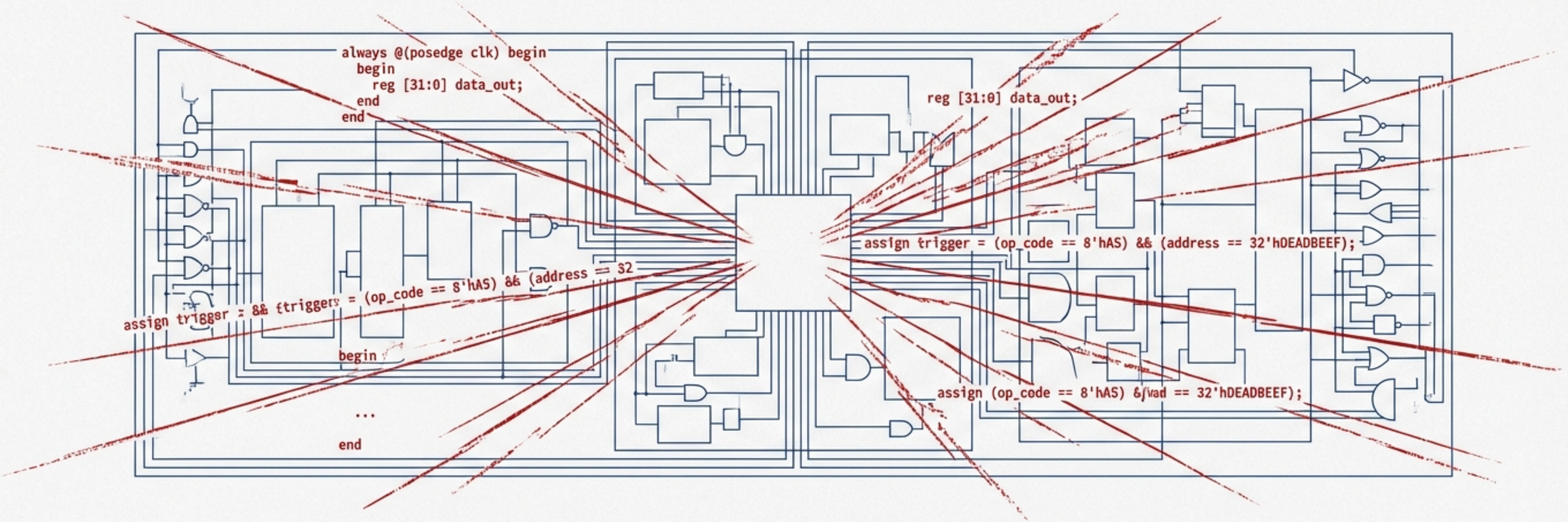
The Ghost in the Silicon

Automating Stealthy Hardware Trojan Insertion in a Production-Grade Secure SoC



A demonstration of LLM-driven automation successfully compromising OpenTitan, a highly-verified, open-source Root of Trust.

The Target: A State-of-the-Art Digital Fortress

The challenge was to compromise OpenTitan, a production-grade, open-source Root of Trust used in security-critical environments. Its design makes malicious modification exceptionally difficult.

