SEAL THE TROJAN

CSAW: AI Hardware Attack Challenge

Team Name: SEAL

Shubhi Shukla

Tishya Sharma Sarkar

Upasana Mandal

Kislay Arya

Prof. Debdeep Mukhopadhyay

Indian Institute Of Technology, Kharagpur

Overview

- Hardware Trojans & LLMs
- Proposed Methodology
- Results

Hardware Trojans & LLMs

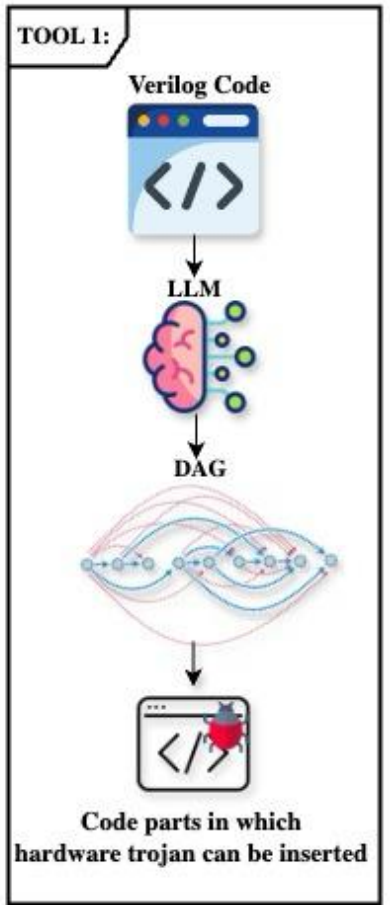
A Large Language Model (LLM) is a deep learning model trained on vast amounts of text data to understand, generate, and analyze human language by predicting word sequences based on context.

A hardware Trojan is a malicious modification in a hardware design or component that enables unauthorized access, control, or sabotage, often remaining stealthy to evade detection.

