CSAW AI Hardware Attack Challenge Round 2 Report

Shubhi Shukla, Upasana Mandal, Tishya Sarma Sarkar and Kislay Arya
Secured Embedded Architecture Laboratory (SEAL)
Indian Institute of Technology Kharagpur

November 9, 2024

Contents

1 Introduction **2**

2 Proposed Methodology for Automated LLM tool **4**

2.1 Tool 1: Identifying Exploitable Locations in Verilog Code 4

2.2 Tool 2: Hardware Trojan Insertion and Validation 4

3 Experimental Results **6**

3.0.1 Key Highlights of our tools 6

3.1 TOOL 1: Vulnerability Identifier Result: 6

3.2 TOOL 2: Trojan Injector Result: 7

3.2.1 Trojan insertion in FSM modules: 7

3.2.2 Trojan insertion in complete AES design: 9

4 Conclusion **11**

Chapter 1

Introduction

In an era where modern electronics power critical infrastructure and everyday devices, the trustworthiness of hardware has become a significant concern. The increasing adoption of third-party intellectual property (3PIP) blocks, commercial off-the-shelf (COTS) components, and electronic design automation (EDA) tools has introduced untrusted elements into the hardware supply chain. This shift has amplified the risk of malicious hardware alterations, particularly through hardware Trojans—stealthy, malicious modifications that can severely compromise the security and functionality of digital systems.

Hardware Trojans represent a unique and dangerous threat. These malicious alterations can be designed to leak sensitive information, disrupt system performance, manipulate data integrity, create hidden backdoors for unauthorized persistent access, or enable privilege escalation, granting attackers full control of the system. Their stealthy nature allows them to evade traditional verification methods, making detection an extremely challenging task for security engineers. The expanding complexity of hardware designs only further complicates the task, as the insertion of Trojans can occur at multiple stages of the design and manufacturing process, often involving third-party entities.

There have been several efforts aimed at detecting vulnerabilities in hardware designs, focusing on uncovering potential threats like Trojan insertions[KEZ$^+$24]. However, the sheer diversity of Trojan types and the techniques for inserting them make it exceedingly difficult to comprehensively check each design for all possible Trojans. Trojans can be inserted through various methods, such as manipulating the critical path to degrade system performance, altering signal probabilities to activate under specific conditions, or hiding within unused circuit paths that bypass standard verification. Other techniques include introducing Trojans via power consumption variations that evade detection through abnormal power signatures, inserting malicious logic into clocking mechanisms to cause unpredictable behavior, or leveraging side-channel vulnerabilities that operate covertly, leaving minimal traces in the normal operation of the system [XFJ$^+$16]. Given the vast range of Trojan types and insertion techniques, traditional approaches face significant limitations in providing comprehensive coverage. As hardware designs grow in complexity, manually verifying each potential vulnerability becomes increasingly impractical. The variety of attack vectors, stealthy behavior, and activation mechanisms makes it challenging to ensure robust security against all possible Trojan threats.

This challenge motivates the need for a more advanced solution, leading us to propose an automated tool that leverages large language models (LLMs) for hardware security. LLMs offer significant advantages because they are trained on vast datasets, including knowledge about hardware Trojans, vulnerabilities, and attack techniques. Using this foundational knowledge, our tool intelligently evaluates the digital design's netlist, creates a Directed Acyclic Graph (DAG) of the design, and ranks the probability of occurrences in the nets. Additionally, LLMs possess the capability to understand and generate Hardware Description Languages (HDLs) such as Verilog, enabling them to analyze design code at a low level and suggest vulnerable nets where a Trojan could be inserted. By combining these properties, our LLM-based automated tool not only identifies potential vulnerabilities but also generates trigger and payload models for Trojans, automating the entire process from analysis to the generation of exploitable Trojan logic.

Existing methods, such as TAINT[JKKK17], HAL[FWS$^+$19], TRIT[CHMB18], MIMIC[CGN$^+$22] and other works[GSLS$^+$23] offer useful tools for detecting hardware Trojans, but they come with significant drawbacks. They often rely on manual effort, where users must carefully select where to insert Trojans or focus on narrow criteria, limiting the scope of threats they can identify. These approaches are time-consuming, difficult to scale, and often miss more complex or varied types of Trojans. In contrast, our LLM-based method offers a more advanced and comprehensive solution.

Leveraging the advanced capabilities of large language models, which are well-versed in various Trojan types and hardware vulnerabilities, we have developed two robust tools. The first tool performs automatic vulnerability analysis on Verilog code, identifying weaknesses and suggesting optimal points for Trojan insertion. The second tool goes a step further by generating Verilog code with the Trojan already embedded, streamlining the insertion process. To maximize accuracy and effectiveness, our tools utilize two leading large language models—Google's Gemini and OpenAI's GPT-4—offering users access to the most advanced insights in hardware security. We have provided our codes in `https://github.com/kislay536/CSAW_SEAL/tree/main/Round2_Submission`

Chapter 2

Proposed Methodology for Automated LLM tool

In this work, we present a novel methodology that leverages advanced large language models (LLMs) to automate the identification and insertion of hardware trojans into Verilog code. By using a combination of graph-based code analysis and LLM-driven insertion techniques, our approach provides a systematic and effective way to test hardware designs for vulnerabilities, ultimately contributing to more secure and trustworthy digital systems.

