LASP: LLM Assisted Security Property Generation for SoC Verification

Avinash Ayalasomayajula, Rui Guo, Jingbo Zhou,
Sujan Kumar Saha, Farimah Farahmandi

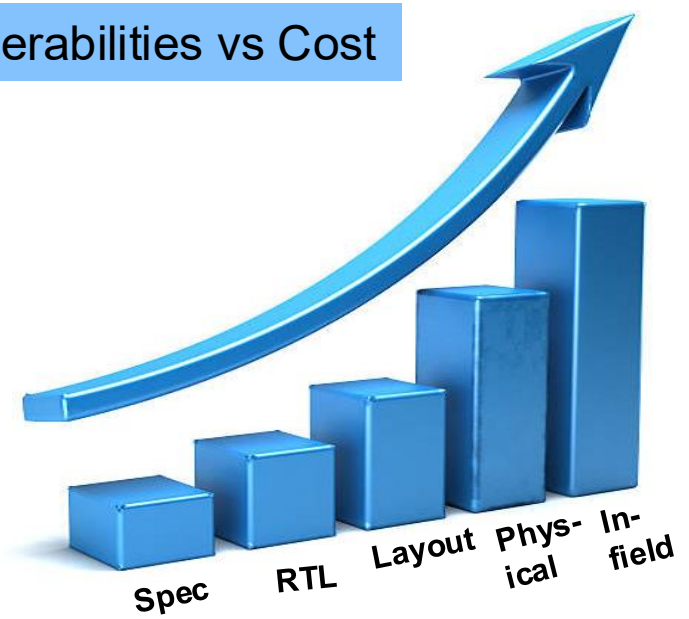
Electrical and Computer Engineering,
University of Florida

SoC Security Verification

Objective: To provide comprehensive SoC security at the pre-silicon stage.

- Identification of security vulnerabilities at the pre-silicon level, can reduce costs to design houses, compared to post-silicon fixes.
- They also provide the flexibility of introducing security counter measures during the design phase to ensure protection.
- This reduces the number of re-spins required for any SoC chip, reducing monetary and resource costs.

Fixing vulnerabilities vs Cost



Security Assets for an SoC

- SoCs store and propagate users' private and confidential data.

Asset: Any SoC resource of importance that needs to be protected.

Passwords


Medical Data

Financial Data


Media

Objective: Protect the security assets in the design from any threat.

Formal Property Verification (FPV)

Property: A statement that checks assumptions, conditions and the expected behavior of a hardware design

- Formal Property Verification (FPV) utilizes defined properties for security verification of SoCs.
- Properties are represented in the form of Assertion/Cover.
- The expected behavior formally defined as assertions are proved mathematically throughout the complete design space.

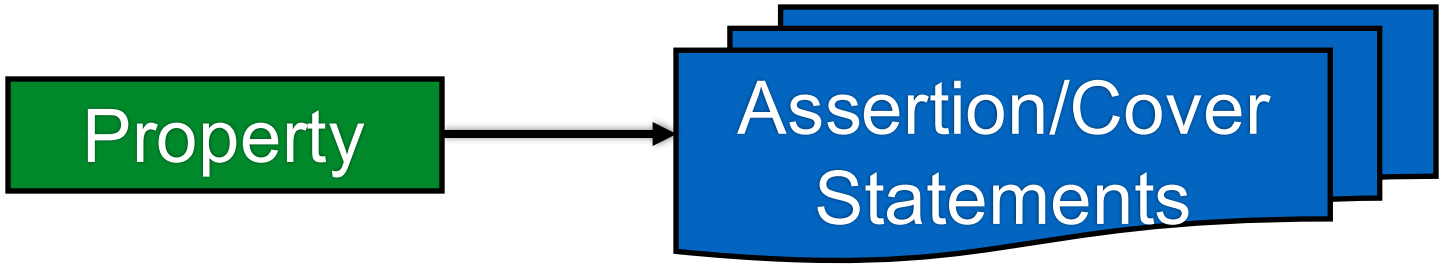


Fig: A single property can be translated into multiple assertion/cover statements.