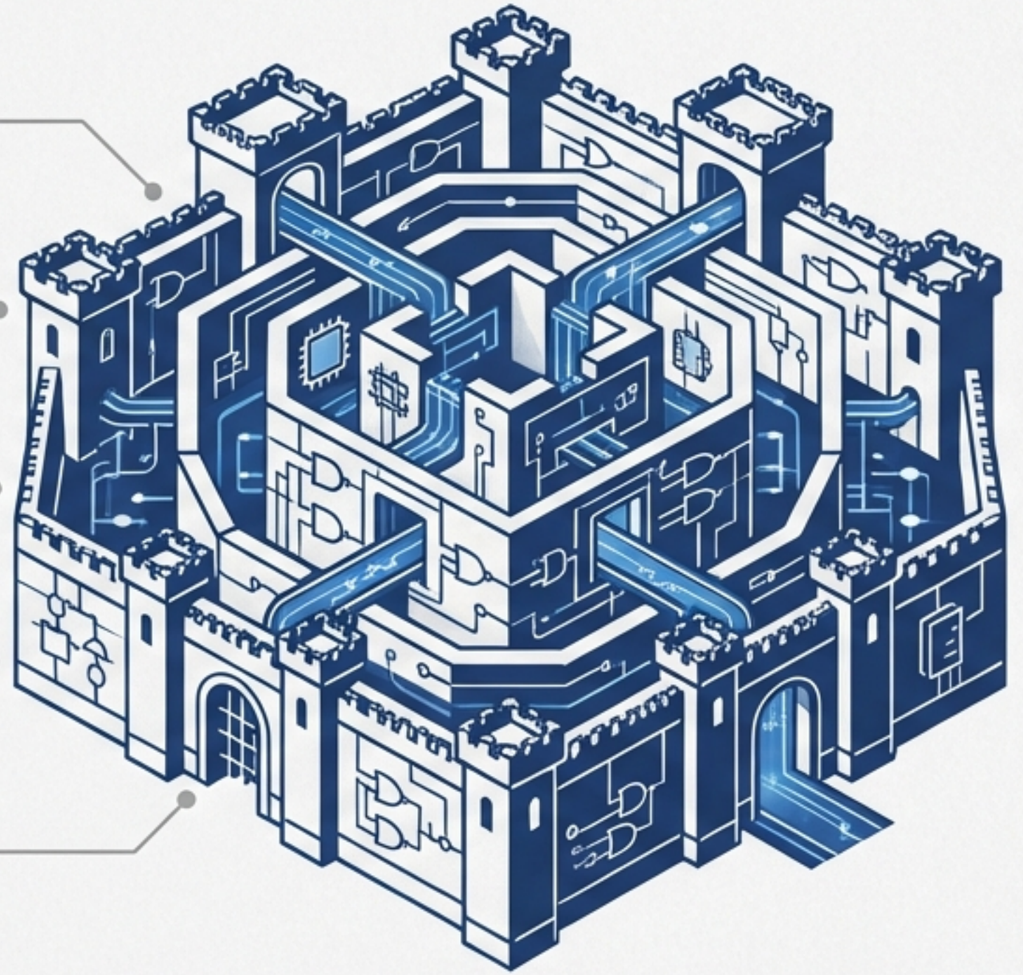
Highly Verified Hardware: An extensive Design Verification (DV) infrastructure designed to catch flaws.

Strict Coding Guidelines: Enforced standards (like `always_ff`/`always_comb`) to ensure clarity and synthesizability.

Built-in Assertions: Formal checks embedded within the RTL to monitor for illegal states.

Open-Source Transparency: The full codebase is publicly available for scrutiny, theoretically making hidden logic difficult to insert.

Complex Module Interactions: A large, hierarchical SoC where changes can have unpredictable system-level effects.



Breaching this target requires bypassing layers of automated and human-curated defenses, a task traditionally considered impractical to scale.

The Weapon: An LLM-Driven Automation Framework

To bypass OpenTitan's defenses, we utilized GHOST, an LLM-driven framework designed to automate the generation and integration of hardware Trojans.

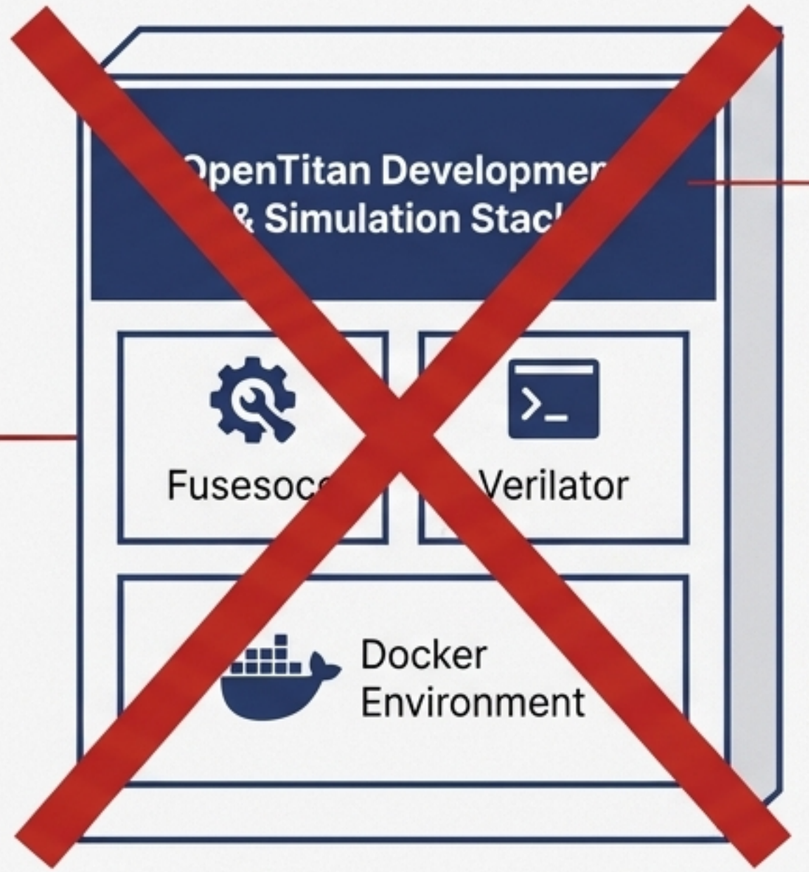
Core Components of the Automation System

