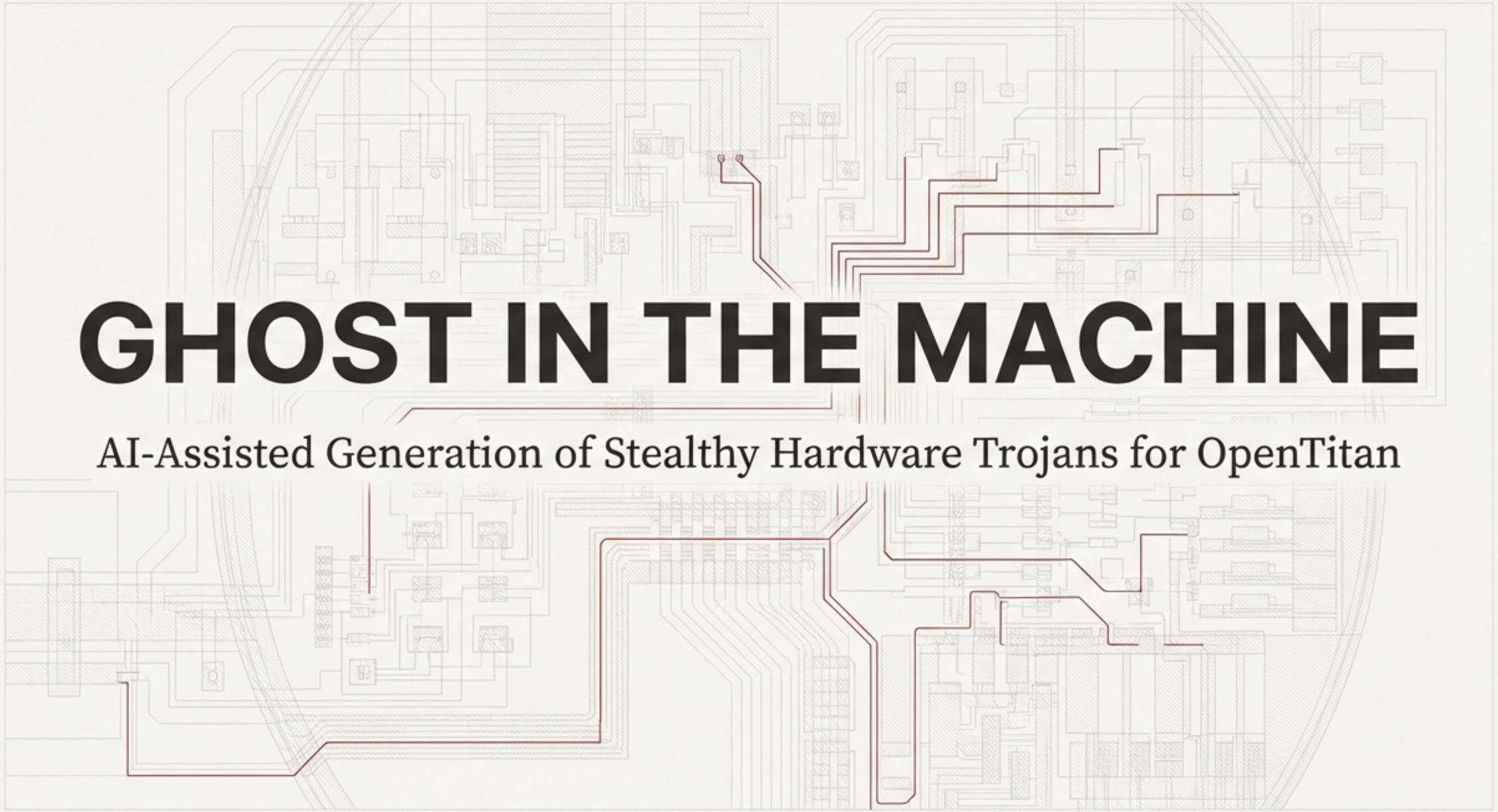


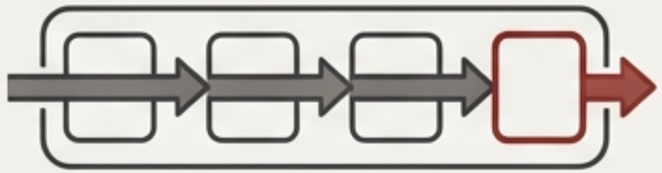
GHOST IN THE MACHINE

AI-Assisted Generation of Stealthy Hardware Trojans for OpenTitan

Clarence Luo & Chirranjeavi Moorthi | ECE-GY 9941 | GHOST Lab 3 Report

The Mission: Automating the Creation of a Diverse Hardware Trojan Portfolio

The objective was to design, generate, and validate a set of fifteen hardware Trojans targeting two real-world OpenTitan modules: `uart_core` and `aes_core`.