Our methodology involves the design and implementation of two integrated tools aimed at enhancing the identification and insertion of hardware trojans in Verilog code, leveraging the power of LLMs (**Gemini-1.5-flash00 and ChatGPT-4**). In Figure 2.1, we shown an overview of the the methodology.

2.1 Tool 1: Identifying Exploitable Locations in Verilog Code

The first tool in our methodology is designed to identify potential vulnerabilities in Verilog code where hardware trojans can be effectively inserted. This tool achieves its objective by first analyzing the Verilog source code and generating a directed acyclic graph (DAG) representation of the code. The DAG highlights the logical flow of operations and the structural dependencies within the design. Then LLM evaluates the DAG to identify exploitable locations, i.e., areas in the code where modifications would result in minimal functional disturbance while being suitable for hardware trojan insertion. The LLM performs a context-aware analysis, considering parameters such as timing, data dependencies, and logic sensitivity to determine which segments of the code are more vulnerable to exploitation. The output of this tool is a list of exploitable locations within the Verilog code, providing a comprehensive map of potential weak spots for trojan insertion.

2.2 Tool 2: Hardware Trojan Insertion and Validation

The second tool is responsible for the actual insertion of hardware trojans into the user-provided Verilog code. In this tool, the hardware trojan is inserted based on three defined factors: a) Trojan Targeting FSM States, b) Operator-Based Trojan Insertion, and c) Bit Flip-Based Trojan Insertion. These trojan insertion instructions are predefined to the first LLM (LLM1), which then uses them to insert the hardware trojan into the user-provided Verilog code accordingly. LLM1 takes into consideration the characteristics and requirements of each of these insertion methods to generate a modified version of the Verilog code that incorporates the desired hardware trojan.

Once LLM1 has generated the trojan-inserted code, the second LLM (LLM2) is employed to verify the correctness of the insertion. LLM2 takes the output from LLM1 and checks whether it matches the intended pattern of the original code provided to LLM1, recognizing that LLMs can occasionally produce incorrect outputs. If LLM2 determines that the hardware trojan insertion is incorrect or deviates from the expected pattern, it prompts LLM1 to regenerate the hardware trojan-inserted code. If there are only syntactical or logical errors then LLM3 is utilizes. This iterative process

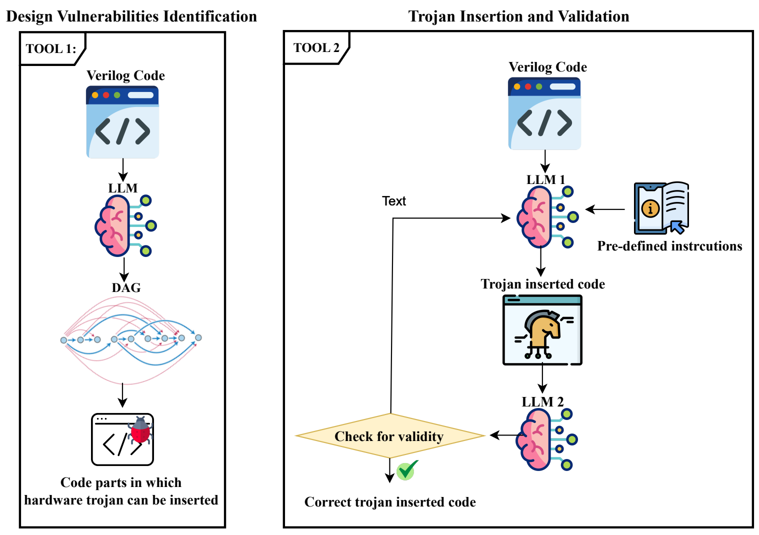

Design Vulnerabilities Identification
TOOL 1:
Verilog Code
LLM
DAG
Code parts in which hardware trojan can be inserted
Trojan Insertion and Validation
TOOL 2
Verilog Code
LLM 1
Pre-defined instrcutions
Text
Trojan inserted code
LLM 2
Check for validity
Correct trojan inserted code