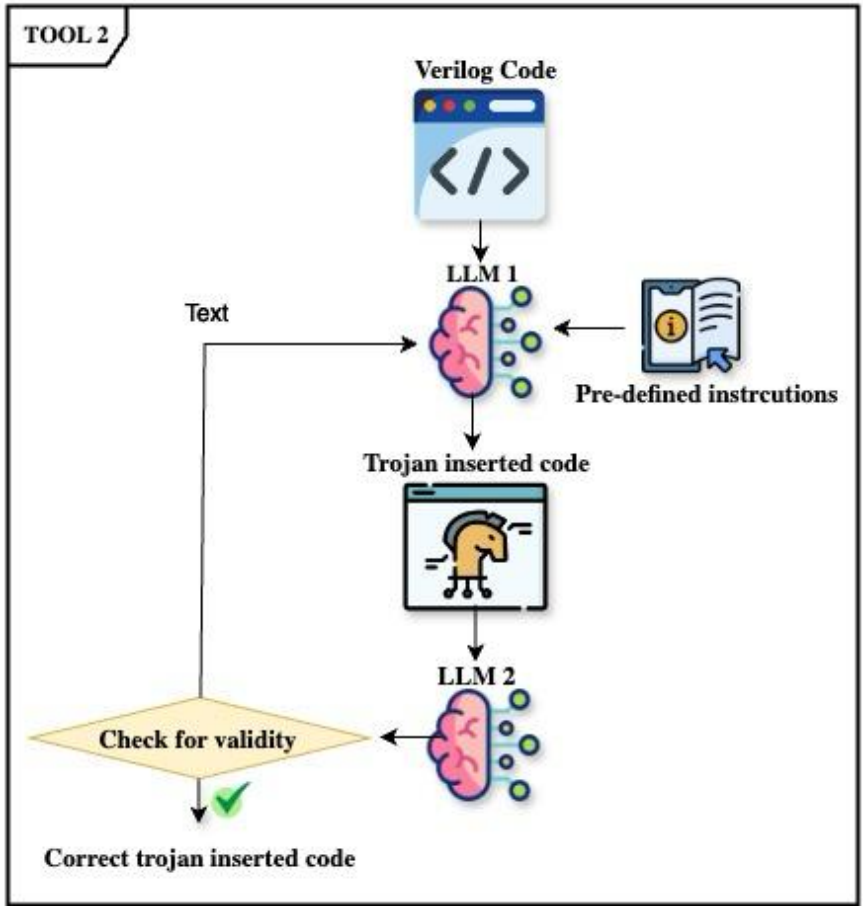
Proposed Methodology

- Tool 1: Identifying Exploitable Locations in Verilog Code:
  - Verilog code converted into DAG
  - DAG evaluated to identify exploitable locations
- Tool 2: Hardware Trojan Insertion and Validation:
  - LLM1 has generates the trojan-inserted code
  - LLM2 checks for the correctness of the code.