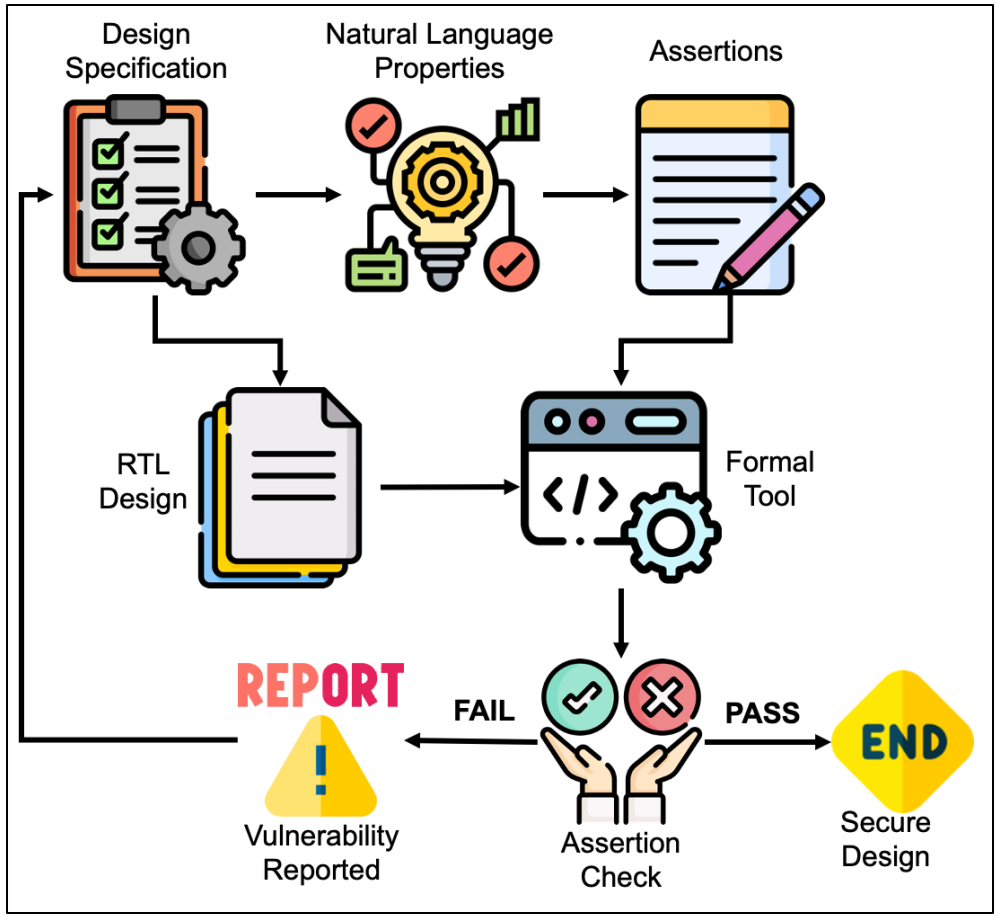


Fig: Formal Property Verification

Traditional Property Generation

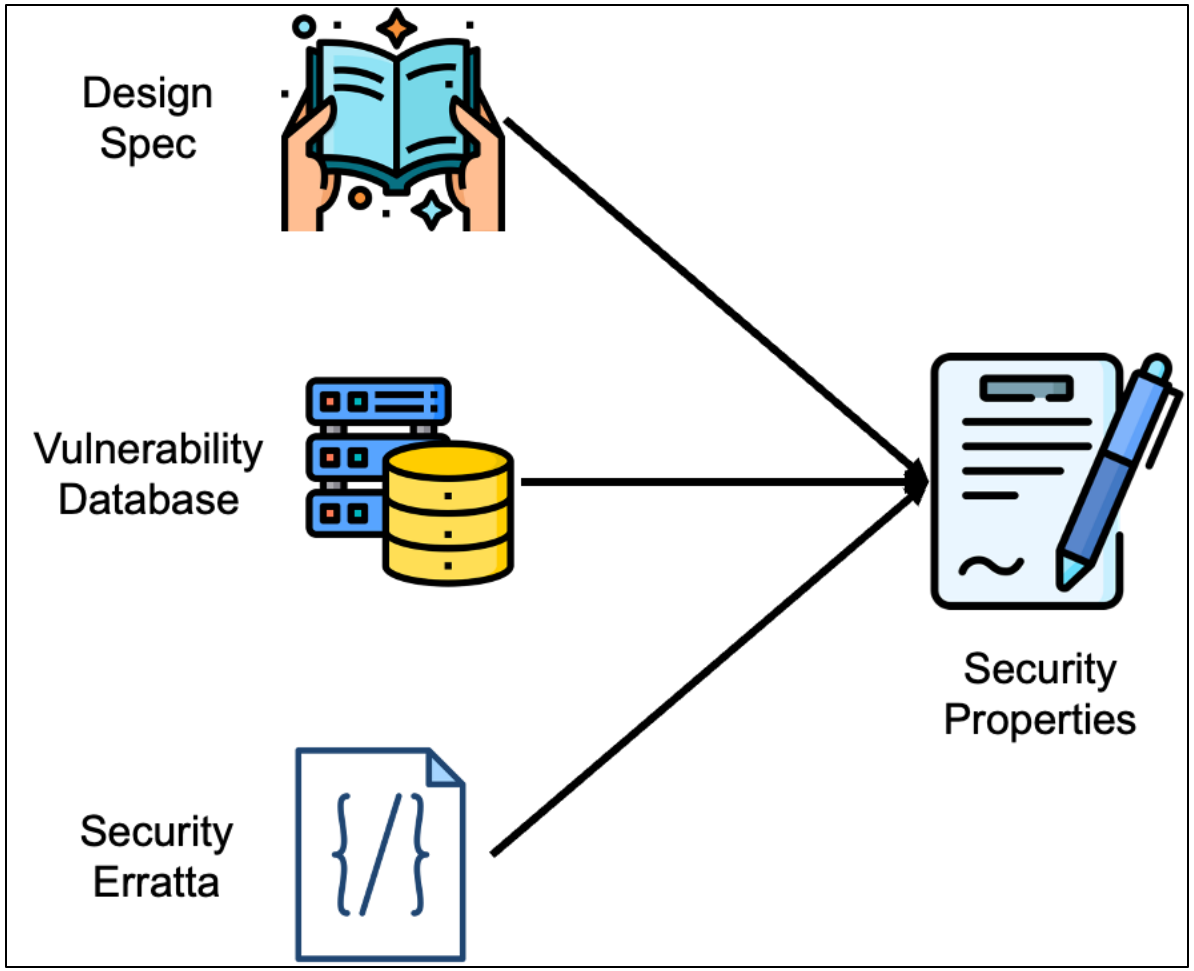


Fig: Traditional Property Generation Methods

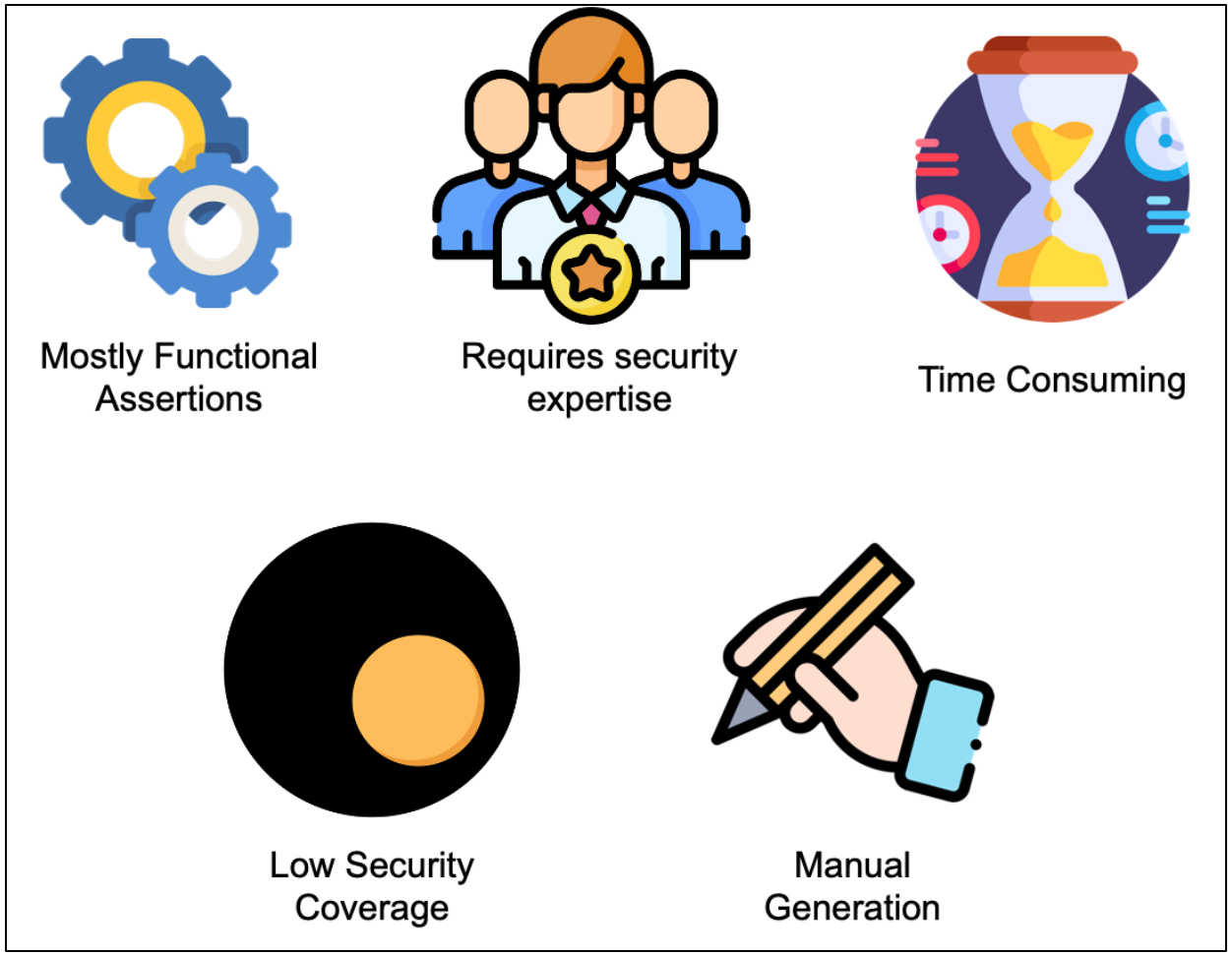


Fig: Limitations

Large Language Models For Security

- Large Language Models (LLMs) are computational models which excel at generating contextually relevant text by learning from extensive text data.
- Previous research has delved into the use of LLMs for hardware design and security:
  - **Architecture Specification Development**
  - **Threat Model Identification**
  - **Bug Detection and Security Analysis**
  - **Security Property Extraction**
- The natural language capabilities of LLMs provide extensive research opportunities towards natural language security property generation.
- However current approaches come with their limitations:
  - **Input Quality is Crucial**
  - **Struggles with Complex Inputs**
  - **Challenges in Generalization**
  - **Manual Integration is Labor-Intensive**

LASP Framework

- **LASP** is an LLM powered CAD tool for the generation of natural language security properties.
- Leverages security specific information such as:
  - ❑ Threat Model.
  - ❑ Security related design features.
- SVA convertible security properties for easy formal verification efforts.
- LASP consists of 3 major steps:
  - ➢ Security Asset Identification.
  - ➢ Design Feature Extraction.
  - ➢ Natural Language Security Property Generation.

