Automating LLMs for Hardware Trojan Insertion in OpenTitan

Buddhi Perera
Milan Patel

---

1. Introduction / Overview

This report presents the design, automated insertion, and analysis of several hardware Trojans introduced into OpenTitan IP cores using a combination of programmatic tools and large language models (LLMs). The project explores how Trojans can be engineered to remain dormant during normal operation yet produce destructive or malicious effects once triggered.

The Trojans developed in this study span multiple classes:

- **Denial of Service (DoS)**
- **Sensitive information leakage**
- **Functionality modification**
- **Performance degradation**

These Trojans were embedded into OpenTitan's **AES**, **I2C**, and **UART** modules—three subsystems representative of modern SoC security boundaries. AES protects cryptographic operations, I2C handles secure peripheral communication, and UART serves debugging and boot integrity roles. This makes them ideal candidates for exploring Trojan insertion in realistic secure hardware.

Objectives

The primary objectives of this work were to:

1. **Identify suitable IP modules** (AES, I2C, UART) for Trojan insertion based on criticality, internal signal visibility, and vulnerability potential.
2. **Use automated LLM-powered tools**, such as **GHOST**, to generate hardware Trojans and integrate them into RTL with minimal manual effort.
3. **Validate that pre-trigger functionality is preserved**, primarily through OpenTitan smoke tests.
4. **Analyze Trojan triggers, payload implementations, stealth characteristics, and activation behavior.**

5. **Evaluate the role of GPT-5.0 and Sonnet 4.5** in understanding RTL, proposing insertion points, generating trigger logic, and validating safety and stealth.

Through these activities, this report demonstrates how LLMs can accelerate hardware security research, enabling rapid prototyping of sophisticated Trojans that would otherwise require extensive manual RTL engineering.

2. Automated System & Workflow Overview

This section combines the toolset and workflow description into a unified explanation of the automation pipeline used to generate and evaluate Trojans.

2.1 Tools Used

GHOST Trojan-Insertion Framework

GHOST is a semi-automated tool for synthesizable hardware Trojan insertion. It was used to:

- Identify structurally suitable insertion points in OpenTitan RTL.
- Generate custom trigger and payload logic.
- Insert counters, accumulators, state machines, and override circuits.
- Produce synthesizable Verilog modifications without altering primary datapaths.

Notable supported Trojan types utilized in this project:

- DoS (AES, UART, I2C)
- Performance degradation (I2C, UART)
- Functionality modification (I2C)
- Leaking information (not used in final submission but supported)

LLM Models Used (GPT-5.0 and Sonnet 4.5)

LLMs were extensively used to:

- Interpret complex RTL control signals and datapath interactions
- Propose architectural insertion points that minimize detectability
- Generate Trojan code: counters, shift registers, FSMs, trigger conditions
- Explain protocol-level behavior (I2C START conditions, UART FIFO watermarks, AES FSM transitions)
- Validate that inserted logic remains timing- and synthesis-safe
- Prove the absence of functional interference pre-trigger

The LLMs were critical for bridging the gap between design intent and stealth engineering.

OpenTitan DV Smoke Tests

OpenTitan's smoke tests—run using Verilator—were used to:

- Validate that base functionality remained correct
- Detect regressions caused by faulty Trojan insertions
- Confirm that Trojan triggers were not accidentally activated

Smoke tests are functional and limited in scope, which helped demonstrate the stealth of Trojans. However, they could not validate Trojan activation (except for AES DoS) due to limited trigger coverage.

2.2 Automated Workflow

Step 1 — Module Selection

AES, I2C, and UART were chosen because:

- They contain well-defined datapaths suitable for Trojan insertion
- They have externally verifiable behavior via smoke tests
- They implement nontrivial control logic, enabling stealthy triggers

Step 2 — Trojan Generation (Using GHOST + LLMs)

GHOST was guided using LLM-suggested concepts to generate:

- **Counters** (AES DoS)
- **Latch-based override logic** (I2C DoS)
- **64-bit rotating shift registers** (I2C/UART power Trojans)
- **Byte-sequence detectors** (UART DoS)
- **Trigger FSMs** based on internal events such as START detection or watermark thresholds

Step 3 — Controlled RTL Integration

The generated RTL was manually reviewed and integrated:

- Ensured correct clock and reset domains
- Checked for unintended combinational loops
- Structured payload logic to avoid interfering with critical timing paths
- Confirmed synthesizability via tool warnings and LLM inspection

Step 4 — Verification of Pre-Trigger Behaviour

Each modified IP was run through the corresponding smoke tests:

- AES encryption/decryption
- I2C peripheral initialization and transfers
- UART RX/TX path tests

All modules passed without error, confirming:

- No functional interference from inserted Trojan logic
- Trigger conditions were sufficiently rare to avoid unintentional activation

Step 5 — Attempted Trigger Testing

- AES DoS Trojan successfully triggered during controlled simulation by reducing trigger threshold.
- I2C and UART Trojans relied on rare conditions not present in smoke tests, so their payloads remained dormant.
- Power-degradation Trojans activated internally but their effects were not directly visible in functional results.