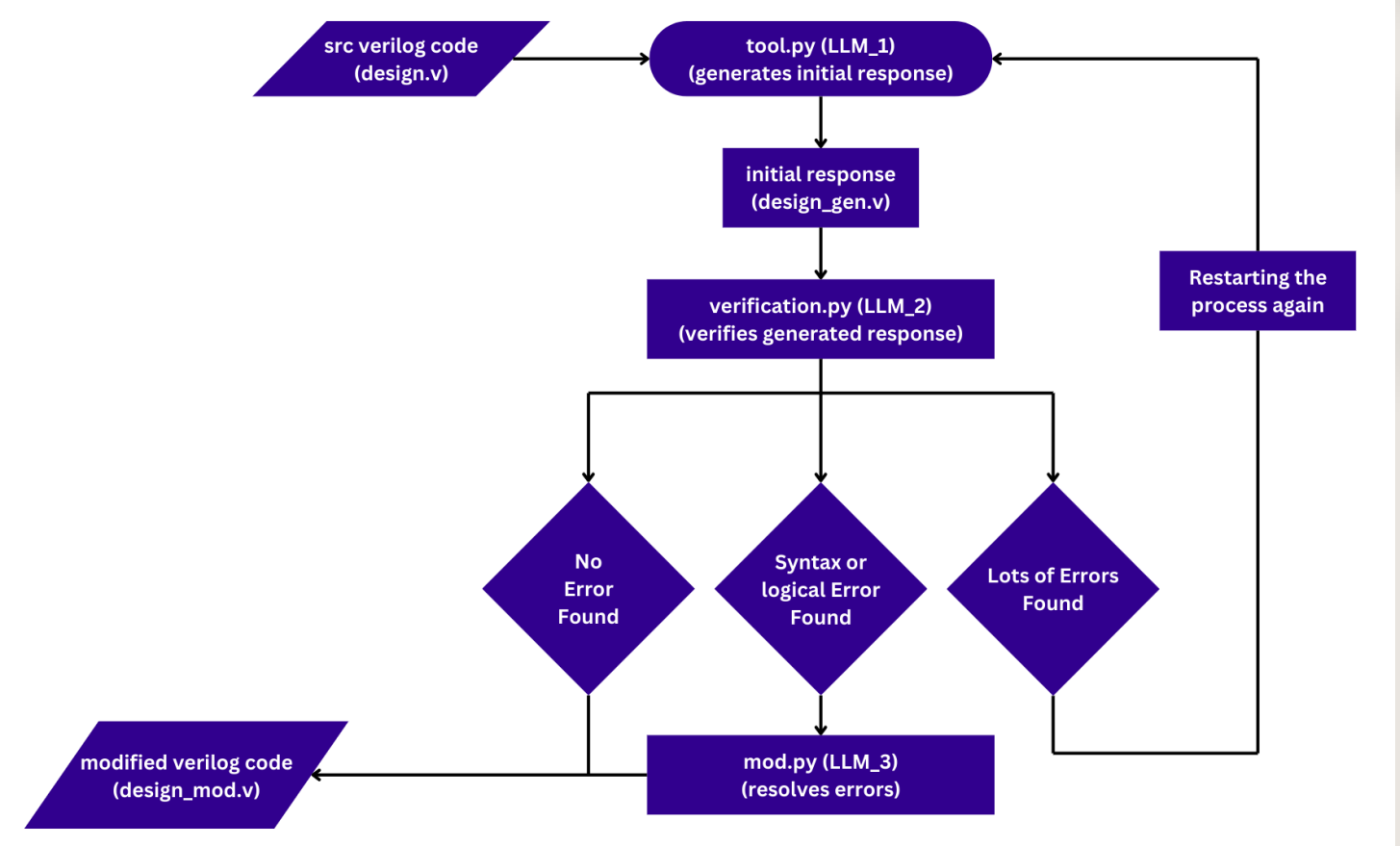
Design Vulnerabilities Identification



Trojan Insertion and Validation



Proposed Methodology



Tool 2 Flowchart: Verilog Code generation and Validation

Key Features:

- The tools are hardware-agnostic and compatible with any Verilog code.
- Users can choose between two leading large language models, Google's Gemini and OpenAI's GPT-4, when initializing the tool.
- We offer both CLI and UI versions of our tool for user convenience.
- An additional validation feature is included to verify the correctness of the code generated by the LLM, which re-queries the LLM if inaccuracies are detected.

Results

❑Tool 1 Result:

:

```
always@(*)
begin
case(pstate)
    6'd0: nstate<=6'd1;
    6'd1:
       begin
           if(karatDone==1)
               nstate<=6'd2;
           else
               nstate<=6'd1;
       end
```

```
    6'd37:
       begin
            if(karatDone==1)
                nstate<=6'd38;
            else
                nstate<=6'd37;
       end
    6'd38: nstate<=6'd39;
    6'd39: nstate<=6'd40;
    6'd40: nstate<=6'd41;
    default: nstate<=6'd41;
    endcase
```

VI TOOL : Vulnerable Points

The primary vulnerable points in this design are:

1. **`pstate` Register:** This register's value dictates the entire behavior of the module. Altering its value can drastically change the sequence of operations. This is a critical point for control flow hijacking.

2. **`karatDone` Input:** The `karatDone` signal acts as a synchronization point. A malicious actor could manipulate this signal, altering the FSM's progression and potentially causing unwanted operations.

3. **`nstate` Combinational Logic:** The logic determining the next state (`nstate`) is complex and spread across the `case` statement. Introducing a Trojan in this logic could subtly change the state transitions, creating hard-to-detect errors.

4. **Output Signals (`addra`, `addrb`, `mulOn`, `mulEnd`, `outLoop`, `karatRst`):** Although not directly storage elements, manipulation of these outputs can lead to significant consequences if a trojan were to influence their values unexpectedly.

Results

❑Tool 1 Result:

```
function [7:0] mb2; //multiply by 2
input [7:0] x;
begin
/* multiplication by 2 is shifting on bit to the left, and if the original 8 bits had a 1 @ MSB,
xor the result with {1b}*/
if(x[7] == 1) mb2 = ((x << 1) ^ 8'h1b);
else mb2 = x << 1;
end
endfunction
```

VI TOOL : 1. **Data Modification in `mb2`:** A Trojan could subtly alter the output of the `mb2` function under specific conditions. This could involve flipping a bit in the result based on a trigger condition (e.g., a specific sequence of input bits). The trigger could be made very difficult to detect by relying on patterns that occur rarely in the input data. This is stealthy because the change is only applied in the rare scenarios determined by the input.

2. **Conditional XOR Manipulation:** A Trojan could conditionally modify one of the XOR operations within the generate loop. The trigger could be based on a combination of clock cycles and input data patterns, adding another layer of stealth. The payload could involve adding an extra, conditional XOR operation to tamper with specific bits of the result.