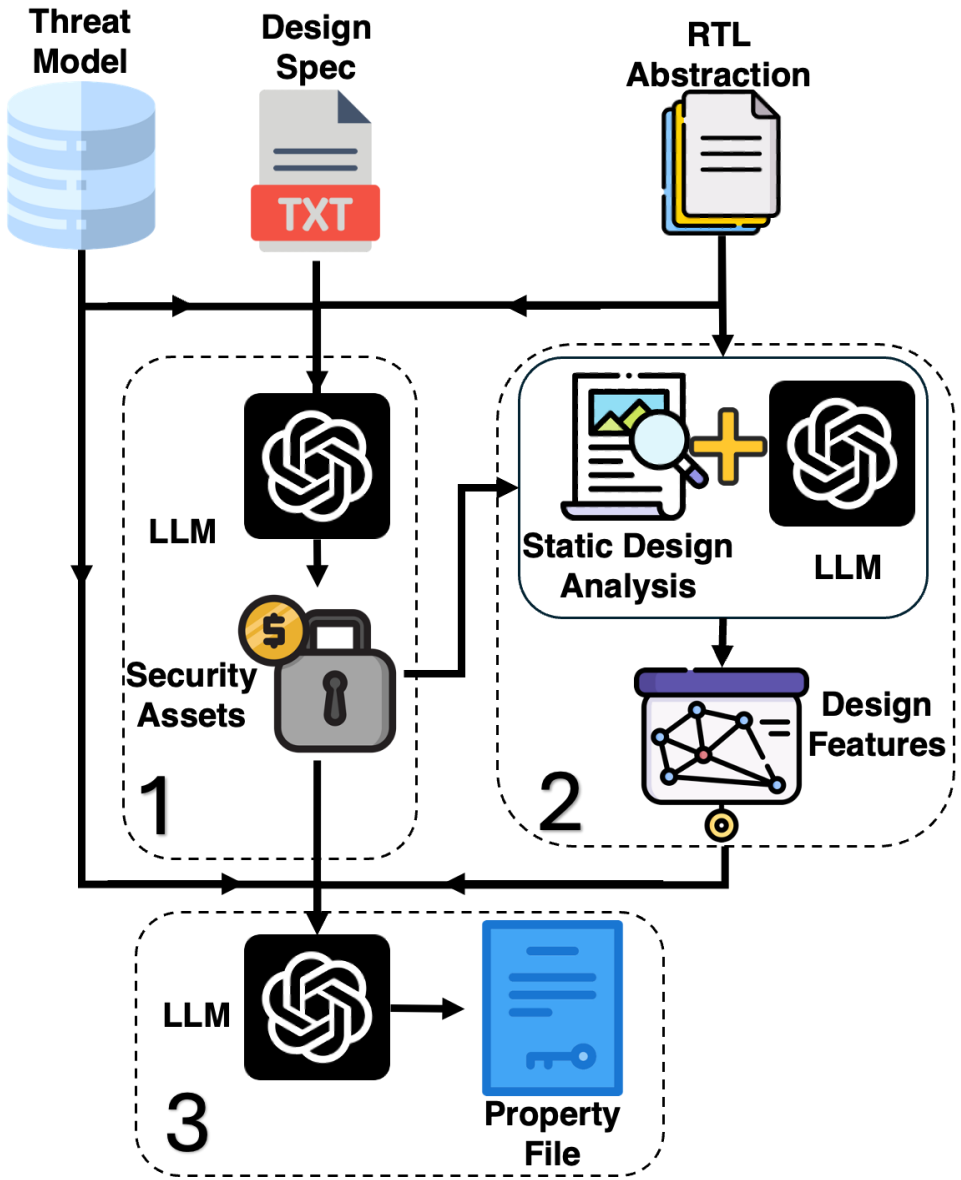
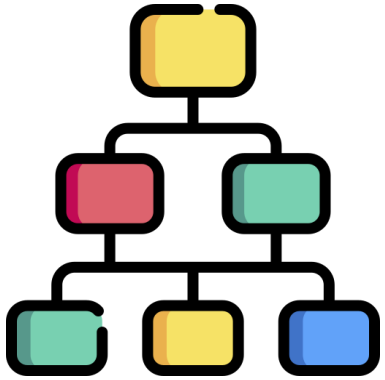


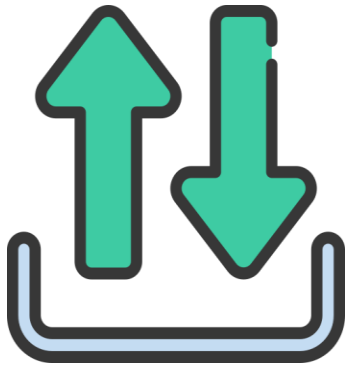
Fig: Overview of LASP Framework

Step 0: Pre-Processing Step

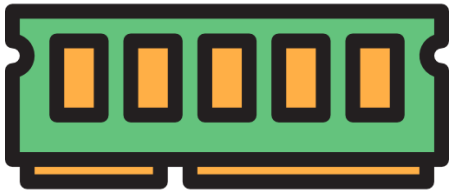
- Current LLMs struggle with analyzing large hardware designs due to limited input tokens.
- LASP addresses this by breaking down the design into Verilog sub-modules, creating concise summaries.



Module Hierarchy



Module Interfaces



Registers & Memory Declarations

Clock & Reset Information

Configuration Register Information

Step 0: Example RSA FSM Controller

```
always @(posedge clk) begin
    if (start == 1'b0) begin
        FSM <= IDLE;
        round_index <= 4'h0;
        finished <= 1'b0;
    end else begin
        case (FSM)
            IDLE:
                if (start == 1'b1)
            INIT:
                FSM <= LOAD1;
            LOAD1:
                FSM <= LOAD2;
            LOAD2:
                FSM <= MULT;
            MULT:
                FSM <= SQR;
            SQR:
                if (round_index == 4'b0010)
                    FSM <= RESULT;
                else begin
                    FSM <= MULT;
                    round_index <= round_index + 4'b1;
                end
            RESULT: begin
                FSM <= IDLE;
                finished <= 1'b1;
            end
        endcase
    end
end
```