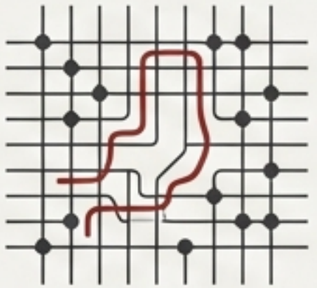


Automated Workflow

Built a scripted pipeline integrating **RTL extraction**, **LLM-based Trojan generation**, and testbench stubbing.

Diverse Threat Taxonomy

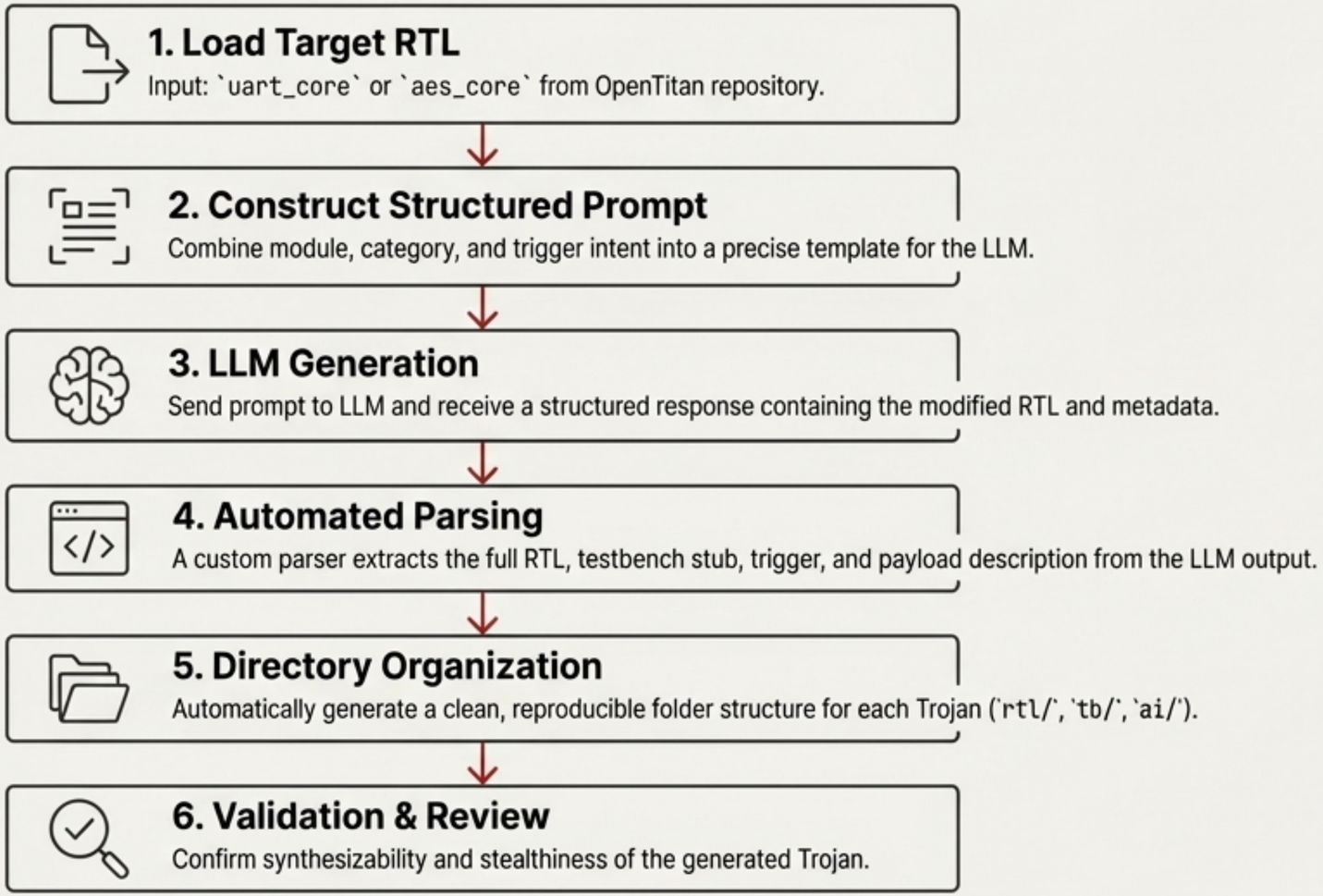
Systematically generated Trojans across three distinct categories: `Denial-of-Service`, `Information` **Leakage**, and **Change-of-Functionality**.