3. **Bit-Flipping in `mb3`:** Similar to the `mb2` Trojan, a bit could be flipped conditionally in `mb3`. The trigger could be based on the value of specific bits of the input. This Trojan could modify the result conditionally and the modified value might propagate to further stages of the AES encryption.

4. **Delayed Output Modification:** A Trojan could delay the correct output by a few clock cycles in specific circumstances, determined by a hidden counter triggered by a rare input pattern. This creates a timing-based Trojan and would be difficult to detect without sophisticated analysis.

5. **Data-Dependent Delay in `mb2`/`mb3`:** The Trojan introduces a variable delay in the `mb2` or `mb3` function, dependent on the input data. If the input data matches a certain trigger, the delay increases, leading to a timing-based attack, possibly causing a delay in the overall encryption process. The added delay is kept small and depends on the input data, thus making detection more difficult.

Results:

```
function [7:0] mb2; //multiply by 2
    input [7:0] x;
    begin
            /* multiplication by 2 is shifting on bit
            xor the result with {1b}*/
            if(x[7] == 1) mb2 = ((x << 1) ^ 8'h1b);
            else mb2 = x << 1;
    end
endfunction
```

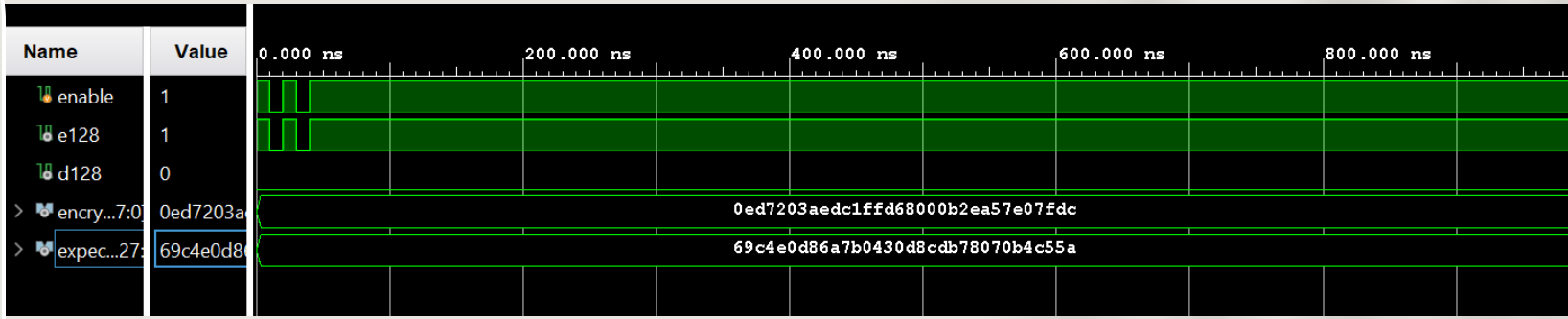


Fig: Simulation results of AES operation without Operator Trojan

TI TOOL:

```
function [7:0] mb2; // multiply by 2
input [7:0] x;
begin
// Trojan condition: If the least significant bit of `sta
if (state_in[0] == 0) begin
if (x[7] == 1)
mb2 = ((x << 2) ^ 8'h1b); // Shift by 2 instead of 1
else
mb2 = x << 2;
end else begin
// Original behavior when `state_in` is odd
if (x[7] == 1)
mb2 = ((x << 1) ^ 8'h1b);
else
mb2 = x << 1;
end
end
endfunction
```

