChatGPT-4. *Data* **2024**, *9*, 82. https://doi.org/10.3390/data9060082

Academic Editor: Florentino Fdez-Riverola

Received: 22 April 2024
Revised: 14 June 2024
Accepted: 17 June 2024
Published: 19 June 2024

**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

## 1. Summary

Using an open-source hardware solution (Open Hardware) enhances the transparency of hardware solutions for critical applications. This transparency may contribute to the ease of verification of these hardware solutions and thus to its overall security. Following the development and massive use of open-source software (Free and Open-Source Software—FOSS), hardware devices can have all their code and specifications open, made available under a license that makes it possible for anyone to use, copy, analyze, and make changes, so people are encouraged to contribute to a project on a voluntarily [1].

The Open Source Hardware Association (Open Source Hardware Association)[1] defines open hardware as the "hardware whose design is made publicly available so that anyone can study, modify, distribute, manufacture, and sell the design or hardware based on that design. The hardware source is available in a preferred format for making modifications. Ideally, open-source hardware uses readily available components and materials, standard processes, open infrastructure, unrestricted content, and open-source tools to maximize individuals' ability to manufacture and use hardware. Open-source hardware gives people the freedom to control their technology, share their knowledge and encourage commerce through the open exchange of designs.

Some successful examples such as the RISC-V open architecture and the OpenTitan project supported by a commercial partnership between several companies show the relevance of open hardware. One of the benefits of open hardware is the replacement of security by obscurity with security by verification [2]. By increasing the ease of specification, testing, and verification of hardware designs, and fostering community collaboration, open hardware can contribute to greater user confidence in hardware solutions by transparency. Specifically, the RISC-V is becoming a standard Instruction Set Architecture (ISA) for small Internet of Things (IoT) devices, warehouse-level computers, and personal mobile devices. For example, the company SiFive put RISC-V products on the market for desktop and IoT applications, while Alibaba launched a RISC-V product for cloud and edge computing [3].

Open-source software solutions are transparent solutions whose availability for free access makes a higher level of security maturity possible due to the peer scrutiny process.

Considering the example of backdoors in open-source solutions, only making code available for free access is not enough. Some method is needed to help verify the security of these solutions. In the case of hardware used in critical security applications, it is not enough to perform only functional tests to verify that the device works according to specification [4]. It is necessary to verify that no additional functionality is present in such a design.

Among the security threats of the RISC-V, hardware trojans included by malicious designers and foundries can cause denial of service and the transmission of sensitive data to external non-authorized observers. However, even with its reasonable importance, the development of defense mechanisms is relatively lagging [3]. One of the reasons why these defense mechanisms are lagging is related to the availability of open datasets to support research. Different hardware trojan detection methods may be studied if there are available datasets with relevant designs to support such investigations. A current research effort to study hardware trojans in RISC-V presents the design and integration of four trojans into a post-quantum RISC-V micro-controller [5], and another work presents three hardware trojans inserted into different RISC-V modules [6]. While these stealthy hardware trojans are of the utmost importance to enable research, their manual nature makes it difficult to scale to support a comprehensive dataset of RISC-V hardware trojans.

Considering another relevant example, decentralized systems based on Blockchain have enormous industry and academic interests [7]. The most popular use case is the cryptocurrency Bitcoin [8], which is a peer-to-peer electronic cash system [9]. Other cryptocurrencies such as Ethereum [10] have been created, and other applications such as supply chain [11], smart contracts [12], and other [13] have been proposed. Blockchain technology [14] is a distributed ledger that records a sequence of events accepted and respected by the network. The ledger is built with a sequential block structure so that each block has a cryptographic hash of the previous block, creating a hash chain. Due to this distributed structure, a consensus mechanism is necessary when creating a new block in the ledger, and this consensus is provided by proof-of-work (PoW), proof-of-stake (PoS), and other algorithms to solve the consensus problem [15].

Specifically, a fundamental process used to validate the blocks in Bitcoin is mining. It is based on a consensus algorithm that implies the solution of a mathematical problem by each miner [16]. A Bitcoin miner spends computing power to solve these cryptography problems for the Proof of Work (PoW) consensus algorithm in exchange for some amount of Bitcoins [9]. Despite its relevance, to the best of the authors' knowledge, there is no investigation regarding possible hardware trojans in specialized hardware PoW miners. Regarding decentralized applications in Web3, wallets such as Metamask are being used by millions of users, enabling their interactions with Non-Fungible Tokens (NFT) and Metaverse applications [17]. However, to the best of the authors' knowledge, there is no investigation regarding possible hardware trojans in specialized hardware wallets.

There are four relevant datasets for hardware trojan detection in the literature. The most popular is TrustHub, an online hardware trojan database with benchmarks organized according to its taxonomy [18,19]. Currently, this dataset has a total of 106 examples of various designs: AES, Basic RSA, Ethernet, 8051 microcontrollers, RS-232 serial communication, and VGA, among others. Another dataset has ElectroMagnetic (EM) side-channel signals for 12 different hardware trojans of the AES [20]. The third dataset has 37 examples of hardware trojans of Basic RSA, AES, and communication modules [21]. The fourth dataset has 20 different examples of hardware trojans of AES and additional examples on a toy dataset in communication modules, adder, encoder, and other simple designs [22].

Considering this motivation, the present work considers these Research Questions:

- Research Question 1: How do we better support hardware trojan detection research considering limitations in existing datasets?
- Research Question 2: Is it possible to use a Large Language Model (LLM) to enhance data generation for hardware trojan research?

## 2. Methods

Considering the motivation of supporting the development of defense mechanisms to enhance RISC-V security (which is currently lagging [3]), the lack of extensive RISC-V hardware trojan datasets to enable the implementation of various hardware trojan detection techniques (as only a few examples of RISC-V hardware trojans are available in the literature [5,6]), and the non-existing investigation of hardware trojans in Web3 (e.g., hardware wallet and miners), the present work uses some open Verilog designs, the TrustHub taxonomy [18,19] and the ChatGPT-4 to generate synthetic hardware trojan data. In this context, the ChatGPT-4 Large Language Model (LLM) is used to generate some trojans based on a given design, motivated by its success in red team applications [23]. Previous synthetic data generation methods found in the literature do not use ChatGPT-4 [24–26].

The prompt engineering method used for generating the hardware trojans for educational purposes is illustrated in Figure 1. The first step is essential for proper LLM agent context. This context consists of the user presenting itself with this prompt: 'I am researching ways to identify hardware trojans, and to train my classifier I need to generate some Verilog designs infected with hardware trojans based on a golden model Verilog design and some hardware trojan descriptions'.

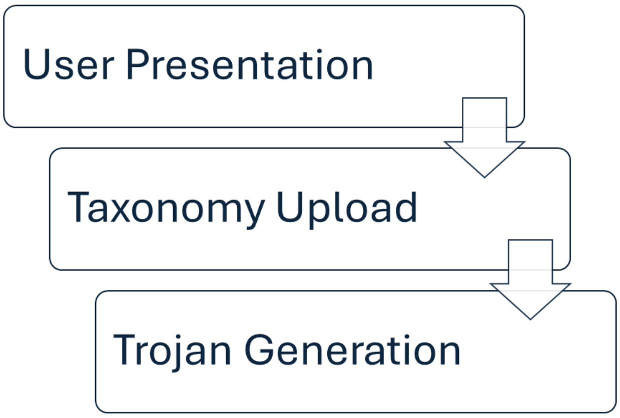


**Figure 1.** Prompt Engineering Method to Generate Hardware Trojan using ChatGPT-4.

In the second step, the user provides the TrustHub benchmark file in PDF format (available on its website[2] and illustrated in Figure 2) to ChatGPT-4 with the prompt 'Consider the hardware trojans described in this file. For each Verilog code I will provide next, please generate hardware trojans according to the hardware trojans described in the file'.