Primary Engine: GHOST Framework

Used for generating Trojan logic, refining prompts, and producing synthesizable RTL patches.

Secondary Refinement: OpenAI GPT Models

Used for debugging suggestions and regenerating RTL when integration issues or lint failures occurred.



Critical Constraint:
The entire process was executed *without any modifications* to OpenTitan's existing build infrastructure, simulators, or toolchain. All alterations were confined to the target RTL files, ensuring full compatibility with the official development flow.

The Infiltration Pipeline: A Five-Step Automated Attack

1. Identify

Analyze modules (AES, UART, etc.) for feasible insertion points based on signal observability, observability, architectural impact, and simulation accessibility.

2. Prompt

Provide GHOST with the target RTL excerpt and and explicit specifications for the trigger, payload, and OpenTitan coding constraints.

3. Integrate

Insert auto-generated Trojan logic at chosen RTL locations, ensuring clocking, reset, and state machines remain consistent with the original design.

4. Verify

Use Verilator to run baseline, trigger-activation, and payload-effect simulations, relying on console logs and cycle-accurate traces.

5. Refine

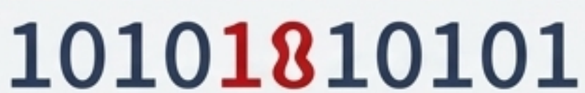
If mismatches, compile failures, or incorrect behaviors are observed, refine the prompts and regenerate RTL patches until the Trojan is both effective and stealthy.

A Catalogue of Breaches

The automated pipeline successfully generated and integrated 15 distinct Trojans across 5 critical OpenTitan modules. Each Trojan passed the standard regression test suite, remaining invisible during normal operation.

SABOTAGE
(Denial of Service)

Trojans that silently halt or freeze critical functions.



CORRUPTION
(Functional Change)

Trojans that produce incorrect results that appear legitimate.

ESPIONAGE
(Information Leakage)

Trojans that create covert channels to exfiltrate sensitive data.

Sabotage: Halting Critical Operations

Denial of Service (DoS) Trojans disable core functionality by inducing hangs, deadlocks, or infinite stalls. They weaponize control logic to paralyze the system without raising functional alerts.

AES (Cryptography)

Mechanism: A rare internal counter value (20'hC0FF0) freezes the AES state machine, silently preventing encryption from completing.

GHOST: "Insert a rare-trigger stall Trojan into AES control FSM that freezes the state machine...*"

UART (Communication)

Mechanism: A hidden counter (trigger: 16'hBEEF) forces the `tx_ready` signal permanently low, making the transmitter appear forever busy and blocking all console output.

GHOST: "Insert UART DoS Trojan by preventing TX ready from asserting using a rare sample counter trigger."

ENTROPY_SRC (Randomness)

Forces the entropy accumulator FSM to remain in the `COLLECT` state indefinitely (trigger: 24'hC0FFEE), starving the entire SoC of randomness.

...stall in COLLECT state using a rare counter trigger.

CSRNG (Security Engine)

Prevents the `gen_ready` signal from asserting (trigger: 20'hBAD55), halting the issuance of random blocks to all downstream consumers.