Listing: RSA FSM Controller RTL Design

```
1. Module Hierarchy:
   Single   module: RSA_binary_v2.
2. Module Inputs and Outputs:
    Inputs: clk (clock), rst (reset), start.
    Output: finished (indicates completion of the process).
3. Registers and Memories:
    Registers:
    FSM (current state of the finite state machine).
    next_FSM (next state).
    round_index (tracks the rounds in the RSA process).
    next_round_index (next value of round_index).
4. FSM Information:
    States: IDLE, INIT, LOAD1, LOAD2, MULT, SQR, RESULT.
    Operation: Transitions through these states based on start
    signal and round_index, managing the RSA computation sequence
    .
5. Clock and Reset:
    Synchronous operation controlled by clk.
    Reset (rst) sets the FSM to IDLE state, ensuring a clean
    start.
```

Listing: RSA FSM Controller Abstraction

Step 1: Security Asset Identification

- **Threat Model-Driven Approach**:
  - The identification of security assets is tailored to specific threat models, requiring different asset protections.
- **Comprehensive Input Analysis**:
  - LASP analyzes RTL code abstractions, specification documents, and threat model definition to understand the security impact.
- **Strategic Asset Identification**:
  - LASP intelligently identifies key assets like registers, memory locations, and FSMs based on their relevance to the threat model.