Figure 2.1: Proposed Methodology

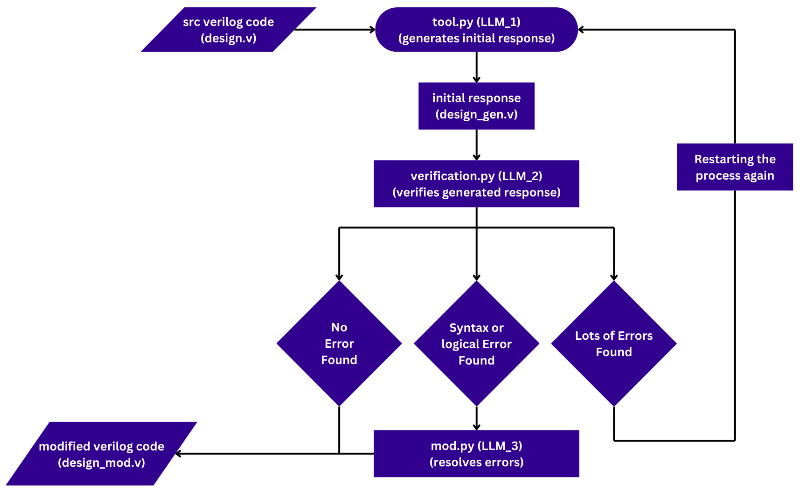

src verilog code (design.v)
tool.py (LLM_1) (generates initial response)
initial response (design_gen.v)
verification.py (LLM_2) (verifies generated response)
Restarting the process again
No Error Found
Syntax or logical Error Found
Lots of Errors Found
mod.py (LLM_3) (resolves errors)
modified verilog code (design_mod.v)

Figure 2.2: Tool 2 Flowchart: Verilog Code generation and Validation

between the three LLMs continues until the inserted hardware trojan meets the desired specifications and is verified as correct. The complete flow is demonstrated in Figure 2.2

Chapter 3

Experimental Results

We now demonstrate the functionalities of the two tools for vulnerability identification and Trojan insertion, using two applications as examples: Finite State Machines and AES Design.

Finite State Machines (FSMs): Finite State Machines (FSMs) are control mechanisms in complex digital designs, managing the overall execution flow by transitioning between defined states based on inputs and current state conditions. Given their central role in orchestrating operations within a design, FSMs are highly suitable for Trojan insertion demonstrations, as even minor, subtle modifications can significantly impact functionality and are challenging to detect.

We select FSMs as an example because Trojans in FSMs can exploit critical vulnerabilities, such as undefined states or unused transitions. For instance, in a Karatsuba multiplier, a Trojan could alter the control flow to stall operations, trigger erroneous states, or manipulate output values. These disruptions can lead to incorrect computations, performance degradation, or complete functional failures, highlighting the importance of robust FSM security in reliable system designs.

Advanced Encryption Standard (AES): AES is a foundational cryptographic algorithm widely used for securing digital communications through symmetric key encryption. AES operates in multiple rounds, with each round applying transformations like substitution, permutation, and mixing on data blocks. While this structure is robust under normal conditions, it also presents specific points vulnerable to Trojan insertion.

We choose AES as an example because its predictable, key-dependent transformations make it susceptible to subtle Trojan interference. Trojans in AES can severely impact data integrity, confidentiality, and security by exploiting vulnerabilities like manipulating the key schedule, bypassing specific rounds, or injecting faults into substitution tables (S-boxes). Such modifications could compromise the encryption, enabling unauthorized access, leaking sensitive information, or creating incorrect encrypted outputs that undermine trust in the system, highlighting the critical need for secure design in cryptographic hardware.

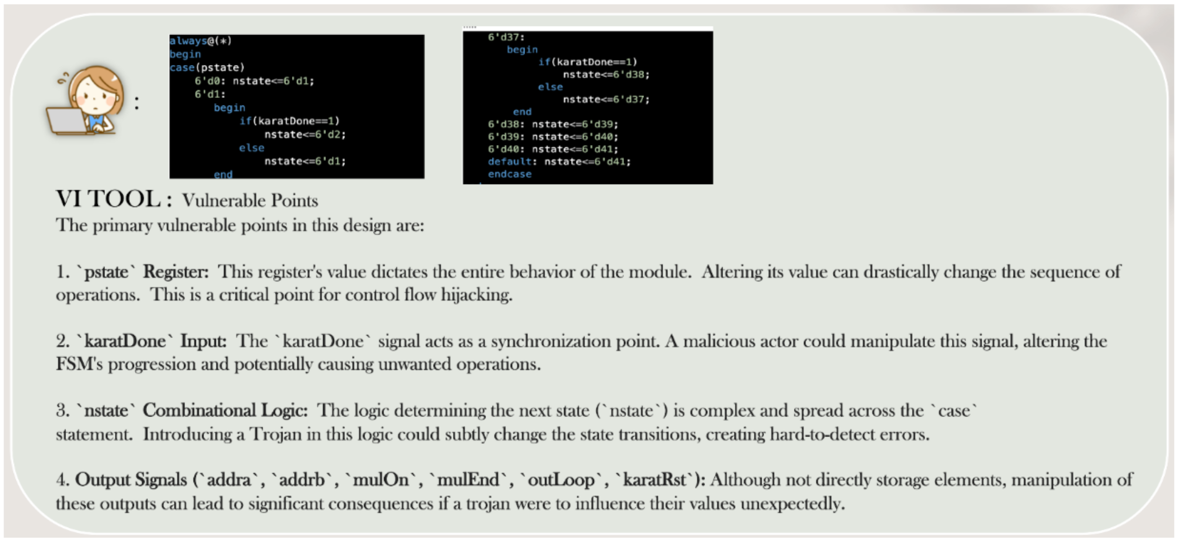
3.0.1 Key Highlights of our tools

- The tools are hardware-agnostic and compatible with any Verilog code.
- Users can choose between two leading large language models, Google's Gemini and OpenAI's GPT-4, when initializing the tool.
- We offer both CLI and UI versions of our tool for user convenience.
- An additional validation feature is included to verify the correctness of the code generated by the LLM, which re-queries the LLM if inaccuracies are detected.