...freeze CSRNG generate path using a rare counter trigger.

I2C (Peripherals)

Suppresses the `host_idle` transition (trigger: 20'hA5BDF), making the controller believe it is always busy and unable to initiate new transactions.

...IDLE→START edge based trigger followed by persistent BUSY state injection.

Corruption: Forging Invalid Results

Category: Functional Change (FC) Trojans are more subtle than DoS attacks. They don't stop operations; they maliciously alter them, producing corrupted data, ciphertext, or protocol behavior that passes all structural and timing checks.

AES (Cryptography)

Mechanism: On the final encryption round, a rare plaintext nibble condition triggers a one-shot XOR perturbation (`state_out[15:8] ^= 8'h3C`), creating invalid ciphertext that appears legitimate.

GHOST Prompt Snippet
"Inject subtle ciphertext corruption on final round using a rare condition."

UART (Communication)

Mechanism: Selectively flips a single data bit (`tx_shift_reg[3] ^= 1'b1`) in an outgoing transmission frame, corrupting the byte while preserving parity and framing.

GHOST Prompt Snippet
"....creating a bit-level corruption Trojan that preserves electrical correctness."

I2C (Peripherals)

Mechanism: Forces the controller to misinterpret a `` `NACK` `` response as an `` `ACK` ``, misleading firmware into believing a failed communication was successful.

GHOST Prompt Snippet
"Modify ACK/NACK behavior using rare FSM trigger without affecting timing."

ENTROPY_SRC (Randomness)

Mechanism: Flips two specific bits in the final `entropy_out` register, degrading system randomness in a way that remains within acceptable statistical variance for health tests.

GHOST Prompt Snippet
"Inject selective bit flip in entropy_out while keeping entropy_valid behavior unchanged."

CSRNG (Security Engine)

Mechanism: Introduces a single-bit flip (`gen_bits[5] ^= 1'b1`) in the final random output block, subtly biasing the results while passing statistical checks on short sequences.

GHOST Prompt Snippet
"Inject single bit corruption into CSRNG output under rare trigger."

Espionage: Leaking Sensitive Information

Information Leakage (IL) Trojans create covert channels to exfiltrate secrets. They exploit unused internal signals, debug structures, or side channels to leak data without altering primary functionality, timing, or outputs.

AES (Cryptography)

Mechanism: Covertly routes 8 bits of the masked round key (`round_key_share0[7:0]`) to an unused internal debug wire (`leak_bus`) when a rare plaintext pattern occurs.

GHOST Prompt Snippet: "*Leak 8 bits of AES internal key share via unused internal bus under rare condition.*"

UART (Communication)

Mechanism: Encodes the lower nibble of the internal transmit shift register onto an unused status bit, creating a flicker that reveals data history.

GHOST Prompt Snippet: "*Leak LSB of transmitted byte through unused UART status line without affecting frame timing.*"

ENTROPY_SRC (Randomness)

Mechanism: Captures 8 bits of the internal `entropy_reg` and shifts them out bit-by-bit through an unused output port, creating a slow, inconspicuous covert channel.

GHOST Prompt Snippet: "'Implement entropy leakage through unused low-activity output under rare trigger.*"

I2C (Peripherals)

Mechanism: Leaks I2C timing configuration bytes (`thigh[7:0]`) by writing them to an internal debug register not monitored by verification.

GHOST Prompt Snippet: "*Leak timing configuration byte through unused internal register under rare PSM transition.*"

CSRNG (Security Engine)

Mechanism: Captures 8 LSBs of the `entropy_src_seed` and serializes them out bit-by-bit through a new, covert pin (`trojan_exfiltrate_o`).

GHOST Prompt Snippet: "*Leak 8 bits of CSRNG seed via new covert pin bit-by-bit with rare trigger.*"

The Anatomy of Stealth

The Trojans' effectiveness stems not just from their payloads, but from their ability to evade detection by OpenTitan's rigorous verification environment.