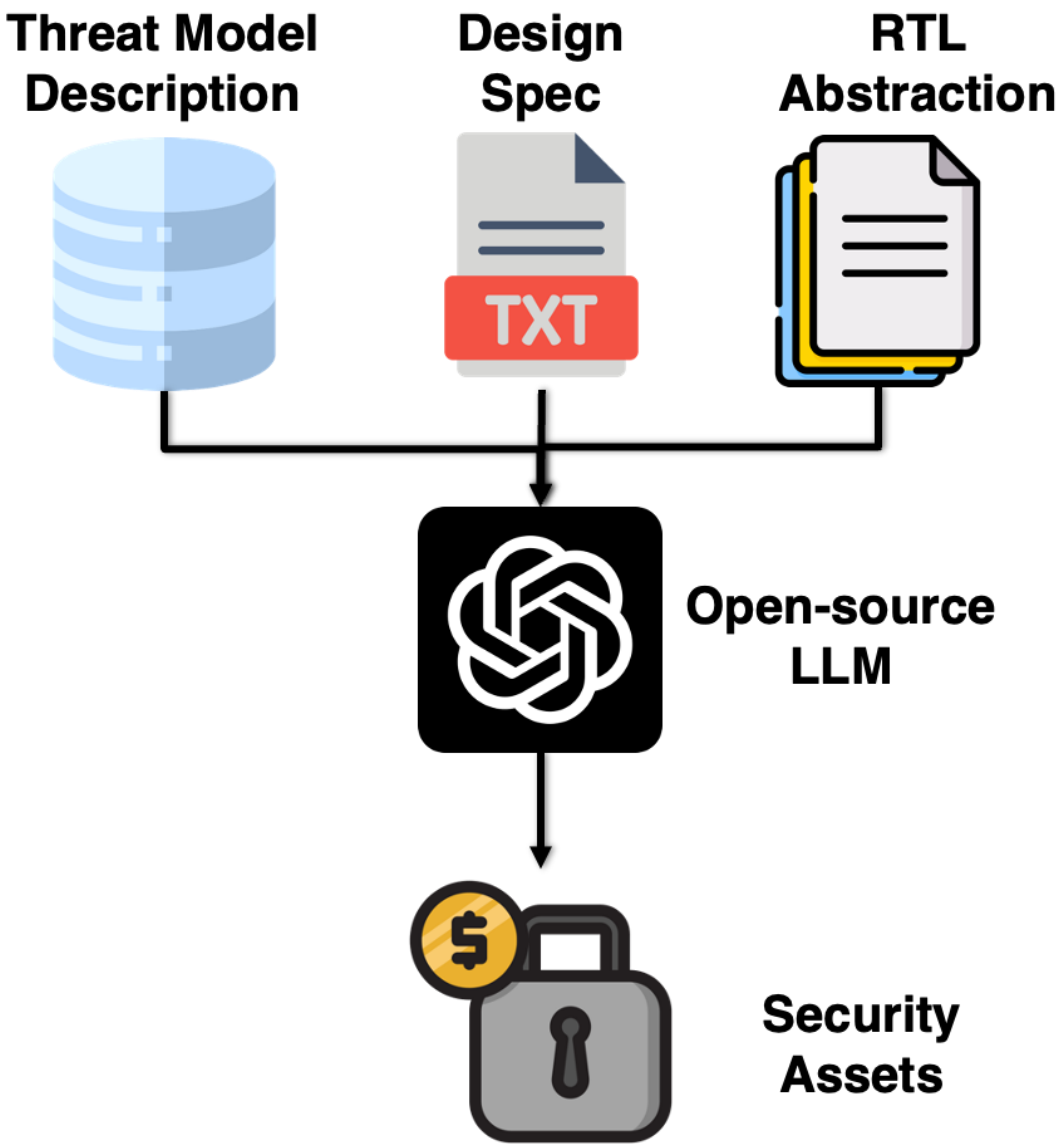


Fig: Security Asset Identification Step

Step 1: Example RSA FSM Controller

```
1. Module Hierarchy:
  Single   module: RSA_binary_v2.
2. Module Inputs and Outputs:
    Inputs: clk (clock), rst (reset), start.
    Output: finished (indicates completion of the process).
3. Registers and Memories:
    Registers:
    FSM (current state of the finite state machine).
    next_FSM (next state).
    round_index (tracks the rounds in the RSA process).
    next_round_index (next value of round_index).
4. FSM Information:
    States: IDLE, INIT, LOAD1, LOAD2, MULT, SQR, RESULT.
    Operation: Transitions through these states based on start
    signal and round_index, managing the RSA computation sequence
    .
5. Clock and Reset:
    Synchronous operation controlled by clk.
    Reset (rst) sets the FSM to IDLE state, ensuring a clean
    start.
```

```
Illegal States and Transitions:
This threat model concerns the potential for a system to enter
    invalid or unintended states, or to transition between states
     in an unauthorized or unexpected manner. Such behavior could
     disrupt normal operation, compromise data integrity, or
    expose vulnerabilities that could be exploited by attackers.
    This includes violations of expected protocols, unexpected
    state transitions in finite state machines, or control
    signals being activated outside of their intended sequence.
```

Listing: Illegal States and Transitions Threat Model Definition

Security Asset	Description
FSM & next_FSM	Controls RSA operation flow. Manipulation could bypass steps, alter sequences, or force errors, disrupting operations or extracting information.
round_index and next_round_index	Control rounds in modular exponentiation. Tampering could lead to incorrect results, weakening encryption and easing attack.
start	Controlling this input could prematurely trigger or halt RSA, causing denial of service or interrupting processes.

Table: Identified Security Assets

Step 2: Design Feature Extraction.

- LASP identifies and analyzes interactions between critical assets and design components to ensure comprehensive security.
- Combines static analysis with LLM interpretation **Static Analysis Phase**:
  - Converts RTL code into an Abstract Syntax Tree (AST) and builds a Control and Data Flow Graph (CDFG) to map data and control pathways.
- **LLM Complementation**:
  - LLMs add insights by analyzing code abstractions and documentation, capturing security-relevant behavior and features.