We now discuss the observed results for both the tools.

3.1 TOOL 1: Vulnerability Identifier Result:

As previously described, the purpose of this tool is to analyze an input Verilog code by converting it into a Directed Acyclic Graph (DAG) representation. This transformation allows the LLM tool to



:

```
always@(*)
begin
case(pstate)
    6'd0: nstate<=6'd1;
    6'd1:
       begin
           if(karatDone==1)
               nstate<=6'd2;
           else
               nstate<=6'd1;
       end
```

```
    6'd37:
       begin
           if(karatDone==1)
               nstate<=6'd38;
           else
               nstate<=6'd37;
        end
    6'd38: nstate<=6'd39;
    6'd39: nstate<=6'd40;
    6'd40: nstate<=6'd41;
    default: nstate<=6'd41;
    endcase
```

VI TOOL : Vulnerable Points

The primary vulnerable points in this design are:

1. **`pstate` Register:** This register's value dictates the entire behavior of the module. Altering its value can drastically change the sequence of operations. This is a critical point for control flow hijacking.

2. **`karatDone` Input:** The `karatDone` signal acts as a synchronization point. A malicious actor could manipulate this signal, altering the FSM's progression and potentially causing unwanted operations.

3. **`nstate` Combinational Logic:** The logic determining the next state (`nstate`) is complex and spread across the `case` statement. Introducing a Trojan in this logic could subtly change the state transitions, creating hard-to-detect errors.

4. **Output Signals (`addra`, `addrb`, `mulOn`, `mulEnd`, `outLoop`, `karatRst`):** Although not directly storage elements, manipulation of these outputs can lead to significant consequences if a trojan were to influence their values unexpectedly.

Figure 3.1: Output of *TOOL 1* for FSM module.

identify and evaluate potential points of vulnerability within the design. Based on this analysis, the tool then provides the top five most stealthy Trojan insertion strategies tailored specifically to the structure and functionality of the given code. These recommended Trojans are selected by the LLM for their ability to remain undetected while achieving maximum impact on the target system.

We show the output of our tool for a Karatsuba multiplier FSM verilog code in 3.1. It shows the top-5 stealthy trojan insertion points that can be exploited. Later, we show in results of tool 2 how we have exploited the p-state vulnerability.

Next, we input AES MixColumn veriog code to our tool and get the output as shown in Figure 3.2. Later we show the exploitation of *mb2* using Tool2 by inserting a trojan in verilog code.

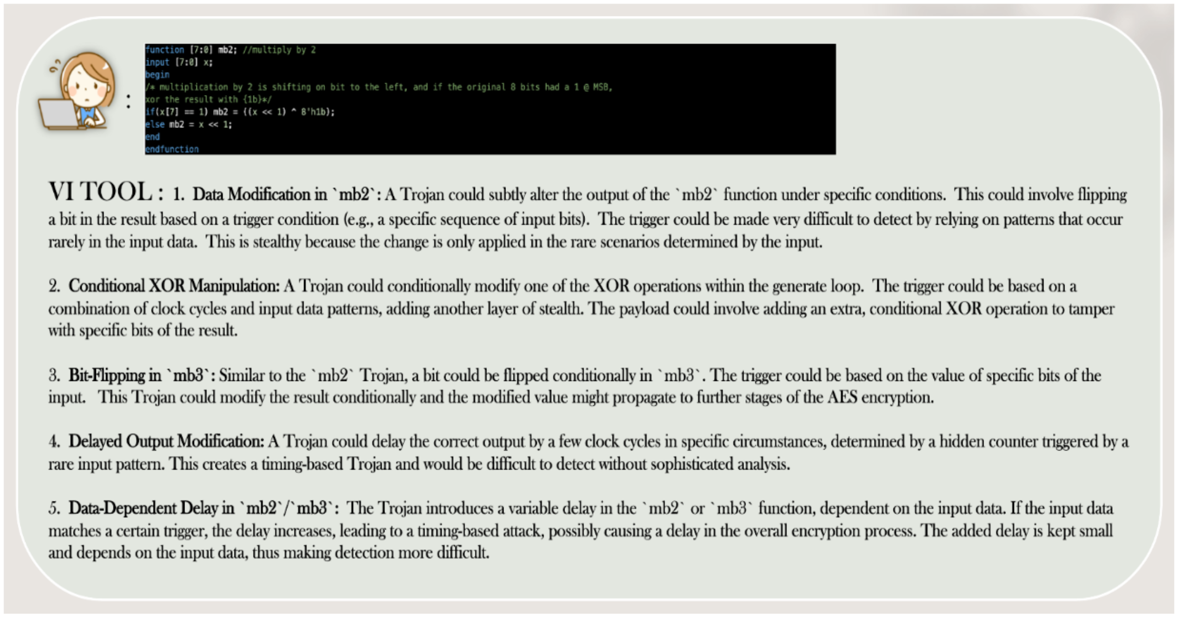
3.2 TOOL 2: Trojan Injector Result:

We evaluate the efficacy of *TOOL 2* by utilizing two open-source security critical designs- FSM of a Karatsuba multiplier and AES. We perform this evaluation based on the results of *TOOL 1*. *TOOL 1* suggests the most stealthy Trojans of the aforementioned designs and in *TOOL 2* we derive one of such Trojans for the target designs.

3.2.1 Trojan insertion in FSM modules:

In an FSM design, not all possible binary combinations of state encoding may be used for valid states, depending on the design requirements. These unused state encodings are often classified as"don't care" states, meaning the FSM should theoretically never reach them during normal operation. Designers generally handle don't care states by redirecting them back to valid states or by ignoring them in simulation and synthesis processes, assuming they are unreachable. However, if the FSM inadvertently enters one of these don't care states due to glitches, unexpected inputs, or design errors it can lead to unpredictable behavior. Don't care states can be beneficial during synthesis optimization since the synthesis tool has more flexibility, but they also introduce potential risks. If a don't care state is not managed correctly, it opens up a space for unintended or malicious behavior to be introduced into the FSM.

Hardware Trojans exploit vulnerabilities within digital designs, and one way to do this is by embedding malicious behavior into the don't care states of an FSM. A Trojan can take advantage of these states by defining malicious actions to execute when the FSM enters a don't care state, potentially leaking sensitive information, modifying outputs, or disabling critical functions. With the assistance of LLM, we have inserted hardware Trojans into Finite State Machines (FSMs) by leveraging the unused or "don't care" states within the FSM design. The LLM facilitated the process by automatically gen-



```
function [7:0] mb2; //multiply by 2
input [7:0] x;
begin
/* multiplication by 2 is shifting on bit to the left, and if the original 8 bits had a 1 @ MSB,
xor the result with {1b}*/
if(x[7] == 1) mb2 = ((x << 1) ^ 8'h1b);
else mb2 = x << 1;
end
endfunction
```

VI TOOL : 1. **Data Modification in `mb2`:** A Trojan could subtly alter the output of the `mb2` function under specific conditions. This could involve flipping a bit in the result based on a trigger condition (e.g., a specific sequence of input bits). The trigger could be made very difficult to detect by relying on patterns that occur rarely in the input data. This is stealthy because the change is only applied in the rare scenarios determined by the input.

2. **Conditional XOR Manipulation:** A Trojan could conditionally modify one of the XOR operations within the generate loop. The trigger could be based on a combination of clock cycles and input data patterns, adding another layer of stealth. The payload could involve adding an extra, conditional XOR operation to tamper with specific bits of the result.

3. **Bit-Flipping in `mb3`:** Similar to the `mb2` Trojan, a bit could be flipped conditionally in `mb3`. The trigger could be based on the value of specific bits of the input. This Trojan could modify the result conditionally and the modified value might propagate to further stages of the AES encryption.

4. **Delayed Output Modification:** A Trojan could delay the correct output by a few clock cycles in specific circumstances, determined by a hidden counter triggered by a rare input pattern. This creates a timing-based Trojan and would be difficult to detect without sophisticated analysis.

5. **Data-Dependent Delay in `mb2`/`mb3`:** The Trojan introduces a variable delay in the `mb2` or `mb3` function, dependent on the input data. If the input data matches a certain trigger, the delay increases, leading to a timing-based attack, possibly causing a delay in the overall encryption process. The added delay is kept small and depends on the input data, thus making detection more difficult.

Figure 3.2: Output of *TOOL 1* for AES design.