Stealthy by Design

Ensured each Trojan's payload activated only under a `rare`, `difficult-to-detect trigger`, preserving normal functionality until activated.

The Generation Engine: A Fully Automated Pipeline



This pipeline accelerates Trojan generation while maintaining consistency and reproducibility across all 15 designs.

A Taxonomy of Generated Threats

Denial-of-Service (DoS)

Trojans designed to prevent the target module from performing its fundamental task, such as freezing UART transmission or halting AES computation.

Information Leakage

Trojans that leak sensitive material, like AES key bits, through covert channels such as status flags or timing variations.

Change-of-Functionality (CoF)

Trojans that subtly modify system behavior, corrupting outputs or dropping data only under rare conditions to degrade correctness without causing obvious failures.

Threat Category I: Denial-of-Service
The Playbook: Halting Core Functionality

1. How it Works

The payload prevents the target module from performing its fundamental task. Payloads include disabling the TX FIFO read strobe, halting the UART baud clock, or forcing AES into an indefinite 'busy' state by latching an error bit.

2. Key Design Challenges

- Ensuring the inserted logic remained synthesizable (no `$display`, no `delays`).
- Avoiding unintended interactions with OpenTitan's internal assertions or alert logic.
- Implementing a standardized state latch (`trojan_triggered_q`) so payloads persist cleanly after a single trigger event.