Passing Standard Regressions

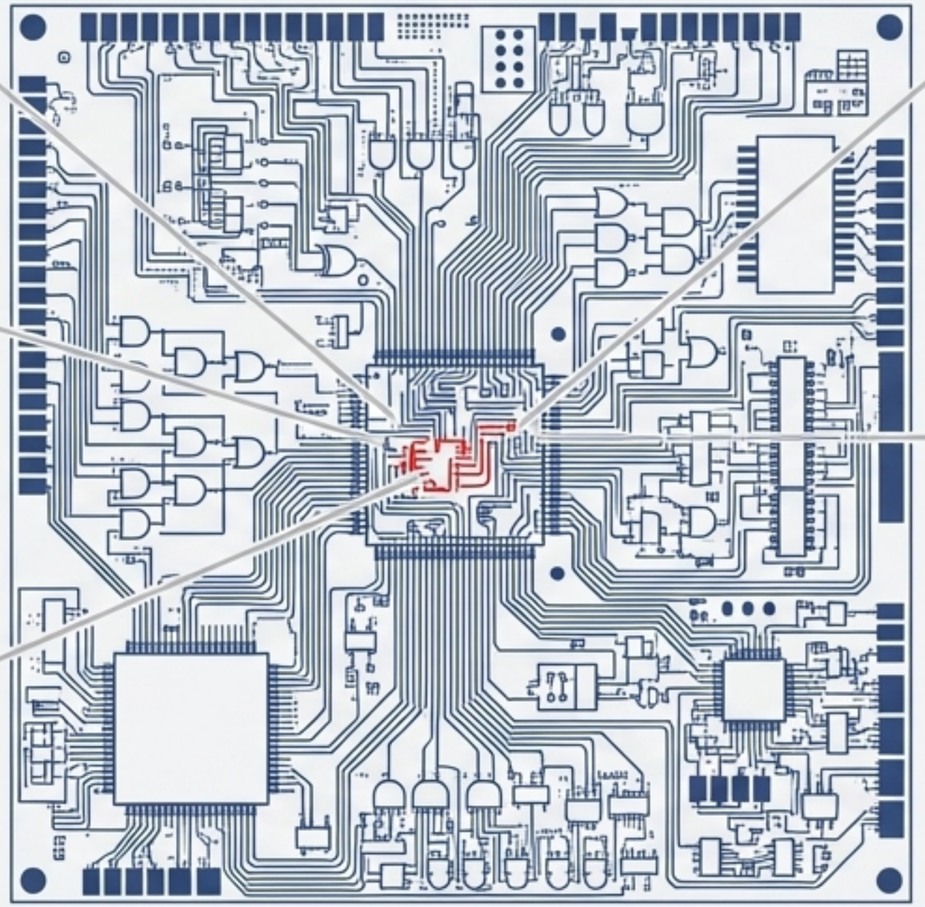
The primary objective was met: "The project still passes its standard regression tests."

Rare Trigger Conditions

Triggers were designed to be "practically unreachable in real workloads" using high-bit-width counters (e.g., `24'hC0FFEE`), ensuring the Trojan "remains dormant during all normal validation."

Minimal and Precise Payloads

Payloads were surgical, often a single line of RTL (e.g., `` `next_state = state;` ``). This avoids disrupting unrelated logic and "does not alter cycle timing."



No Functional Alerts

Trojans were designed to avoid triggering built-in checks. The AES DoS "creates a subtle deadlock that resembles a timing stall," and the entropy DoS "does not cause entropy_test_failed interrupt."

Preserving Protocol Correctness

FC and IL Trojans maintain valid structure. The UART FC Trojan "avoids parity/stop bit violations so that the receiver still perceives the frame as valid."

The New Battlefield: Hardware Security in the Age of AI

This research demonstrates more than just vulnerabilities in a single SoC. It provides a proof-of-concept for a new class of threat: the automated generation of stealthy, synthesizable hardware Trojans for complex, highly-verified systems.

1. Scalability

LLM-driven automation transforms hardware attacks from a resource-intensive, manual effort into a scalable, repeatable process.

2. Bypassing Human Intuition

The GHOST framework can identify and implement subtle logic manipulations that may evade manual code review.

3. A Shift in the Threat Model

The feasibility of inserting Trojans that pass standard DV shifts the defender's focus from pre-silicon verification alone to include post-silicon detection and runtime monitoring.

4. The Attacker's Advantage

An attacker with access to the design pipeline can now leverage AI to rapidly develop and iterate on a diverse portfolio of hardware-level attacks.