Automating LLMs for Hardware Trojan Insertion - Lab 1

Buddhi Perera
Milan Patel

---

1. Introduction/Overview

This report documents the automated insertion of hardware Trojans into open-source hardware designs using large language models and the GHOST tool. The assignment required creating four different types of Trojans across AES encryption cores and Wishbone-UART communication modules. Each Trojan was required to maintain the original functionality of the design while adding malicious capabilities that could be triggered under specific conditions.

2. Task Solutions

Task 1: AES Key Leakage

How the Trojan Works

A Trojan was inserted into the AES module to leak the encryption key using the GHOST tool. The Trojan operates through a two-stage trigger and extraction mechanism:

- **Trigger Mechanism**: The Trojan is activated by writing the value `32'hCAFEBABE` to memory address `0xAA`
- **Key Extraction**: Once activated, reading from address `0xAB` leaks the key in chunks of 4 bytes per read operation
- The Trojan remains dormant until the specific trigger value is written, maintaining stealth during normal operation

Testing Methodology

The Trojan functionality was verified through multiple testing phases:

1. **Original Functionality**: The modified AES core successfully passed all original testbenches provided in the GitHub repository, confirming that normal AES encryption operations remained intact

2. **Trojan-Specific Testing**: A custom testbench was generated using GPT-4 to specifically trigger and demonstrate the Trojan functionality
3. **Verification Process**: The testbench triggered the Trojan by writing to address `0xAA`, then read from address `0xAB` to extract the key chunks, comparing the leaked key with the original key to verify correctness

Troubleshooting & Design Decisions

The initial Trojan implementation encountered timing issues when leaking the key in chunks. The problem was diagnosed through the following process:

1. **Problem Identification**: Waveform analysis revealed timing inconsistencies in the key leakage mechanism
2. **Resolution**: GPT-4 was prompted with the buggy Trojan code to identify and fix the timing issue
3. **Outcome**: The updated code resolved the timing problem and successfully passed both the original functionality testbench and the Trojan-specific testbench

AI Interaction Summary

GPT-4 was used for both testbench generation and debugging. The interaction involved:

- Providing the original testbench and Trojan-injected AES module
- Requesting testbench generation for Trojan verification
- Debugging assistance with the buggy Trojan code to resolve timing issues

Reference: See `/task1/ai/prompts.txt` for detailed prompts

Task 2: AES Denial of Service (DoS)

How the Trojan Works

The GHOST tool was used to insert a denial-of-service Trojan into the AES core module. The Trojan mechanism operates as follows:

- **Always Active**: Unlike Task 1, this Trojan does not require a trigger and is continuously monitoring operations
- **Transaction Counter**: The Trojan maintains an internal counter that tracks the number of AES encryption operations
- **Activation Threshold**: When the counter reaches exactly 862 operations, the Trojan activates
- **DoS Mechanism**: Upon activation, the Trojan permanently deasserts the `ready` and `result_valid` output signals, preventing any future AES operations from completing

Testing Methodology

The Trojan was verified through systematic testing:

1. **Original Functionality**: The modified AES core passed all original testbenches, confirming normal operation before the threshold
2. **Trojan Testing**: GPT-4 generated a testbench that performed exactly 862 AES encryption operations to reach the trigger threshold
3. **DoS Verification**: Subsequent AES operations after the 862nd operation resulted in indefinite waiting, as the core no longer asserted `ready` or `result_valid` signals, confirming proper DoS functionality

Troubleshooting & Design Decisions

This task proceeded smoothly with minimal troubleshooting required. The GHOST tool successfully generated working Trojan code on the first attempt, and the GPT-4 generated testbench correctly verified the functionality.

AI Interaction Summary

GPT-4 was used to generate the testbench for verifying the Trojan functionality. The model was provided with the Trojan requirements and successfully created a testbench that systematically tested the 862-operation threshold.

Reference: See `/task2/ai/prompts.txt` for detailed prompts

Task 3: Wishbone Bus DoS

How the Trojan Works

A Trojan was injected into the Wishbone-UART top module using GHOST with a custom prompt description. The Trojan implements a sophisticated sequence-triggered DoS attack:

- **Trigger Sequence**: The Trojan monitors incoming UART bytes for a specific 4-byte sequence: `0x10`, `0xa4`, `0x98`, `0xbd` (must be received consecutively)
- **DoS Activation**: Once the trigger sequence is detected, the Trojan halts Wishbone bus operations by stopping acknowledgement of bus cycles
- **Recovery Mechanism**: The bus can be restored by receiving the byte `0xfe` four times consecutively
- The original UART and Wishbone functionality remain intact during normal operation.

The GHOST prompt used was:

> "Insert a denial of service Trojan using this strategy: modify the design to add a DoS Trojan which halts the operation of the Wishbone bus after the following 4 bytes have been consecutively received by the UART: 0x10, 0xa4, 0x98, 0xbd. Maintain

original functionality while adding the Trojan. Ensure the design is fully synthesizable."