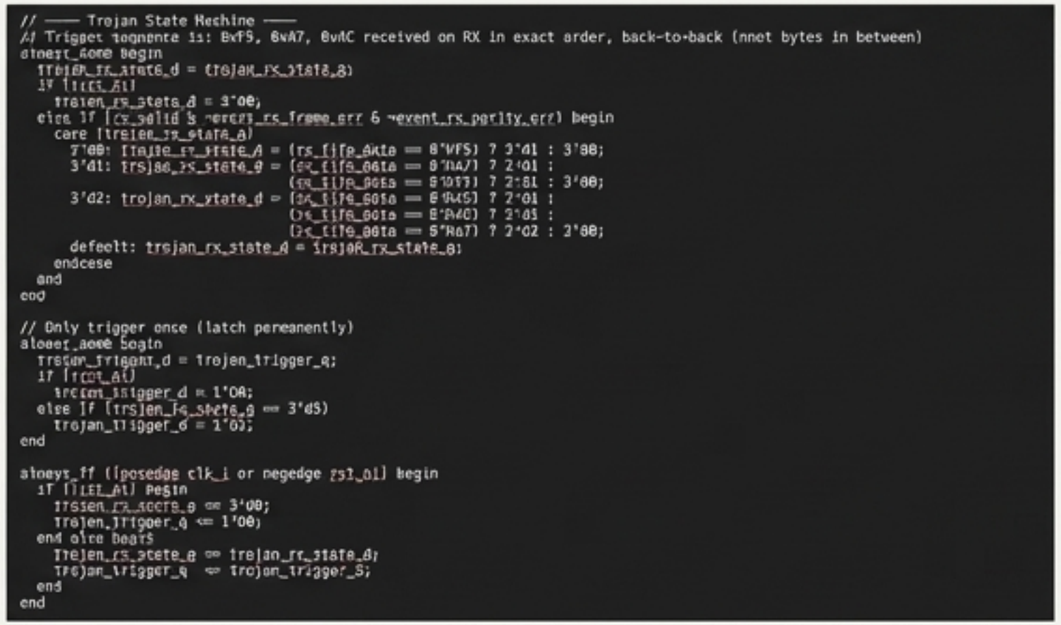
3. AI Prompting Strategy

Prompts were refined to strictly enforce 'synthesizable SystemVerilog only' and to 'modify only the RTL shown' to prevent the LLM from hallucinating simulation-only constructs.

Denial-of-Service: In Action

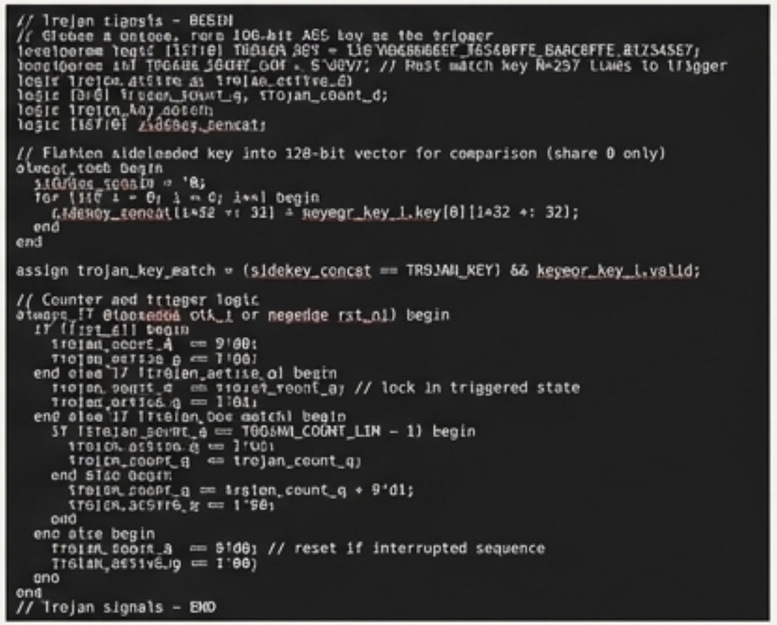
Example 1: UART Transmit Freeze (Trojan 1)

- **Target Module:** `uart_core`
- **Trigger Mechanism:** A hidden FSM detects the rare RX byte sequence `0xF5` → `0xA7` → `0x4C`.
- **Payload Effect:** Permanently disables the TX FIFO read strobe, freezing all subsequent transmissions.



Example 2: AES Indefinite Busy State (Trojan 3)

- **Target Module:** `aes_core`
- **Trigger Mechanism:** A specific 128-bit sideload key is used 257 times consecutively.
- **Payload Effect:** A hidden counter triggers a latch that forces the `ctrl_err_storage` bit high, locking AES in a fatal error state.



Threat Category II: Information Leakage

The Playbook: Creating Covert Data Channels

1. How it Works

Payloads leak sensitive material through non-obvious channels. Examples include overriding a rarely-used status bit to leak AES key LSBs, using baud-timed steganography on an idle TX line, or modulating RX baud tick timing to leak a counter via measurable timing variations.