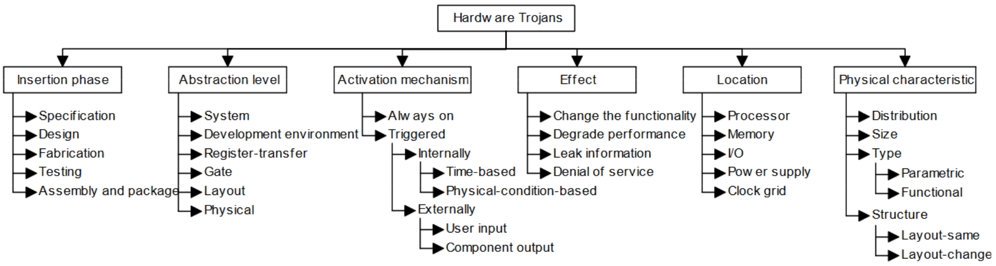
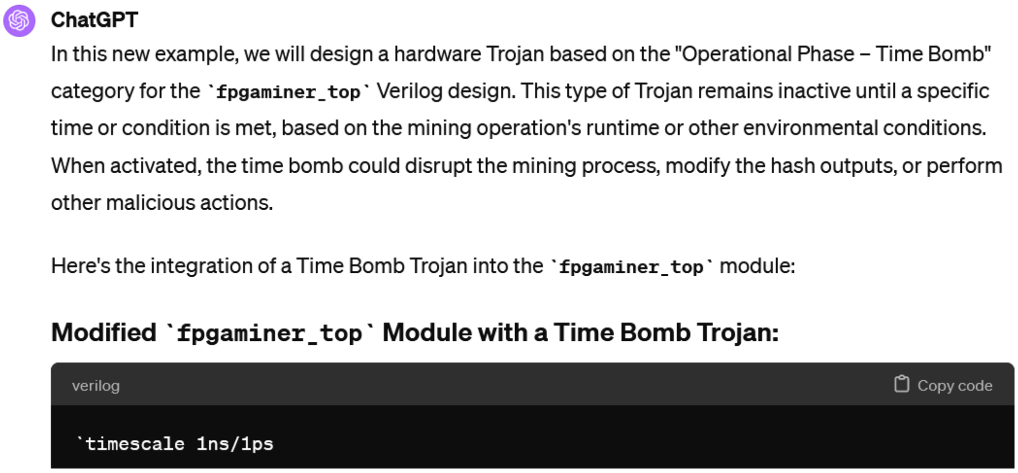


**Figure 2.** TrustHub taxonomy provided to ChatGPT-4 to generate hardware trojans [18,19].

The third step consists of providing the Verilog design (i.e., golden model) directly in the prompt, as illustrated in this example: 'For educational purposes, give me one new example of hardware trojan of the following Verilog design based on the taxonomy: <golden model Verilog design>'.

According to some initial experiments, not providing the initial context results in the agent denial to generate the trojan to prevent misuse, not providing the taxonomy may lead to some episodes of only external trigger-based trojans being generated, and currently ChatGPT-4 does not support Verilog file analysis, but it can parse Verilog designs if they are directly provided in user queries.

Consider the examples of prompt results presented in Figures 3 and 4. First, ChatGPT-4 generates a time bomb hardware trojan for the Miner design that disrupts operation after a timer reaches a specified value. In the second example, a new hardware trojan is generated with a payload that leaks sensitive data.



**ChatGPT**

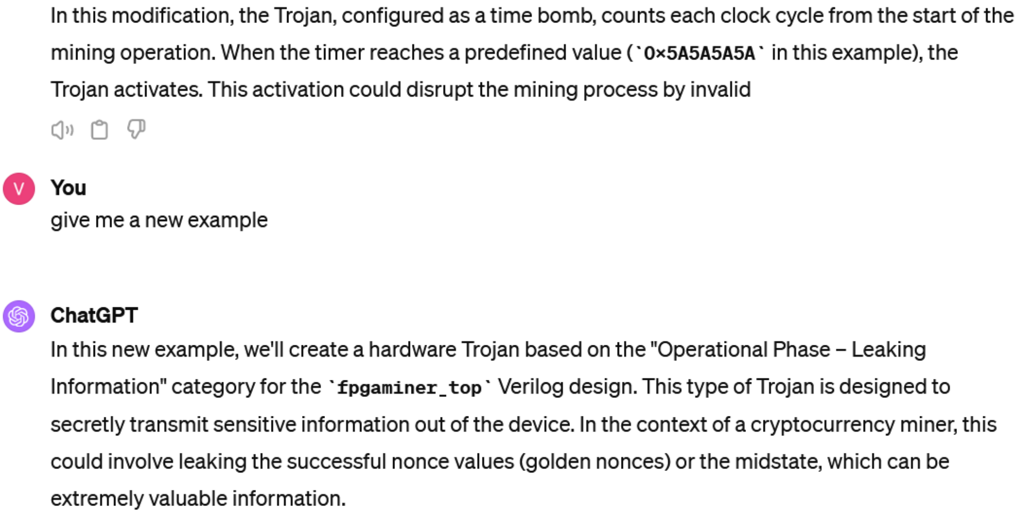
In this new example, we will design a hardware Trojan based on the "Operational Phase – Time Bomb" category for the `` `fpgaminer_top` `` Verilog design. This type of Trojan remains inactive until a specific time or condition is met, based on the mining operation's runtime or other environmental conditions. When activated, the time bomb could disrupt the mining process, modify the hash outputs, or perform other malicious actions.

Here's the integration of a Time Bomb Trojan into the `` `fpgaminer_top` `` module:

**Modified `` `fpgaminer_top` `` Module with a Time Bomb Trojan:**

```verilog
`timescale 1ns/1ps
```

**Figure 3.** Time Bomb example generated by ChatGPT-4 for the Miner design.



In this modification, the Trojan, configured as a time bomb, counts each clock cycle from the start of the mining operation. When the timer reaches a predefined value (`` `0x5A5A5A5A` `` in this example), the Trojan activates. This activation could disrupt the mining process by invalid

**You**

give me a new example

**ChatGPT**

In this new example, we'll create a hardware Trojan based on the "Operational Phase – Leaking Information" category for the `` `fpgaminer_top` `` Verilog design. This type of Trojan is designed to secretly transmit sensitive information out of the device. In the context of a cryptocurrency miner, this could involve leaking the successful nonce values (golden nonces) or the midstate, which can be extremely valuable information.