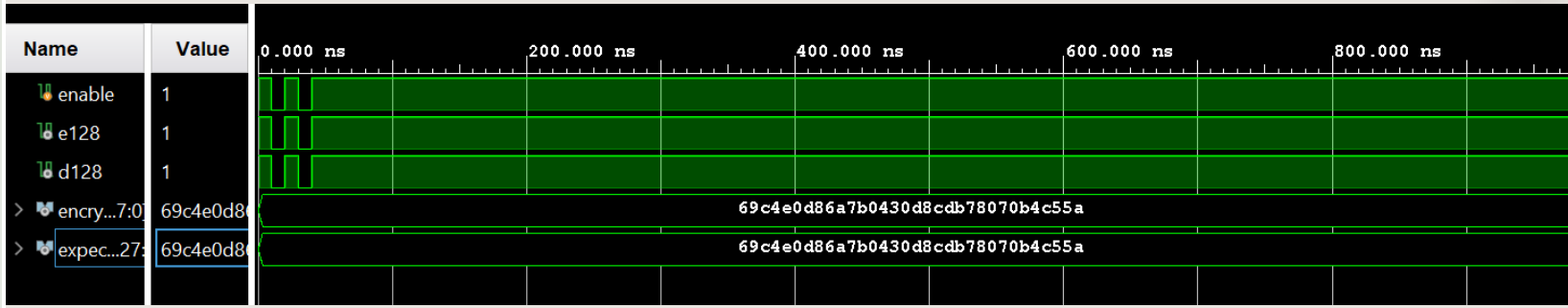


Fig: Simulation results of AES operation with Operator Trojan.

TI: Trojan Insertion

Results:

:

```
        end
    6'd38: nstate<=6'd39;
    6'd39: nstate<=6'd40;
    6'd40: nstate<=6'd41;
    default: nstate<=6'd41;
    endcase
end
```

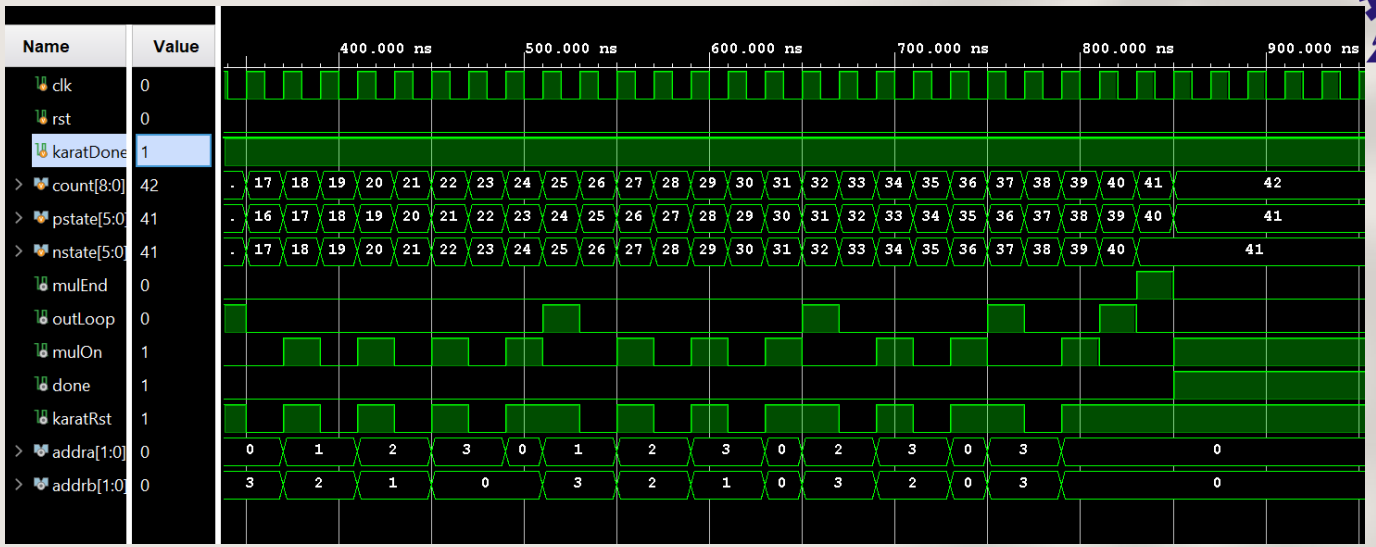


Fig: Simulation results of an FSM without any hardware Trojan intervention

TI TOOL:

```
default:
    begin
        mulOn<=1'b1; // Trc
        mulEnd<=1'd0;
        outLoop<=1'd0;
        addra<=2'd0;
        addrb<=2'd0;
        karatRst<=1'd1;
    end
endcase
```