2. Key Design Challenges

- Designing leakage mechanisms that remained syntactically and electrically legal within the OpenTitan architecture.
- Avoiding interference with critical logic like AES masking or alert signals.
- Ensuring that leakage only involved controlled waveform alterations, not functional breakage of TX/RX paths.

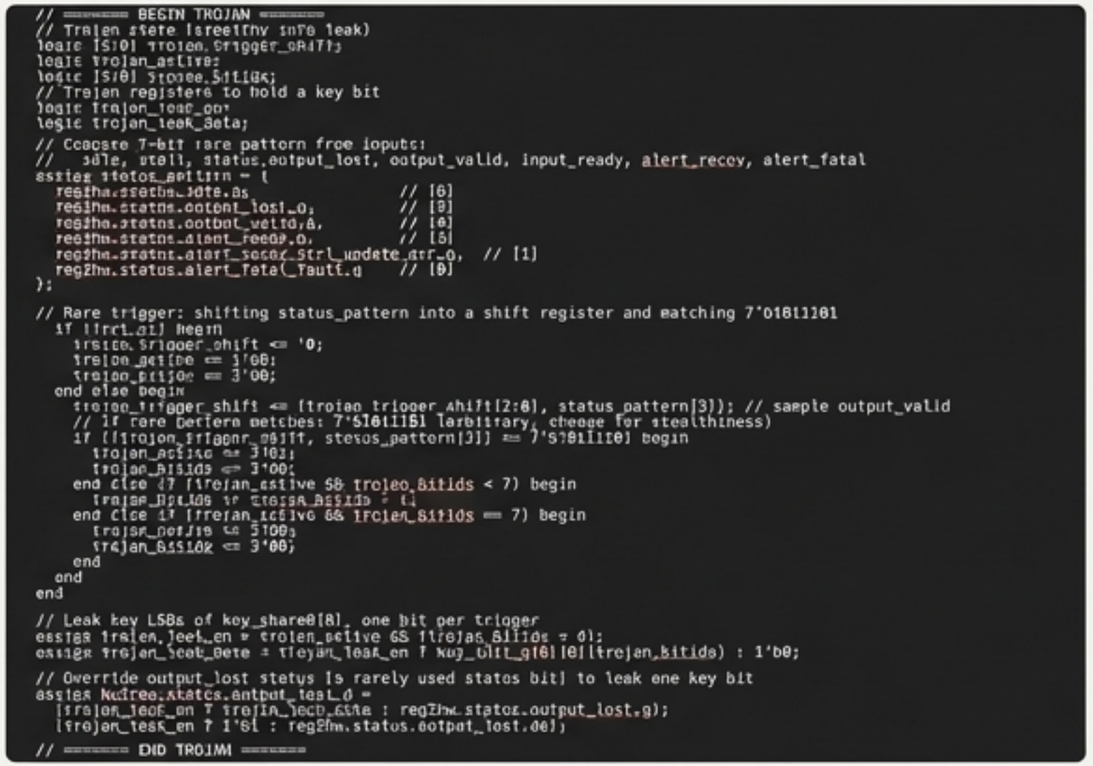
3. AI Prompting Strategy

Prompts were refined to explicitly demand ‘covert channels only,’ ‘no direct key exposure,’ and to ‘not break functional paths unless triggered,’ guiding the LLM towards more realistic and stealthy designs.