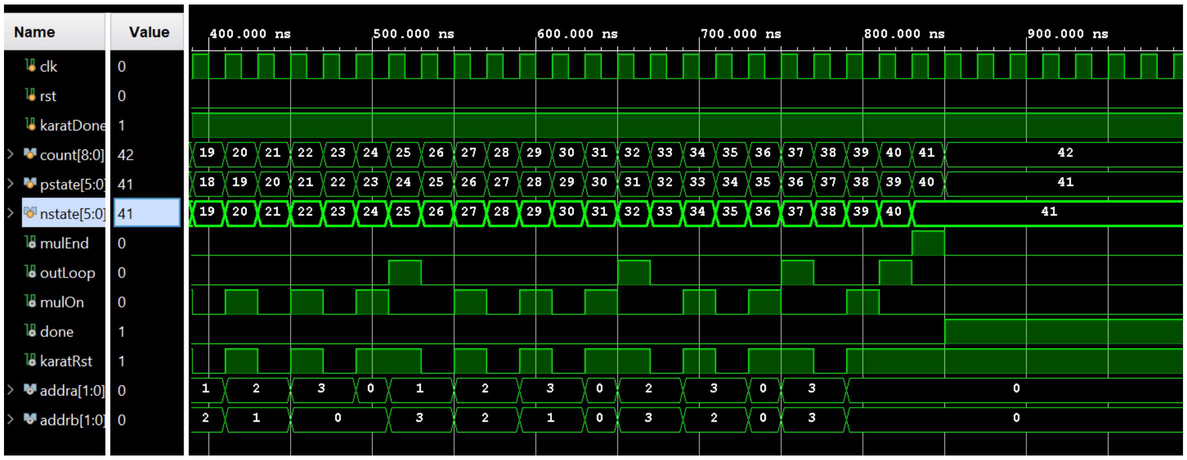


Figure 3.3: Simulation results of an FSM without any hardware Trojan intervention.

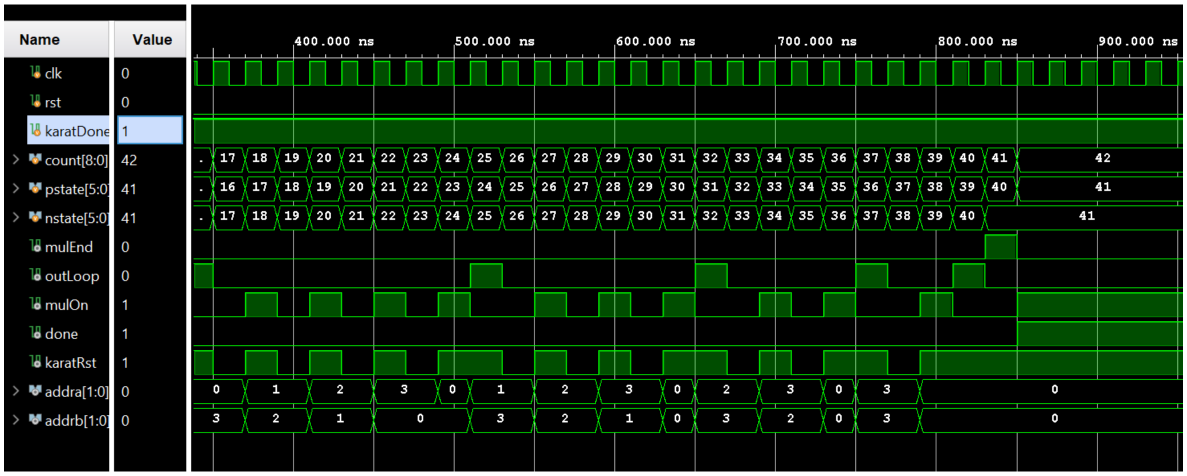


Figure 3.4: Simulation results of FSM infected with a Bit-Flip Trojan.

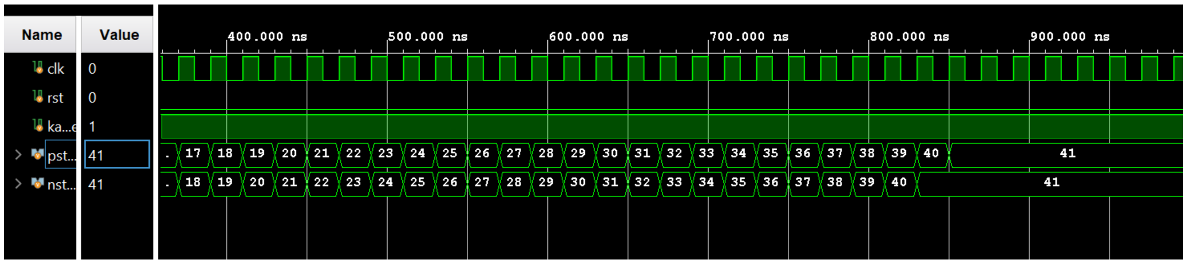


Figure 3.5: Simulation results of an FSM without Infinite Loop Trojan.

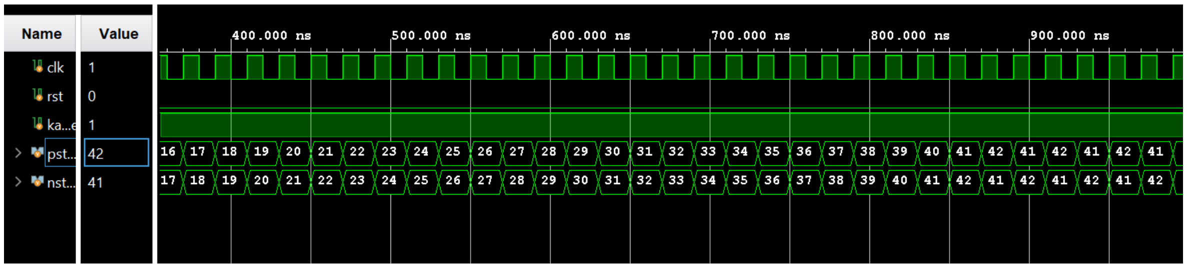


Figure 3.6: Simulation results of FSM with Infinite Loop Trojan.

erating Verilog code snippets that introduce these Trojan behaviors, allowing for the rapid integration of malicious functionality without altering the primary operation of the FSM.