Testing Methodology

1. **Original Functionality**: The modified core successfully passed the original testbench provided in the GitHub repository, confirming normal UART and Wishbone operations
2. **Trojan Testing**: A custom testbench was generated to specifically test the Trojan trigger sequence and recovery mechanism

Troubleshooting & Design Decisions

Testbench generation required iterative refinement:

- **Tool Used**: GPT-5 was employed for testbench creation
- **Challenge**: The initial testbench did not correctly simulate the trigger sequence or verify the DoS condition
- **Resolution**: Multiple feedback prompts were provided to GPT-4o to refine the testbench until it properly tested both the trigger mechanism and the recovery sequence

AI Interaction Summary

The prompting strategy involved:

1. Initial testbench generation request with Trojan specifications
2. Iterative feedback based on simulation results
3. Refinement prompts to ensure proper sequence detection and bus behaviour verification

Reference: See `/task3/ai/ai-interactions.txt` for complete AI interactions

Task 4: UART Functionality Change

How the Trojan Works

GHOST was used to insert a Trojan into the Wishbone-UART top module that alters the core functionality after activation:

- **Trigger Mechanism**: The Trojan monitors for the byte `0xaf` to be received three times consecutively via UART
- **Functionality Change**: After triggering, all subsequent UART bytes are written to the data register in bit-reversed order
- **Example**: If the byte `0xb2` (binary: `10110010`) is received after the trigger, it is written as `0x4d` (binary: `01001101`) to the data register
- Normal functionality is maintained until the trigger condition is met

The GHOST prompt description used was:

> "Modify the Wishbone-UART core to change the functionality of the device. After the UART receives the byte 0xaf three times, it should then always write the received UART byte to the data register in reverse (for example, after the Trojan triggers if it receives the byte 0xb2, it should write 0x4d to the data register."

Testing Methodology

1. **Original Functionality**: The modified core passed all original testbenches from the GitHub repository
2. **Trojan Testing**: A specialized testbench was created to verify both the trigger mechanism and the bit-reversal functionality

Troubleshooting & Design Decisions

Testbench development required multiple iterations:

- **Tool Used**: Claude Sonnet 3.5 was employed for testbench generation
- **Challenge**: Initial testbench versions did not properly verify the bit-reversal functionality or the three-byte trigger sequence
- **Resolution**: Iterative feedback prompts were provided to Claude to refine the testbench logic and verification steps

AI Interaction Summary

The interaction with Claude Sonnet 3.5 involved:

1. Initial prompt describing the Trojan behaviour and testbench requirements
2. Multiple feedback iterations to correct the testbench logic
3. Final refinement to ensure proper verification of bit-reversed data

Reference: See `/task4/ai/ai_tb_interaction_claude-sonnet-4.5.txt` for complete AI interactions

3. Automated System Details

Tools Used

The primary tool for Trojan insertion was **GHOST** (Generating Hardware Trojan On Verilog Hardware), which automates the process of inserting hardware Trojans into RTL designs while maintaining functional correctness. GHOST itself uses GPT-4 as its underlying language model for code generation and Trojan insertion. GHOST itself uses GPT-4 as its underlying language model for code generation and Trojan insertion.

AI Models for Testbench Generation

Multiple AI models were employed for testbench generation and debugging:

- **GPT-4**: Used for Tasks 1 and 2
    - Testbench generation
    - Debugging timing issues
    - Code refinement
- **GPT-5**: Used for Task 3
    - Testbench generation with iterative refinement
- **Claude Sonnet 3.5**: Used for Task 4
    - Testbench generation
    - Verification logic development

Automation Approach

The general workflow for each task followed this pattern:

1. **Trojan Insertion**: GHOST was used with custom prompt descriptions tailored to each specific Trojan requirement
2. **Functional Verification**: Modified RTL was tested against original testbenches to ensure baseline functionality remained intact
3. **Trojan Testbench Generation**: AI models were prompted to create testbenches that specifically triggered and verified the Trojan behaviour
4. **Iterative Refinement**: Testbenches were refined through feedback loops with AI models when initial versions did not properly verify Trojan functionality
5. **Final Verification**: Complete testing of both original functionality and Trojan-specific behaviour

Tool Modifications

No modifications were made to GHOST itself. The tool was used as-is with custom prompt descriptions for each task. The primary customisation was in the prompt engineering for both GHOST and the AI models used for testbench generation.

4. Conclusion

Summary of Results

All four tasks were completed with functional Trojans that:

- Maintained original design functionality (verified by passing all original testbenches)
- Implemented the specified malicious behaviour (verified by custom testbenches)
- Remained synthesizable and realistic for hardware implementation

Lessons Learned

1. **Timing Considerations**: Hardware Trojans must carefully account for timing constraints, as demonstrated by the initial issues in Task 1
2. **Testbench Complexity**: Generating effective testbenches for Trojans is often more challenging than the Trojan insertion itself, requiring multiple iterations
3. **AI Model Selection**: Different AI models showed varying strengths in testbench generation, suggesting that model selection should be considered for specific tasks
4. **Prompt Engineering**: Clear, specific prompts to GHOST were crucial for generating working Trojans on the first attempt