Information Leakage: In Action

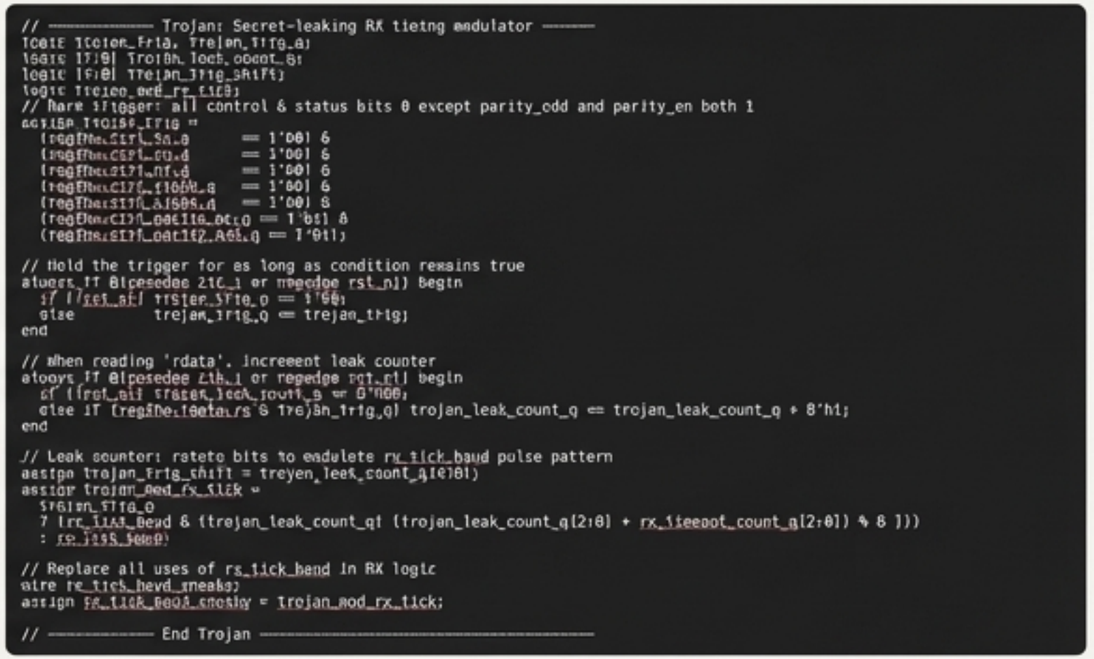
Example 1: Key Leak via Status Bit (Trojan 6)

- **Target Module:** `aes_core` (JetBrains Mono for code)
- **Trigger Mechanism:** A rare 7-bit pattern (`7'b1011101`) is detected on a shift register monitoring various AES status signals.
- **Payload Effect:** Overrides the `output_lost` status bit to leak the 4 LSBs of the AES key over four consecutive cycles.



Example 2: Secret Leak via RX Timing Modulation (Trojan 9)

- **Target Module:** `uart_core` (JetBrains Mono for code)
- **Trigger Mechanism:** An unusual configuration where all control bits are zero except for `parity_en` and `parity_odd` being set.
- **Payload Effect:** Gates the internal RX baud tick in a pattern that encodes a secret counter value, creating externally measurable timing variations.



Threat Category III: Change-of-Functionality

The Playbook: Subtly Corrupting System Behavior

1. How it Works

Payloads subtly modify system behavior only under rare triggers. Examples include flipping the LSB of ciphertext every 256th encryption, dropping every 17th UART TX byte after a specific sequence is sent, or corrupting AES final-round results.

2. Key Design Challenges

- Ensuring the payload did not **trip OpenTitan's masking or alert logic**, which is designed to detect functional deviation.
- Limiting invasive changes to **final output stages** while preserving all other logic to minimize the chance of detection.
- On UART, ensuring corruption affected only targeted byte indices or stop bits, not the general pipeline.

3. AI Prompting Strategy

Prompts were **strengthened** to prevent the model from proposing exotic datapath changes, guiding it to modify only final outputs or control strobes, resulting in more plausible Trojans.