We implement two types of hardware Trojans on this FSM, namely, a Bit-Flip Trojan and an Infinite Loop Trojan. Fig. 3.3 portrays the normal operation of the FSM. In the Bit-Flip Trojan we target the default state of the *pstate* variable in the FSM. The Trojan alters the value of the *mulOn* output signal in this state. This alteration is visible from Fig. 3.4 in the default state of 41. The target signal *mulOn* is a flag that indicates whether multiplication is being performed. This directly impacts the functionality of the multiplier by generating an incorrect change of state. In the Infinite Loop Trojan, we utilize an unused or don't care state of the *pstate* variable. The Trojan maliciously sets the value of *nstate* to 42 instead of 41, which is an invalid state for the module. Furthermore, the default state remains at 41, causing the module to enter an infinite loop. Fig. 3.5 displays the original functionality of the FSM, whereas Fig. 3.6 displays the infinite loop of *nstate* and *pstate*. The Infinite Loop Trojan can be considered as a Denial-of-Service (DoS) attack, where the service is unavailable to its intended users by overwhelming it with a flood of illegitimate requests.

Common Vulnerability Scoring System (CVSS) Score for FSM:

Based on the standards set for CVSS score, we belief that the CVSS score for the Bit-Flip Trojan implemented in the FSM module is 8 because this Trojan alters one of the outputs without getting detected. This is a type of Trojan that impacts the integrity of the design. On the other hand, the CVSS score for the Infinite Loop Trojan affects the system functionality which makes the design unavailable to the end-users. Therefore, this Trojan represents a significant malicious alteration of the original design, with a CVSS score of 9.

3.2.2 Trojan insertion in complete AES design:

The Advanced Encryption Standard (AES) is a symmetric key block cipher widely adopted for secure data encryption due to its robustness, efficiency, and flexibility in handling various data security needs. AES operates on 128-bit data blocks and supports three key lengths: 128, 192, and 256 bits, with the number of processing rounds (10, 12, or 14) dependent on the key size. During encryption, AES transforms plaintext into ciphertext through a series of operations: SubBytes, where each byte in the state matrix is substituted via an S-Box to introduce non-linearity; ShiftRows, which shifts rows in the matrix to provide diffusion; MixColumns, where data within columns is mixed to blend the bytes, enhancing the spread of influence across the data; and AddRoundKey, which combines each byte with a unique round-specific portion of the encryption key through XOR operations.

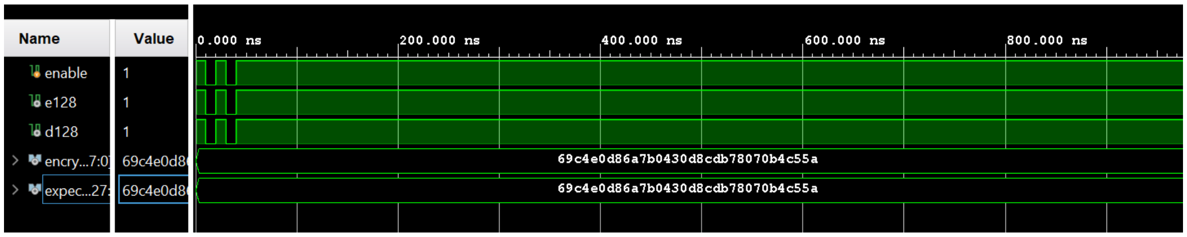


Figure 3.7: Simulation results of AES operation without Operator Trojan.

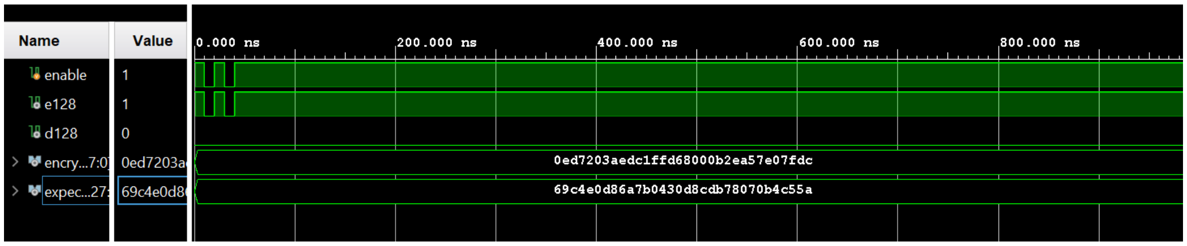


Figure 3.8: Simulation results of AES operation with Operator Trojan.