These outcomes validated stealth properties and demonstrated that some Trojans require specialized hardware-level testing or directed verification to fully exercise.

3. Task Breakdown: Trojans 1–5

Each Trojan is detailed below, including module, mechanism, trigger, validation, and LLM interaction.

Task 1 — AES Denial of Service

Module: `aes_core`

How the Trojan Works

The Trojan tracks AES encryption completions using internal control signals:

```
cipher_op_buf == CIPH_FWD

cipher_out_valid == SP2V_HIGH
```

A counter increments on each completed encryption. After **N encryptions**, it asserts a Trojan-controlled flag that:

- Forces AES into a reset-like state
- Blocks all future encryption/decryption
- Leaves AES permanently unusable until hardware reset

Trigger

- Completion of **N** encryption cycles (configurable)

Testing Methodology

- AES smoke tests passed (baseline functionality unaffected)
- Trigger threshold reduced to test activation
- DoS behavior successfully observed in simulation

AI Interaction Summary

- LLM selected safe internal AES signals for triggering
- Validated the counter logic and disable flag integration
- Ensured no pre-trigger interference

Task 2 — I2C Functionality Change

Module: `i2c_core`

How the Trojan Works

Monitors transmitted bytes from the FMT FIFO. If:

```
byte == 0xFF
```

then:

- A latch permanently sets
- SCL and SDA outputs forcibly driven LOW
- I2C bus enters an illegal static-low state
- Creates irreversible Denial of Service

Trigger

- Transmission of `0xFF`

Testing Methodology

- Smoke tests passed

- Smoke tests do not include 0xFF or exercise internal signals
- Trojan payload not triggered during DV

AI Interaction Summary

- LLM identified correct FMT FIFO signal for tapping
- Validated protocol-compliant override logic
- Ensured Trojan did not accidentally disrupt START/STOP detection

Task 3 — I2C Performance Degradation

Module: `i2c_core`

How the Trojan Works

On the first **start_detect**:

- A 64-bit shift register cycles continuously
- An accumulator updates every cycle
- Massive internal toggling increases dynamic power draw
- No functional outputs are affected

Trigger

- First assertion of `start_detect`

Testing Methodology

- Passed smoke tests
- Trojan activation observed via waveform, but not externally visible

AI Interaction Summary

- LLM designed the shift register + accumulator
- Ensured stealth and invisible side effects

Task 4 — UART Performance Degradation

Module: `uart_core`

How the Trojan Works

When:

```
TX watermark == 1 AND RX watermark == 1
```

the Trojan activates a 64-bit rotating shift register for 1024 cycles, causing:

- Continuous toggling
- Increased power usage
- No visible change in UART behavior

Trigger

- Simultaneous TX and RX FIFO watermark events

Testing Methodology

- Passed smoke tests
- Watermark condition does not naturally occur, so Trojan dormant

AI Interaction Summary

- Verified watermark logic
- Ensured no interference with FIFO control paths

Task 5 — UART Denial of Service

Module: `uart_core`

How the Trojan Works

An internal FSM monitors the RX byte stream for the pattern:

```
0x10 → 0xa4 → 0x98 → 0xbd
```

If observed in order:

- TX and RX logic is disabled
- UART forced into inert state
- Hardware reset required to recover

Trigger

- Exact 4-byte magic sequence

Testing Methodology

- Passed smoke tests
- Smoke tests never include the sequence
- Trojan payload not exercised

AI Interaction Summary

- LLM generated byte-sequence detector FSM
- Ensured false positives were impossible
- Verified compatibility with UART RX timing

---

4. Updated Code with Trojans

All modified RTL files containing Trojan insertions are located under:

```
<task_*>/rtl/<module>.sv
```

Each file includes:

- Trigger logic
- Payload logic
- Required state elements
- Additional registers or combinational blocks

All modified modules passed OpenTitan smoke tests, ensuring:

- **Baseline correctness**
- **Stealth before activation**
- **Synthesizable integration**

Some Trojans (AES DoS) were also activation-tested with modified thresholds.

5. Conclusion

Summary of Results

- Five Trojans successfully inserted across AES, I2C, and UART.
- All designs passed smoke tests, confirming **stealth and correctness**.
- AES Trojan fully validated through controlled activation.
- I2C and UART Trojans relied on rare triggers not covered by smoke tests.

- Power-based Trojans worked internally but remained invisible to functional testing.

Reflections on Automation

- **GHOST** provides rapid and effective Trojan insertion capabilities.
- **LLMs greatly accelerate RTL understanding**, insertion point selection, and debugging.
- Verification remains the biggest challenge; stealth Trojans are inherently hard to test.
- Software-based functional tests cannot fully validate hardware payload effects.
- Large SoC IPs naturally hide Trojans well due to internal complexity.