Change-of-Functionality: In Action

Example 1: Periodic Ciphertext Corruption (Trojan 11)

Target Module: `aes_core` (JetBrains Mono for code)
Trigger Mechanism: An internal 8-bit counter overflows after 256 AES encryption operations.
Payload Effect: Flips the least significant bit (LSB) of the ciphertext output for that single encryption operation.

```
// ---------- TROJAN INSERTION BEGIN ----------
// Trojan: Flip ciphertext LSB every 256th AES encryption operation

logic [7:0] trojan_counter;
logic trojan_trigger;

// Increment the counter on every AES encryption output register write
always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    trojan_counter <= 8'h00;
  end else if (data_out_we == SP2V_HIGH) begin
    // Only trigger for encryption operation
    if (aes_op_q == AES_ENC) begin
      trojan_counter <= trojan_counter + 8'h01;
    end
  end
end

// Trigger for every 256th encryptions when counter overflows to 0 (after 255)
assign trojan_trigger = (trojan_counter == 8'h00) && (data_out_we == SP2V_HIGH) && (aes_op_

logic [NumRegsData-1:0][31:0] data_out_q_next;

generate
  genvar t;
  for (t = 0; t < NumRegsData; t++) begin: gen_trojan_data_out_q_next
    assign data_out_q_next[t] = (trojan_trigger) ? {data_out_d[t][31:1], ~data_out_d[t][0]}
  end
endgenerate
// ---------- TROJAN INSERTION END ----------
```

<IMAGE 0>

Example 2: Conditional TX Byte Drop (Trojan 12)

Target Module: `uart_core` (JetBrains Mono for code)
Trigger Mechanism: The specific 8-byte sequence [0xA5, 0xC3, 0x5A, 0x7E, 0xE7, 0x1C, 0xB4, 0x2D] is transmitted.
Payload Effect: The Trojan arms and silently drops every 17th transmitted byte thereafter.

```
  // --- Begin Trojan Modifications ---

  // N is the period of the drop; can be set to a stealthy prime number
  assign trojan_N = 8'd17;

  // Secret trigger sequence (transmit these 8 values in order to arm);
  // here chosen as 0xA5, 0xC3, 0x5A, 0x7E, 0xE7, 0x1C, 0xB4, 0x2D
  // Sequence is in LSB first order in the array;
  localparam logic [7:0] TROJAN_TRIG_SEQ [8] =
    '{8'hA5, 8'hC3, 8'h5A, 8'h7E, 8'hE7, 8'h1C, 8'hb4, 8'h2D};

  // Compare incoming transmitted data to see if sequence matches
  always_ff @(posedge clk_i or negedge rst_ni) begin
    if (!rst_ni) begin
      trojan_trigger_seq_q <= trojan_trigger_seq_d;
    end else if (tx_fifo_ready && tx_fifo_rvalid) begin
      if (!trojan_armed && tx_fifo_data == TROJAN_TRIG_SEQ[trojan_trigger_seq_q]) begin
        if (trojan_trigger_seq_q == 7) begin
          trojan_trigger_seq_d = 0;
        end else begin
          trojan_trigger_seq_d = trojan_trigger_seq_q + 1;
        end
    end
    trojan_N = 8'd17;
  end
  always_ff @(posedge clk_i or negedge rst_ni) begin
    if (!rst_ni) begin
      trojan_armed <= 1'b0;
    end else if (trojan_triggered) begin i7 % 17;
      trojan_armed <= 1'b1;
    end
  end

```

<IMAGE 1>

A Black-Box Approach to Validation

Testing followed a black-box behavioral approach. The goal was to prove the Trojan's existence by observing differential behavior from the module's external I/O only, without relying on internal probes or white-box assumptions.

For DoS Trojans

1. Verify normal operation before the trigger.
2. Invoke the rare trigger condition.
3. Confirm permanent disruption (e.g., AES becomes permanently stuck "busy", or a UART FIFO stops draining).