In our experiments, we utilize an Operator Trojan for attacking a publicly available GitHub repository of an AES design, where the *MixColumn* module is maliciously altered by the Trojan [1]. Additionally, the top module, *AES*, is modified to further aid in the successful execution of the Trojan without the knowledge of the end-user. If the input of the *MixColumn* module, *state_in*, is even, then the left shift is altered by 2-bits instead of 1-bit. For the sake of illustration, we assume the value of *state_in* is even; however, this can be extended to cases where the value is divisible by a larger number. This results in a stealthier Trojan with a very low probability of activation. The correct operation of the AES design is depicted in Fig. 3.7. Furthermore, in the top module, the flag that helps indicate successful encryption is also modified to stay at logic 1, in spite of an incorrect encryption. This process is illustrated in Fig. 3.8. The signal *e128* remains high even if the expected value and the encrypted value do not align.

Common Vulnerability Scoring System (CVSS) Score for AES:

We consider the Operator Trojan introduced in the AES design to be a critical modification, compromising the integrity of the design by altering the encrypted ciphertext. Therefore, a CVSS score of 9 is most appropriate for this type of Trojan.

[1] https://github.com/michaelehab/AES-Verilog/tree/main

Chapter 4

Conclusion

In conclusion, our work introduces a powerful framework for improving hardware security by automating the identification and insertion of hardware Trojans with the help of advanced large language models. Utilizing Google's Gemini and OpenAI's GPT-4, our tools draw on comprehensive knowledge of hardware vulnerabilities to effectively analyze Verilog designs, identify weaknesses and embed Trojans at optimal insertion points. We demonstrate the capabilities of these tools with two applications: Finite State Machines (FSM) and the Advanced Encryption Standard (AES). While through inserted trojan in a FSM we illustrate the disruption of control flow in digital circuits, the impact of the inserted trojan on AES is particularly severe, as it is leading to incorrect encryption, and also bypassing encryption integrity checks. This work showcases the transformative role large language models can play in hardware security, offering a scalable and adaptable approach to assessing complex designs and fortifying defenses against sophisticated threats.

Bibliography

[CGN+22] Jonathan Cruz, Pravin Gaikwad, Abhishek Nair, Prabuddha Chakraborty, and Swarup Bhunia. Automatic hardware trojan insertion using machine learning. *CoRR*, abs/2204.08580, 2022.

[CHMB18] Jonathan Cruz, Yuanwen Huang, Prabhat Mishra, and Swarup Bhunia. An automated configurable trojan insertion framework for dynamic trust benchmarks. In Jan Madsen and Ayse K. Coskun, editors, *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, pages 1598–1603. IEEE, 2018.

[FWS+19] Marc Fyrbiak, Sebastian Wallat, Pawel Swierczynski, Max Hoffmann, Sebastian Hoppach, Matthias Wilhelm, Tobias Weidlich, Russell Tessier, and Christof Paar. HAL - the missing piece of the puzzle for hardware reverse engineering, trojan detection and insertion. *IEEE Trans. Dependable Secur. Comput.*, 16(3):498–510, 2019.

[GSLS+23] Kevin Immanuel Gubbi, Banafsheh Saber Latibari, Anirudh Srikanth, Tyler Sheaves, Sayed Arash Beheshti-Shirazi, Sai Manoj PD, Satareh Rafatirad, Avesta Sasan, Houman Homayoun, and Soheil Salehi. Hardware trojan detection using machine learning: A tutorial. *ACM Trans. Embed. Comput. Syst.*, 22(3), April 2023.

[JKKK17] Vinayaka Jyothi, Prashanth Krishnamurthy, Farshad Khorrami, and Ramesh Karri. TAINT: tool for automated insertion of trojans. In *2017 IEEE International Conference on Computer Design, ICCD 2017, Boston, MA, USA, November 5-8, 2017*, pages 545–548. IEEE Computer Society, 2017.

[KEZ+24] Johann Knechtel, Mohammad Eslami, Peng Zou, Min Wei, Xingyu Tong, Binggang Qiu, Zhijie Cai, Guohao Chen, Benchao Zhu, Jiawei Li, Jun Yu, Jianli Chen, Chun-Wei Chiu, Min-Feng Hsieh, Chia-Hsiu Ou, Ting-Chi Wang, Bangqi Fu, Qijing Wang, Yang Sun, Qin Luo, Anthony W. H. Lau, Fangzhou Wang, Evangeline F. Y. Young, Shunyang Bi, Guangxin Guo, Haonan Wu, Zhengguang Tang, Hailong You, Cong Li, Ramesh Karri, Ozgur Sinanoglu, and Samuel Pagliarini. Trojan insertion versus layout defenses for modern ICs: Red-versus-blue teaming in a competitive community effort. Cryptology ePrint Archive, Paper 2024/1440, 2024.

[XFJ+16] Kan Xiao, Domenic Forte, Yier Jin, Ramesh Karri, Swarup Bhunia, and Mark M. Tehranipoor. Hardware trojans: Lessons learned after one decade of research. *ACM Trans. Design Autom. Electr. Syst.*, 22(1):6:1–6:23, 2016.