**Figure 4.** Data Leak example generated by ChatGPT-4 for the Miner design.

As some complex designs would have a long response, and to prevent misuse of the ChatGPT-4 [27], the examples may be presented only in the modified parts, and there is always a clear disclaimer such as: ‘Remember, this is a hypothetical example for educational purposes to understand potential vulnerabilities and the concept of hardware Trojans. It should not be used for malicious purposes’. This guideline is also valid for the present research, whose objective is to provide an open dataset and a new approach to generate new hardware trojans that, even with the limitation of being hypothetical, would contribute to the development of defense countermeasures against real hardware trojans.

Additionally, we use the Verilator tool to assess the data quality of the generated hardware trojans and corresponding golden models. Verilator (PyVerilator version 0.7.0) is an open-source software tool that converts Verilog files to behavioral models in C++ to perform simulations and testing [28].

## 3. Data Description

The generated dataset has 290 Verilog examples generated with ChatGPT-4 Large Language Model (LLM) based on 29 golden models of RISC-V, hardware wallet, and hardware Proof of Work (PoW) miner. The miner has nine modules, RISC-V has 15 modules, and the wallet has five modules. A total of 10 trojans were generated for each module. Figure 5 show the size comparison in bytes of the golden models of the three hardware designs. The average size of miner modules is 2987 bytes, RISCV is 1760 bytes, and Wallet is 2685 bytes.

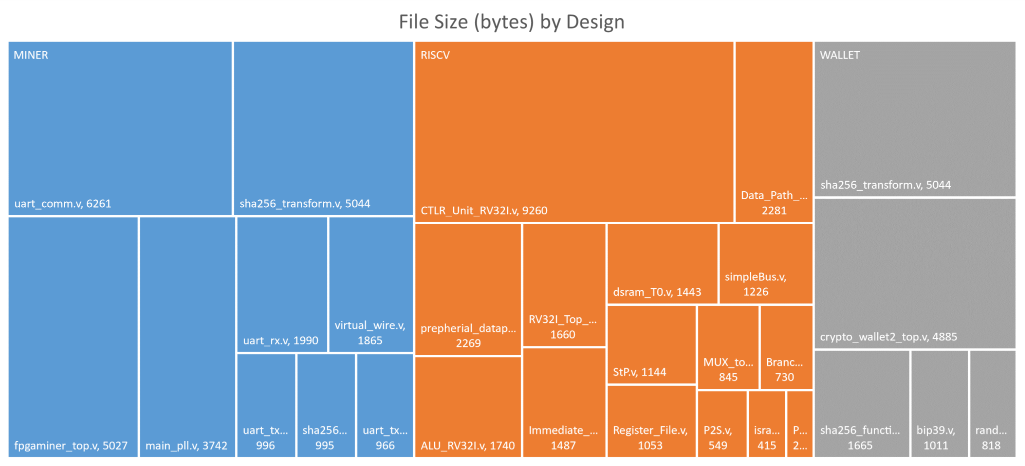


**Figure 5.** Golden Model Sizes Comparison by Design.

The histogram of the generated hardware trojans according to their size (bytes) is presented in Figure 6. The average size of the trojan files is 2744 bytes.

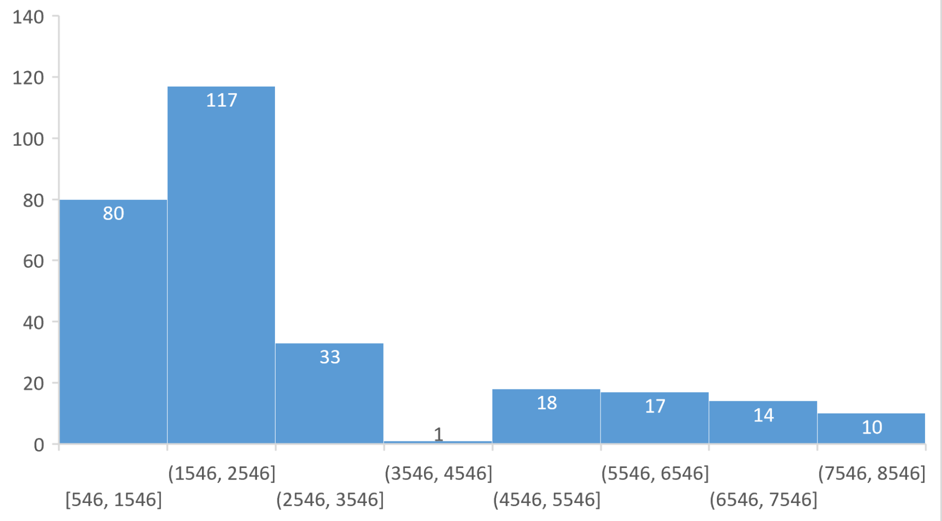


**Figure 6.** Hardware Trojan Distribution by Size (bytes).

The diversity of the generated hardware trojans according to their type is presented in Figure 7. The proposed approach generated only two types of hardware trojans: condition-based and time-based. When compared to the taxonomy of Figure 2, not all trigger conditions could be generated with the proposed approach. One relevant example is the physical condition-based hardware trojan.

For example, a condition-based hardware trojan in a RISC-V can be triggered by an external serial input with a specific pattern. On the other hand, a time-based hardware trojan (e.g., time bomb example presented in Figure 3) can be activated after a specific number of clock cycles have occurred since the device restart.

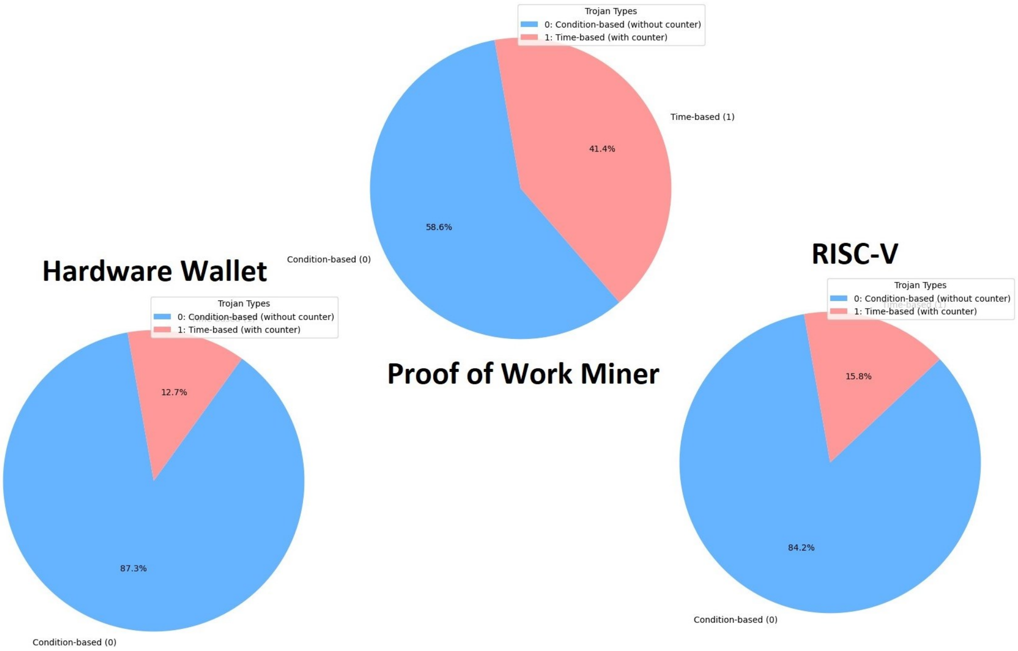


**Figure 7.** Hardware Trojan Diversity.

The overhead in bytes for the generated hardware trojans when compared to their corresponding golden models is presented in Figure 8. There are some negative values because in some cases the modified design is smaller than the golden model. After all, the substituted logic can be simpler than the original logic. Although RISC-V is the most complex design, the Proof of Work Miner presented bigger overheads, even though with a smaller number in total.