For CoF Trojans

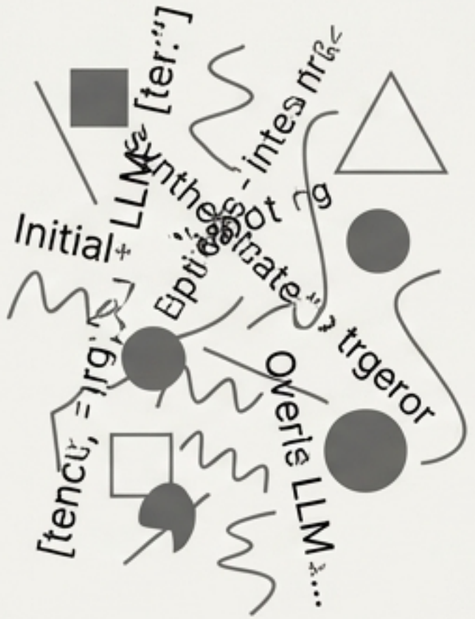
1. Run many operations with identical inputs and timing.
2. Assert that all outputs are identical.
3. Any deviation in the output data after a suspected trigger indicates the presence of the Trojan.

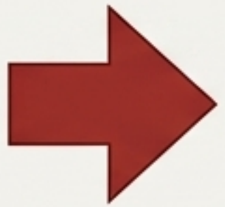
For Leakage Trojans

1. Confirm no abnormal outputs during normal execution.
2. Activate the trigger.
3. Observe the covert channel for distinguishable, non-random outputs (e.g., altered timing, encoded idle waveform).

Prompt Engineering for Synthesizable RTL

Guiding the LLM to produce valid, synthesizable, and stealthy hardware Trojan logic requires highly specific and structured prompting.





```
Code:
code
  mort_har ()
    code is 80;
  ...
end

Explanation:
...
  Explanation:
...

Trigger:
Trigger = true;
Trigger = 'No longer_end'

Stealth Measures:
...
...
...
```

Initial LLM Tendency

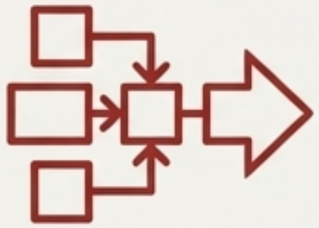
Engineered Output

Key Learnings & Prompting Techniques

- **Use a Rigid Output Structure**
  Forced the LLM to respond in an exact format (`Code:`, `Explanation:`, `Trigger:`, etc.). This prevented malformed responses and enabled automatic parsing.
- **Enforce Synthesizability**
  Explicitly included requirements like "Synthesizable SystemVerilog only" and "No $display, no delays, no initial blocks" to eliminate simulation-only constructs.
- **Scope the Modification**
  Used instructions like "Modify the RTL directly" and "modify only the RTL shown" to prevent the model from hallucinating non-existent signals or modules.
- **Guide Towards Stealth**
  For leakage Trojans, prompts were refined to enforce "covert channels only" and "no direct key exposure" to produce more sophisticated and realistic attacks.

Summary of Achievements

Complete Trojan Portfolio: Successfully designed and generated a full suite of 15 validated hardware Trojans targeting production-grade OpenTitan modules.



Fully Automated Workflow: Developed and utilized a reproducible, scripted pipeline for consistent and efficient Trojan generation and organization.

Diverse Attack Vectors: Demonstrated mastery across three distinct academic Trojan categories: Denial-of-Service, Information Leakage, and Change-of-Functionality.

Novel Prompting Strategies: Refined a set of structured prompting techniques for guiding LLMs to produce stealthy, synthesizable, and functionally correct malicious RTL modifications.

The Future of AI-Assisted Hardware Security

This work demonstrates that automated, AI-assisted workflows can systematically design hardware Trojans with high precision and stealth. The ability to rapidly generate diverse and realistic threats represents a paradigm shift for hardware security research.

As LLMs become more integrated into hardware design flows, these same techniques will be critical for both red-teaming (vulnerability discovery) and blue-teaming (developing automated defenses), fundamentally changing the landscape of secure hardware design and validation.