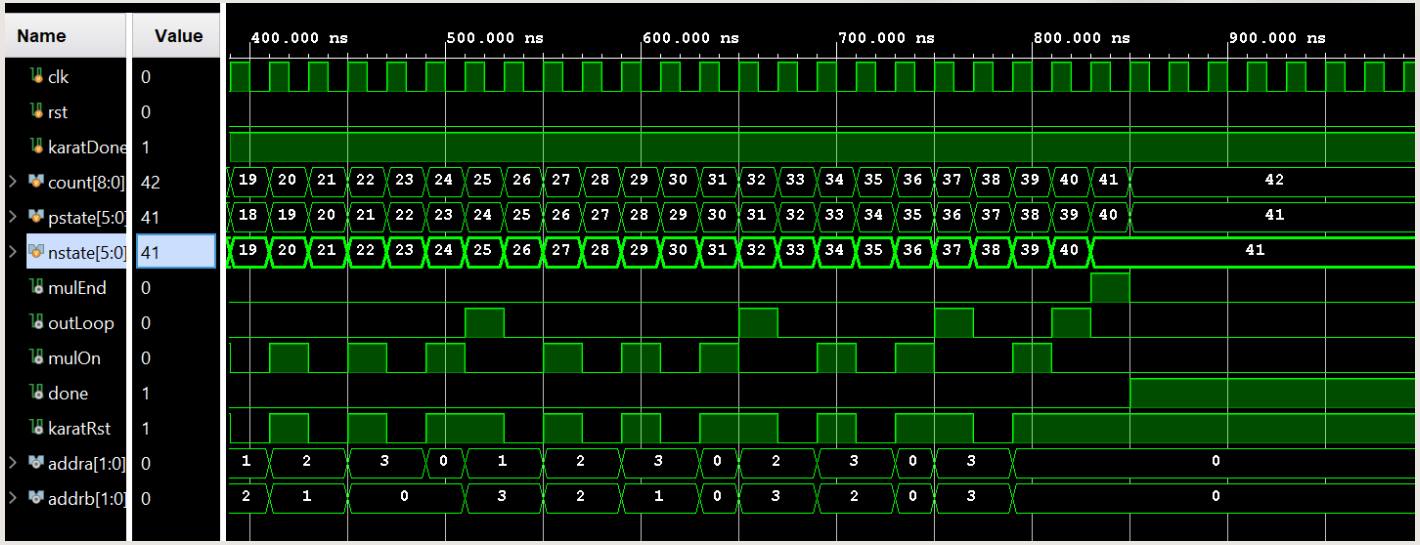
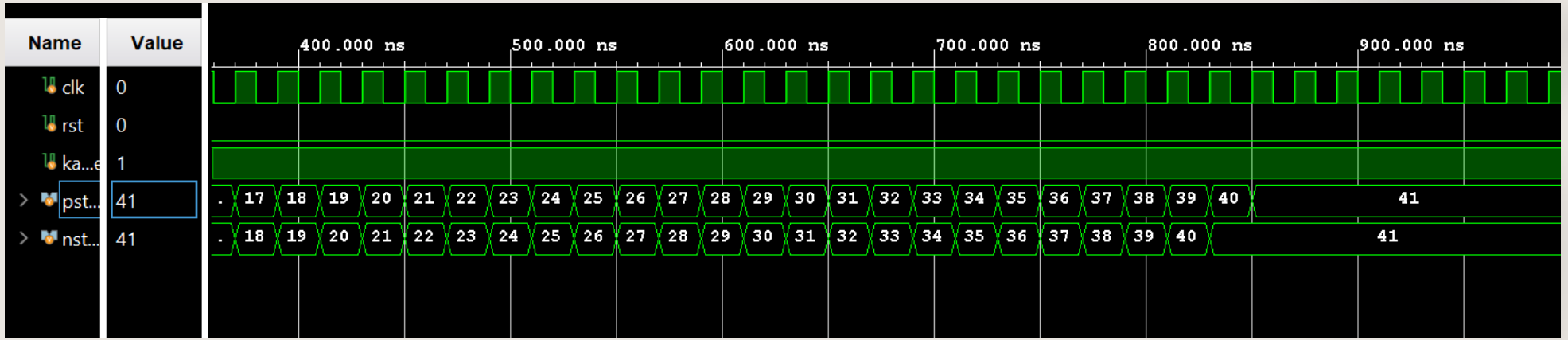