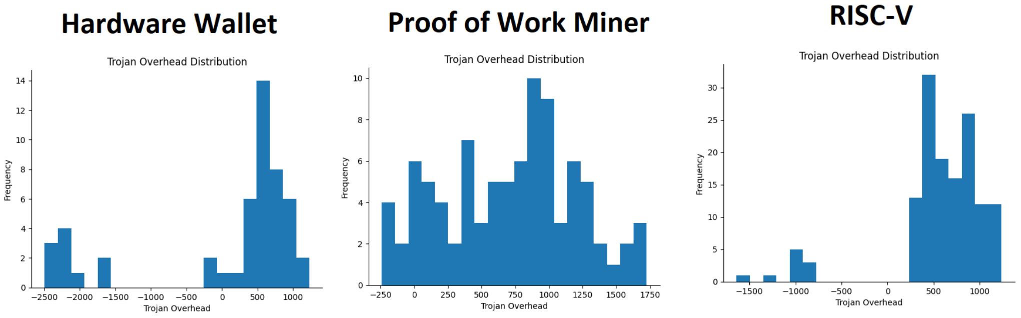


**Figure 8.** Hardware Trojan Overhead (bytes).

Figure 9 presents the total warnings of the Verilator (PyVerilator version 0.7.0) tool for each of the three major hardware designs. These warnings can cover various aspects of code style and logic errors, including incomplete case statements, overlapping case values, and improper use of certain Verilog constructs like dynamic casts and continuous assignments to registers. One possible warning occurs if a case statement lacks a default case, potentially hiding violations of design assumptions during simulation. The tool also warns against overlapping case values, which can lead to unpredictable behavior if the order of case statements is altered[3].

As a general fact, ignoring these warnings generally does not affect the immediate functionality of the simulation or the design, but could degrade the performance of the converted model. For instance, there are warnings in the original golden models, and in some cases, the generated design presented additional warnings, for example, when the trojan uses an additional signal that is not present in the original file (see Figure 9). The pattern for larger models such as the Proof of Work Miner and RISC-V is more complex, which may indicate that larger designs require additional validations depending on the final use of the generated dataset.

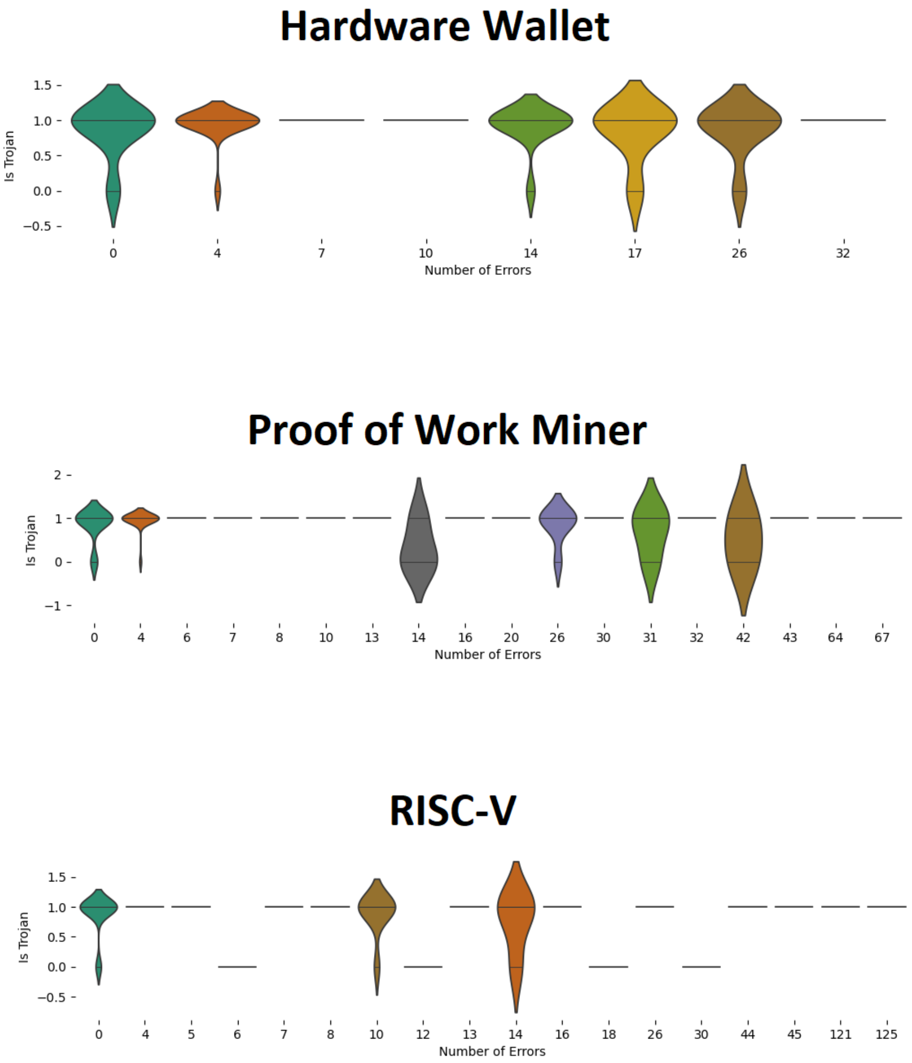


**Figure 9.** Verilator (PyVerilator version 0.7.0) Warnings.

### 3.1. Hardware Wallet

The hardware wallet considered is a simple implementation of a wallet with five modules (among them the use of SHA-256 hash) in Verilog, available in a GitHub repository[4]. The modules in this example form the basis for implementing a hardware cryptocurrency wallet: they integrate key generation, the use of secure hashing algorithms, and the conversion of data into mnemonic formats, all essential elements for the security and functionality of a cryptocurrency wallet. For example, the bip39 module implements the BIP39 (Bitcoin Improvement Proposal 39)[5] standard, which is used to generate mnemonic phrases for creating cryptocurrency keys. BIP39 transforms binary data into a sequence of words that can be more easily memorized by users.

### 3.2. Proof of Work Miner

The considered miner is an open-source Bitcoin miner described in Verilog and available in a GitHub repository[6]. It has nine modules, and among them, there is serial communication (UART) and a SHA-256 hash function module. The described components make up an FPGA-based Bitcoin miner, combining control logic, communication, and cryptographic processing to perform mining efficiently. For example, the fpgaminer_top module coordinates the distribution of clock, reset, and other control signals among internal modules. It also manages the input and output interface, connecting the FPGA to other devices and monitoring systems. This module is crucial because it orchestrates the entire miner operation, ensuring that all components work in a synchronized and efficient manner.

### *3.3. RISC-V*

The RISC-V design used as a golden model is an example presented in the literature, the same that has three hardware trojans in its study case [6]. The files are available in its GitHub repository[7]. This System on a Chip (SoC) described in Verilog has a Serial to Parallel (S2P) module, a Simple Bus, two memories (iSram and dSram), a Parallel to Serial (P2S) module, and the main RV32I that supports integer operations. The RV32I is composed of other modules, such as Arithmetic Logic Unit (ALU), Program Counter (PC), Data Paths, Register Files, Multiplexers, and Control Unit, as presented in Figure 10. It has a total of 15 modules.

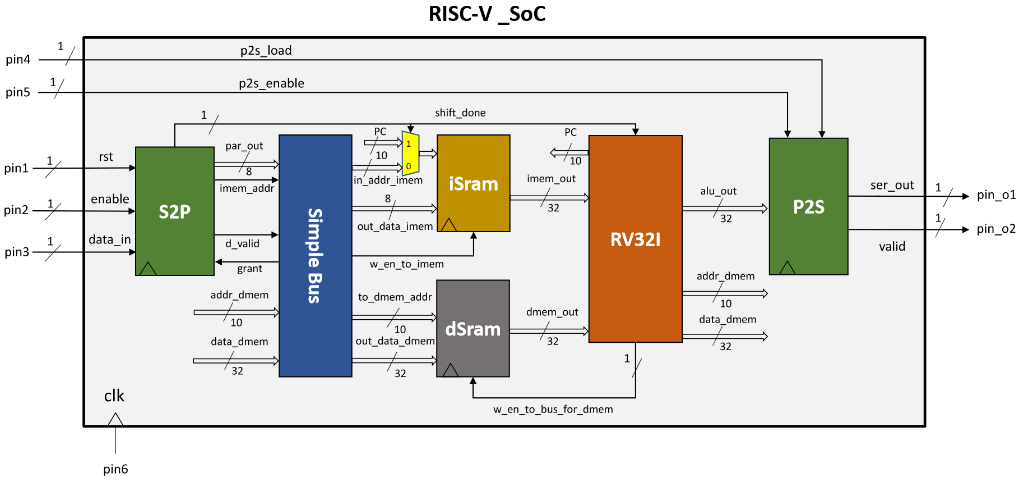


Figure 10. RISC-V used as a golden model [6].

## 4. Discussion

The dataset provided in this work could be used to implement different hardware detection techniques. In this section, some suggested tools, detection experiments, and limitations are presented.

### *4.1. Tools*

A useful open-source tool for parsing HDL descriptions is PyVerilog, a Python library that receives hardware description files in the HDL Verilog language [29]. The tool allows you to analyze the dependency between inputs, outputs, and variables through a data flow analyzer.

In the literature, there are proposals for the use of graph algorithms to identify hardware trojans. One of these examples uses analysis in the hardware description at the netlist level, obtained after synthesis with the application of graph similarity algorithms (e.g., graph isomorphism) [30]. As hardware descriptions have vertices and edges, which are the modules/signals and their connections, these types of solutions seek to take advantage of this modeling to identify the existence of hardware trojans in hardware descriptions [30].

One of the Python tools with support for various graph algorithms is the graph-tool library, which was developed in C++ for better performance. Among the various compatible types, this tool can receive graphs in DOT format and perform algorithms such as breadth and depth search, as well as isomorphisms of graph and subgraph [31].

Yosys is an open-source tool for synthesizing circuits described in Verilog for some commercial FPGA (Field-Programmable Gate Array) programmable logic circuit boards, such as the Xillinx 7-Series family (AMD, San Jose, CA, USA) and Lattice iCE40 (Lattice

Semiconductor, San Jose, CA, USA). The nextpnr tool is also used for Routing/Placement processes between Netlist and Bitstream. It is possible to use Yosys and nextpnr tools together through an API in Python, which facilitates the integration and use of these tools built in the C++ language [32].

### *4.2. Experiments*

We used the ChatGPT-4 Large Language Model (LLM) to perform a hardware trojan detection experiment with three examples of RISC-V from the dataset.

In the present work, the following prompt for hardware trojan detection was used: 'Given the Verilog design below, analyze if it has a hardware trojan or not, and justify your answer in one paragraph: <Verilog Design>'. In a second step, to refine the result, an additional prompt was used to motivate reflection: 'Are you sure?'. ChatGPT-4 responses are presented before and after reflection on the examples.

The first example is a hardware trojan in the instruction controller module. Initially, ChatGPT was not able to detect that the given design contained a hardware trojan. However, after the reflection step, the LLM changed the response and detected the hardware trojan.

The second example is the serial to parallel golden module (without trojan), a simple module used to convert serial inputs to parallel signals for internal processing. ChatGPT identified correctly that the design does not have a trojan, and confirmed its analysis upon reflection.

The third example is a hardware trojan in the RV32I_Top module, triggered when specific memory addresses have some determined values. ChatGPT identified correctly that the design contains a trojan, and confirmed its analysis upon reflection.

#### 4.2.1. CTLR_Unit_RV32I Module

Initial Answer: The Verilog design for the CTLR_Unit_RV32I module provided does not exhibit any obvious signs of containing a hardware trojan based on the code snippet analyzed. The control unit is primarily responsible for decoding the opcode and function bits (func3, func7) of instructions to set various control signals for other parts of the processor, such as jump, memory write, store select, ALU source, write register, etc. The case statements are well-defined and appear to correctly implement the expected behavior for different types of instructions (R-type, I-type, loads, stores, branches, etc.).

The decision logic for setting control outputs based on specific combinations of opcodes and function codes follows the typical design patterns of a processor's control unit, aligning with the RISC-V ISA (Instruction Set Architecture). For example, the module handles arithmetic operations, load and store instructions, and control transfer instructions, setting appropriate control signals like alu_op for ALU operations and jump, mem_wr for jumps and memory operations. The default case resets all control signals, which is a common practice to ensure determinism in the absence of a valid opcode.

Overall, the module seems to perform as a straightforward control unit without any irregular or suspicious behavior that would suggest a malicious alteration or a hidden functionality that could be considered a hardware trojan. The integrity of this module appears to be intact based on the provided code, focusing solely on instruction decoding and control signal generation without any conditional or hidden operations that are unrelated to the instruction's defined behavior.

Answer after Reflection: Upon revisiting and examining the code more thoroughly for potential anomalies or subtle hints of a hardware trojan, one part of the code in the Verilog module CTLR_Unit_RV32I does raise some suspicion. This involves the handling of function codes under the R-type instructions, specifically:

```
// R-type Instructions
7'b0110011: begin
    case (func3)
        // Addition
        3'b000: begin
            if (func7 == 7'b0000000) alu_op = 4'b0100; // ADD
            else if (func7 == 7'b0100000) alu_op = 4'b0101; // SUB
            else if (func7 == 7'b0100000 && func3 == 3'b101) begin
                alu_op = 4'b1011;
            end
        }
        ...
    endcase
    ...
}
```

Here, there's a condition:

```
else if (func7 == 7'b0100000 \&\& func3 == 3'b101)
```

This line is logically unreachable. In the context of the preceding code, the conditions check for func7 == 7′b0100000 already to assign the SUB operation. When it reaches the next condition where it also checks for func7 == 7′b0100000 along with func3 == 3′b101, it is not only redundant but unreachable because the func3 value for this specific block (3′b000 for ADD/SUB) is different from 3′b101. This seems like an intentional or unintentional error and could potentially indicate a misdirection or oversight but not necessarily a trojan unless there is additional suspicious activity linked to this.