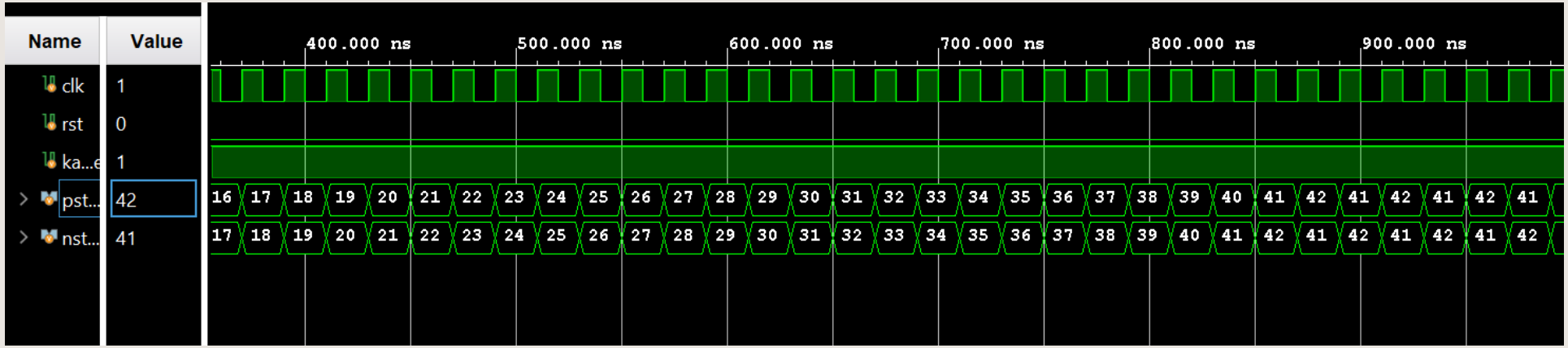
Fig: Simulation results of FSM infected with a Bit-Flip Trojan

TI: Trojan Insertion

Results:



Simulation results of an FSM without Infinite Loop Trojan.



Simulation results of FSM with Infinite Loop Trojan.

Conclusion:

- We introduce a powerful framework for improving hardware security by automating the identification and insertion of hardware Trojans with the help of advanced large language models.
- Utilizing Google's Gemini and OpenAI's GPT-4, our tools draw on comprehensive knowledge of hardware vulnerabilities to effectively analyze Verilog designs, identify weaknesses, and embed Trojans at optimal insertion points.
- We demonstrate the capabilities of these tools with two applications: Finite State Machines (FSM) and the Advanced Encryption Standard (AES).
- Through inserted trojan in a
  - FSM we illustrate the disruption of control flow in digital circuits
  - AES is particularly severe, as it is leading to incorrect encryption, and also bypassing encryption integrity checks.

Thank You