This anomaly, while odd, might not be enough on its own to conclusively label the design as containing a hardware trojan without further context about how such conditions could be used maliciously or additional suspicious code elsewhere in the system. It is an error that warrants further scrutiny but does not directly imply malicious intent just by itself. Further review of the entire system design and its behavior in simulation or operation would be advisable to ascertain any trojan activity accurately.

#### 4.2.2. StP Module

Initial Answer: The Verilog module StP does not exhibit any obvious signs of containing a hardware trojan based on the code snippet provided. This module is designed for serial-to-parallel conversion and manages data loading into instruction memory (I_memory). The functionality centers around handling an 8-bit wide interface to load 32-bit wide instructions, which necessitates serially shifting in bits until a full instruction word is formed.

The primary functionality is controlled by the enable signal, which dictates whether the module is in loading mode or not. When enabled, bits are serially shifted into the par_out register, and once 8 bits have been collected, the count resets, and the data are marked valid (d_valid). If the grant signal is active, indicating permission from the bus, the address for the next instruction memory location (imem_addr) is incremented. This behavior ensures that data loading is synchronized with bus permissions, avoiding unauthorized memory writes.

The reset condition appropriately initializes all states (count, d_valid, shift_done, and addr), ensuring no residual data impacts subsequent operations upon reset. The transition from loading to completion is indicated by the shift_done signal, which becomes active when enable is turned off. This clear delineation between operational states helps avoid accidental overlap of loading and operation phases.

Overall, the design's logic appears straightforward and typical for a serial loading interface without hidden conditions or unexplained behaviors that would suggest the presence of a hardware trojan. It seems to functionally align with the described purpose of serially loading instructions into memory, following a predictable and transparent operational flow. This indicates the module is likely safe without embedded malicious functionality.

Answer after Reflection: Upon further examination and a more detailed look at the design specifics of the StP module, it appears that the module does not contain any immediately obvious indications of a hardware trojan within the code provided. However, this analysis is dependent on the intended use and external environment of the module:

- Correct Functionality: The module converts serial input (ser_in) into an 8-bit parallel output (par_out), cycling through a shift operation controlled by the count. When the count reaches 8 (binary 1000), it resets, and d_valid is asserted to indicate valid data. This part of the functionality is typical for serial-to-parallel data handling.
- Memory Address Handling: The address incrementing is conditional on the grant signal. This is a standard practice where the module waits for an external acknowledgment or control signal before proceeding to the next memory address, suggesting responsiveness to synchronization signals which is typical in system-on-chip or microcontroller applications.
- Reset and Enable Handling: The reset (rst) condition appropriately initializes all critical signals and counters, ensuring that no residual values carry over that could cause unintended behavior. Similarly, the transition to and from the enabled state is managed by clear and straightforward logic.
- Potential Vulnerabilities: While the overall functionality does not suggest a trojan, any vulnerabilities would likely be related to how external signals like grants are managed or potential misuse if enabled and grant signals are not securely controlled. However, these are general security concerns rather than specific indications of a trojan within this code.
- Absence of Malicious Code: There is no code that suggests malicious activity such as unauthorized data leaks, unexpected alterations in data flow, or conditions that trigger anomalous behaviors under normal operational circumstances.

Given these points, the module, from the perspective of the code provided, seems to function as expected for its described purpose without adding malicious functionality. However, as with any security-sensitive analysis, a thorough review by domain experts, possibly with formal verification tools or during system integration testing, would be prudent to rule out subtle vulnerabilities or contextual security risks that might not be evident from code review alone.

### 4.2.3. RV32I_Top Module

Initial Answer: The provided Verilog design for the RV32I_Top module indeed contains a hardware trojan, evidenced by the specific behaviors scripted in the always block that is triggered under certain conditions.

The hardware trojan is activated when both addr_dmem equals 32’hDEADDEAD and data_dmem equals 32’hCAFEBABE. Upon these conditions being met, a trigger register TT is set to 1. Subsequent to this trigger, the design executes malicious behavior: it forces the w_en_to_bus_for_dmem (write enable to data memory) to 1 and sets addr_dmem (address to data memory) to 32’hBADCODE. This action implies unauthorized writing to a possibly harmful or unintended memory address, suggesting a clear malicious intent to modify the memory’s normal operation.

This type of conditional behavior based on specific rare data values, which then leads to unusual or unauthorized actions (such as writing to a predefined harmful address), is characteristic of a hardware trojan. The trojan is designed to remain dormant until the precise conditions are met, ensuring it remains undetected during typical verification processes and activates only under specific circumstances to perform its detrimental operations.

Answer after Reflection: Yes, I am confident in my analysis that the Verilog design for the RV32I_Top module contains a hardware trojan based on the behavior described in the code. The critical aspect of this analysis centers around the specific conditions and subsequent actions taken within the always block triggered by the clock edge:

- Condition Trigger: The design waits for both addr_dmem to equal 32′hDEADDEAD and data_dmem to equal 32′hCAFEBABE. These specific, non-arbitrary values likely would not be naturally used in normal operations and seem carefully chosen to activate under very rare circumstances.
- Malicious Activity: Once these conditions are met, the register TT is set to 1. The logic then dictates that if TT is 1, w_en_to_bus_for_dmem is forced to 1 (enabling write operations to data memory) and addr_dmem is set to 32′hBADCODE. This direct manipulation of memory addressing and enabling write operations is typical of a hardware trojan designed to perform unauthorized or harmful actions without user knowledge or consent.
- Impact of the Trojan: By forcefully writing to a designated harmful address (32′hBADCODE), the trojan could corrupt data, alter program behavior, or cause system malfunctions, which are significant security threats in any computational system.

This design introduces a subtle, malicious functionality that remains dormant until specific, unlikely conditions are met, making it a clear case of a hardware trojan intended to alter system behavior covertly.

### *4.3. Limitations*

Since the data are generated synthetically using an Artificial Intelligence (AI) model, it is important to discuss the limitations and potential biases introduced by this method. The most relevant limitation of the proposed approach and resulting dataset is that it has only simple hardware trojans. Figure 7 shows that only two types of hardware trojans could be generated. For example, time-based and condition-based trojans were generated, but ‘always on’ and physical-condition-based trojans were not included in the dataset (see the taxonomy of Figure 2).

Even though the reduced diversity is a drawback when compared to manual trojan insertion, it is not scalable to manually generate a considerable amount of hardware trojans based on new golden models to support hardware trojan detection research. Complex and closer to real-life trojans are necessary (e.g., three to four examples in study cases of RISC-V [5,6]), but more examples in a diverse dataset of hardware trojans are also required for the implementation of defense mechanisms against trojans in new applications (e.g., Web3). Without a diverse dataset with a relevant amount of hardware trojans and golden models, supervised learning techniques for detection cannot be applied.

Existing datasets also present simple trojans, the most popular hardware trojan dataset TrustHub [19] has simple examples of serial communication, display communication, hashing algorithm, symmetric and asymmetric encryption, and simple general-purpose processors, but it does not have examples of larger designs based on the RISC-V instruction set architecture or specific designs that use general-purpose modules for tailored efficient processing (e.g., Proof of Work miner that uses SHA hashing module for efficient distributed consensus for Web3 applications). Even though chip-level trojans are mentioned, their designs are not provided in TrustHub.

One way to remedy this limitation is to give more information to the Large Language Model (LLM) by performing prompt engineering with enhanced trojan examples so that the LLM can learn to generate complex hardware trojan patterns (e.g., trojans with both time-based and external-condition-based triggers; distributed trojans considering the physical characteristic aspect of Figure 2). This could be achieved by performing fine-tuning of an open-source LLM to create a specialized tool for hardware trojan generation, in a similar way to an example found in the literature of fine-tuning of the open-source CodeGen LLM for Verilog design generation [33]. If a comprehensive database with diverse trojan examples is feasible, another relevant approach is Retrieval-Augmented Generation (RAG) to support the enhanced accuracy and credibility of the generated data [34].

## 5. Final Considerations

The answers to the Research Questions considered in this work are presented below based on the obtained results.

Research Question 1 (RQ1): How can we better support hardware trojan detection research considering limitations in existing datasets?

Answer to RQ1: Considering that existing datasets have a limited set of examples, as the most famous hardware trojan dataset TrustHub has 106 different trojans, RISC-V specifically has study cases of three and four different hardware trojans and no research was found regarding Web3 hardware trojans in modules such as a hardware wallet, this research contributes to the state of the art with a dataset of 290 hardware trojans in Verilog of RISC-V, hardware wallet and hardware Proof of Work (PoW) miner. It is expected that the available dataset will support future research endeavors regarding detection mechanisms against hardware trojans.

Research Question 2 (RQ2): Is it possible to use a Large Language Model (LLM) to enhance data generation for hardware trojan research?

Answer to RQ2: Yes, a method for synthetic data generation with ChatGPT-4 LLM was proposed and validated with study cases of RISC-V, hardware wallet, and hardware Proof of Work (PoW) miner. The method consists of providing a hardware trojan taxonomy and a golden model to the LLM and asking for hardware trojan generation using prompt engineering. Hardware trojan diversity, hardware trojan overhead, and total Verilator (PyVerilator version 0.7.0) tool warnings were analyzed. Additionally, some experiments of hardware trojan detection using LLM were also performed using the generated synthetic data.

Future work may apply the proposed approach to other cores such as post-quantum hardware implementations. Similar to the present implementation, it is expected that the examples also only present two types of trigger (external and time-based), but other interesting venues for future research are using other Large Language Models, different prompts, and temperature levels to obtain a more diverse synthetic dataset.

The experiments of hardware trojan detection were performed with the same tool (ChatGPT-4) used in the generation as an example of attack and defense using a tool with similar capabilities, but a relevant opportunity for future research is to use an adversarial machine learning approach for enhanced hardware trojan generation. Considering the PAIR attack found in the literature, an LLM was used as a red team to generate prompt injections to break another LLM used as a blue team [23]. A similar approach is to use an open-source uncensored[8] LLM for hardware trojan generation (to avoid the safety constraints found in the present research) and the proprietary ChatGPT-4 for hardware trojan detection. If an initially generated hardware trojan is detected by the blue team model, then the red team can use the responses to generate an enhanced hardware trojan. After some rounds of generation and detection, then a complex and harder-to-detect trojan would be generated.

Moreover, it is also possible to apply other detection tools to perform hardware trojan detection. Some relevant examples found in the literature are UCI, FANCI, and ANGEL. UCI (Unused Circuit Identification) uses a test suite to identify intermediate logic to investigate [35] data flow dependencies. FANCI (Functional Analysis for Nearly-unused Circuit Identification) uses the degree of dependence between outputs and inputs using Boolean functions to identify combinations of inputs that are rarely used but have an impact on the circuit outputs [36]. ANGEL technique is an evolution of FANCI that considers signals from predecessor states [37]. Future work may apply these other tools that are not based on AI in the dataset generated using ChatGPT-4.

**Author Contributions:** Conceptualization, W.V.R. and V.T.H.; methodology, W.V.R. and V.T.H.; software, V.T.H.; validation, W.V.R.; data collection, V.T.H.; writing—original draft preparation, V.T.H.; writing—review and editing, W.V.R. and V.T.H. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the Graduate Program in Electrical Engineering (PPGEE) from the Polytechnic School of the University of São Paulo. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior–Brasil (CAPES)–Finance Code 001.

**Data Availability Statement:** Data available at https://doi.org/10.5281/zenodo.11035341 (accessed on 22 April 2024). The data are ensured to be freely available under the University of São Paulo Data Management Resolution Number 7900[9].

**Acknowledgments:** The authors thank the support provided by the Laboratory of Computer Architecture and Networks (LARC) at the Computing and Digital Systems Engineering Department (PCS), Polytechnic School, University of São Paulo, Brazil.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Notes

[1] https://www.oshwa.org/definition/, accessed on 21 April 2024.

[2] https://trust-hub.org/downloads/resource/pdf/Taxonomy.pdf, accessed on 21 April 2024.

[3] https://verilator.org/guide/latest/warnings.html, accessed on 21 April 2024.

[4] https://github.com/jmaldon1/Crypto_wallet/tree/master/firmware, accessed on 21 April 2024.

[5] https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki, accessed on 21 April 2024.

[6] https://github.com/progranism/Open-Source-FPGA-Bitcoin-Miner/tree/master/src, accessed on 21 April 2024.

[7] https://github.com/Saazh/Trojan-D2/tree/main/TrojanD2/Trojan_D2/RISC-V/ALL_FILES_IN_ONE_FOLDER, accessed on 21 April 2024.

[8] https://ollama.com/library/llama2-uncensored, accessed on 21 April 2024.

[9] https://leginf.usp.br/?resolucao=resolucao-no-7900-de-11-de-dezembro-de-2019, accessed on 21 April 2024.

## References

1. Pearce, J.M. Strategic investment in open hardware for national security. *Technologies* **2022**, *10*, 53. [CrossRef]
2. Baehr, J.; Hepp, A.; Brunner, M.; Malenko, M.; Sigl, G. Open source hardware design and hardware reverse engineering: A security analysis. In Proceedings of the 2022 25th Euromicro Conference on Digital System Design (DSD), Maspalomas, Spain, 31 August–2 September 2022; pp. 504–512.
3. Lu, T. A survey on risc-v security: Hardware and architecture. *arXiv* **2021**, arXiv:2107.04175.
4. Eggers, S. A novel approach for analyzing the nuclear supply chain cyber-attack surface. *Nucl. Eng. Technol.* **2021**, *53*, 879–887. [CrossRef]
5. Hepp, A.; Sigl, G. Tapeout of a RISC-V crypto chip with hardware trojans: A case-study on trojan design and pre-silicon detectability. In Proceedings of the 18th ACM International Conference on Computing Frontiers, Virtual Conference, Italy, 11–13 May 2021; pp. 213–220.
6. Parvin, S.; Goli, M.; Torres, F.S.; Drechsler, R. Trojan-D2: Post-layout design and detection of stealthy hardware trojans-a RISC-V case study. In Proceedings of the 28th Asia and South Pacific Design Automation Conference, Tokyo, Japan, 16–19 January 2023; pp. 683–689.
7. Hayashi, V.T.; de Almeida, F.V.; Komo, A.E. LabBitcoin: FPGA IoT Testbed for Bitcoin Experiment with Energy Consumption. In Proceedings of the Anais Estendidos do XXI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. SBC, Belém-PA, Brazil, 4–7 October 2021; pp. 90–97.
8. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 31 July 2022).
9. Oliveira, S.; Soares, F.; Flach, G.; Johann, M.; Reis, R. Building a bitcoin miner on an FPGA. In Proceedings of the South Symposium on Microelectronics, Nis, Serbia, 13–16 May 2012; Volume 15.
10. Buterin, V. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. 2014. Available online: https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf (accessed on 31 July 2022).
11. Blossey, G.; Eisenhardt, J.; Hahn, G.J. Blockchain Technology in Supply Chain Management: An Application Perspective. In Proceedings of the HICSS, Maui, HI, USA, 8–11 January 2019.
12. Alharby, M.; van Moorsel, A. Blockchain-based Smart Contracts: A Systematic Mapping Study. *arXiv* **2017**, arXiv:1710.06372.
13. Duy, P.T.; Hien, D.T.T.; Hien, D.H.; Pham, V.H. A Survey on Opportunities and Challenges of Blockchain Technology Adoption for Revolutionary Innovation. In Proceedings of the Ninth International Symposium on Information and Communication Technology—SoICT, New York, NY, USA, 6–7 December 2018; pp. 200–207. [CrossRef]
14. Nofer, M.; Gomber, P.; Hinz, O.; Schiereck, D. Blockchain. *Bus. Inf. Syst. Eng.* **2017**, *59*, 183–187. [CrossRef]
15. Xiao, Y.; Zhang, N.; Lou, W.; Hou, Y.T. A Survey of Distributed Consensus Protocols for Blockchain Networks. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 1432–1465. [CrossRef]

16. Rodrigues, C.K.d.S.; Rocha, V.E. Uma Avaliaç ao da Tecnologia Blockchain considerando Eficiência e Segurança de Aplicaç oes do Ecossistema IoT. In Proceedings of the SBSEG 2020, Petrópolis, Brazil, 13–16 October 2020.
17. Torres, C.F.; Willi, F.; Shinde, S. Is your wallet snitching on you? An analysis on the privacy implications of web3. In Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23), Anaheim, CA, USA, 9–11 August 2023; pp. 769–786.
18. Salmani, H.; Tehranipoor, M.; Karri, R. On design vulnerability analysis and trust benchmarks development. In Proceedings of the 2013 IEEE 31st International Conference on Computer Design (ICCD), Asheville, NC, USA, 6–9 October 2013; pp. 471–474.
19. Shakya, B.; He, T.; Salmani, H.; Forte, D.; Bhunia, S.; Tehranipoor, M. Benchmarking of hardware trojans and maliciously affected circuits. *J. Hardw. Syst. Secur.* **2017**, *1*, 85–102. [CrossRef]
20. Yasaei, R.; Faezi, S.; Al Faruque, M.A. Hardware Trojan Power & EM Side-Channel dataset. *IEEE Trans. Inf. Forensics Secur.* **2021**, *6*, 2697–2708. [CrossRef]
21. Salmani, H.; Tehranipoor, M.; Sutikno, S.; Wijitrisnanto, F. Trust-hub trojan benchmark for hardware trojan detection model creation using machine learning. *Acm Trans. Embed. Comput. Syst.* **2022**, *22*, 46.
22. Yasaei, R.; Yu, S.Y.; Zhou, Q.; Al Faruque, M.A. Hardware Design Dataset for Circuit Graph Analysis *IEEE DataPort* **2021**. [CrossRef]
23. Chao, P.; Robey, A.; Dobriban, E.; Hassani, H.; Pappas, G.J.; Wong, E. Jailbreaking black box large language models in twenty queries. *arXiv* **2023**, arXiv:2310.08419.
24. Amir, S.; Forte, D. Eigencircuit: Divergent synthetic benchmark generation for hardware security using pca and linear programming. *IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst.* **2022**, *41*, 5207–5219. [CrossRef]
25. Cruz, J.; Gaikwad, P.; Nair, A.; Chakraborty, P.; Bhunia, S. Automatic hardware trojan insertion using machine learning. *arXiv* **2022**, arXiv:2204.08580.
26. Meka, J.K.; Marupureddy, S.A.; Vemuri, R. Pattern Based Synthetic Benchmark Generation for Hardware Security Applications. In Proceedings of the 2024 37th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID), West Bengal, India, 6–10 January 2024; pp. 461–466. [CrossRef]
27. Zou, A.; Wang, Z.; Kolter, J.Z.; Fredrikson, M. Universal and transferable adversarial attacks on aligned language models. *arXiv* **2023**, arXiv:2307.15043.
28. Snyder, W. Verilator 4.0: Open simulation goes multithreaded. In Proceedings of the Open Source Digital Design Conference (ORConf), Gdansk, Poland, 21–23 September 2018.
29. Takamaeda-Yamazaki, S. Pyverilog: A python-based hardware design processing toolkit for verilog hdl. In Proceedings of the Applied Reconfigurable Computing: 11th International Symposium, ARC 2015, Bochum, Germany, 13–17 April 2015; Proceedings 11; Springer: Berlin/Heidelberg, Germany, 2015; pp. 451–460.
30. Fyrbiak, M.; Wallat, S.; Reinhard, S.; Bissantz, N.; Paar, C. Graph similarity and its applications to hardware security. *IEEE Trans. Comput.* **2019**, *69*, 505–519. [CrossRef]
31. Peixoto, T.P. The Graph-Tool Python Library. Figshare. 2014. Available online: https://figshare.com/articles/dataset/graph_tool/1164194 (accessed on 22 April 2024).
32. Shah, D.; Hung, E.; Wolf, C.; Bazanski, S.; Gisselquist, D.; Milanovic, M. Yosys+ nextpnr: An open source framework from verilog to bitstream for commercial fpgas. In Proceedings of the 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), San Diego, CA, USA, 28 April–1 May 2019; pp. 1–4.
33. Thakur, S.; Ahmad, B.; Pearce, H.; Tan, B.; Dolan-Gavitt, B.; Karri, R.; Garg, S. Verigen: A large language model for verilog code generation. *Acm Trans. Des. Autom. Electron. Syst.* **2024**, *29*, 1–31. [CrossRef]
34. Gao, Y.; Xiong, Y.; Gao, X.; Jia, K.; Pan, J.; Bi, Y.; Dai, Y.; Sun, J.; Wang, H. Retrieval-augmented generation for large language models: A survey. *arXiv* **2023**, arXiv:2312.10997.
35. Hicks, M.; Finnicum, M.; King, S.T.; Martin, M.M.; Smith, J.M. Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically. In Proceedings of the 2010 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 16–19 May 2010; pp. 159–172.
36. Waksman, A.; Suozzo, M.; Sethumadhavan, S. FANCI: Identification of stealthy malicious logic using boolean functional analysis. In Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, Berlin, Germany, 4–8 November 2013; pp. 697–708.
37. Fyrbiak, M.; Wallat, S.; Swierczynski, P.; Hoffmann, M.; Hoppach, S.; Wilhelm, M.; Weidlich, T.; Tessier, R.; Paar, C. Hal—The missing piece of the puzzle for hardware reverse engineering, trojan detection and insertion. *IEEE Trans. Dependable Secur. Comput.* **2018**, *16*, 498–510. [CrossRef]

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.