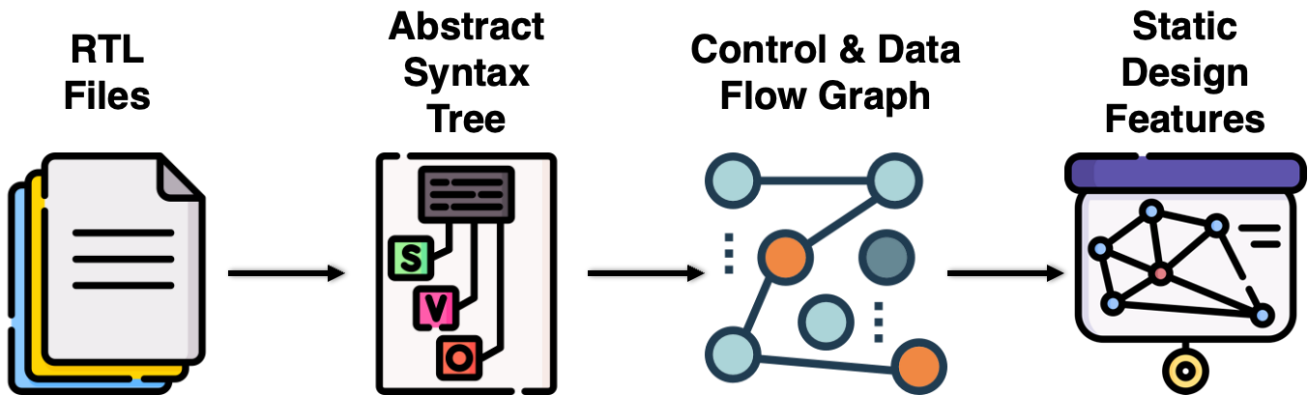


Fig: Static Design Feature Extraction Step

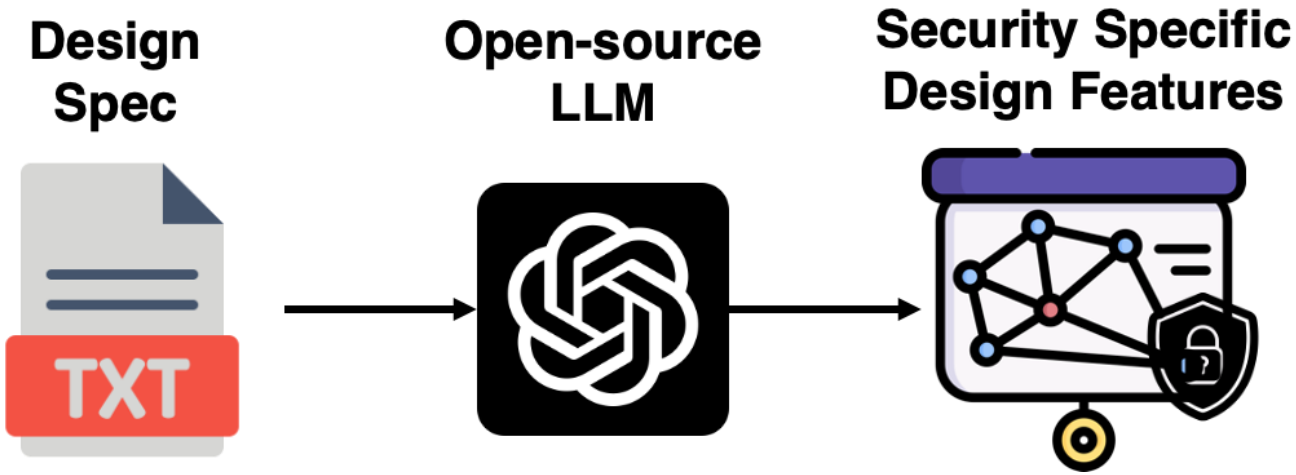


Fig: LLM based security specific design feature extraction step.

Step 2: Example RSA FSM Controller

```
1
2  always @(posedge clk) begin
3      if (start == 1'b0) begin
4          FSM <= IDLE;
5          round_index <= 4'h0;
6          finished <= 1'b0;
7      end else begin
8          case (FSM)
9              IDLE:
10                 if (start == 1'b1)
11             INIT:
12                 FSM <= LOAD1;
13             LOAD1:
14                 FSM <= LOAD2;
15             LOAD2:
16                 FSM <= MULT;
17             MULT:
18                 FSM <= SQR;
19             SQR:
20                 if (round_index == 4'b0010)
21                     FSM <= RESULT;
22                 else begin
23                     FSM <= MULT;
24                     round_index <= round_index + 4'b1;
25                 end
26             RESULT: begin
27                 FSM <= IDLE;
28                 finished <= 1'b1;
29             end
30         endcase
31     end
```

Listing: RSA FSM Controller RTL Design

Feature	Details
I/O	
clk	Input; Clock
start	Input; Initiates RSA
finished	Output (reg); Indicates completion
Control Signals	
start	Initiate Operation
FSM	FSM State Sequence Control
round_index	Loop Control Signal.
FSM States	
States	DLE, INIT, LOAD1, LOAD2, MULT, SQR, RESULT
Transitions	IDLE -> INIT; INIT -> LOAD1; LOAD1 -> LOAD2; LOAD2 -> MULT; MULT -> SQR; SQR (round_index==4'b0010) -> RESULT; SQR (round_index!=4'b0010) -> MULT, RESULT -> IDLE

Step 3: Security Property Generation

- LASP creates natural language security properties using identified assets and design features.
- **Refined Threat Model Integration**:
  - We incorporate a refined threat model with specific security goals (e.g., detecting unauthorized reads/writes, FSM manipulation).
- **Customization and Flexibility**:
  - Designers can provide additional guidelines for addressing specific vulnerabilities, protocols, and performance constraints.
- **Actionable and Human-Readable Outputs**:
  - The generated properties are clear and actionable, easily translatable into System Verilog Assertions (SVA) for formal verification.

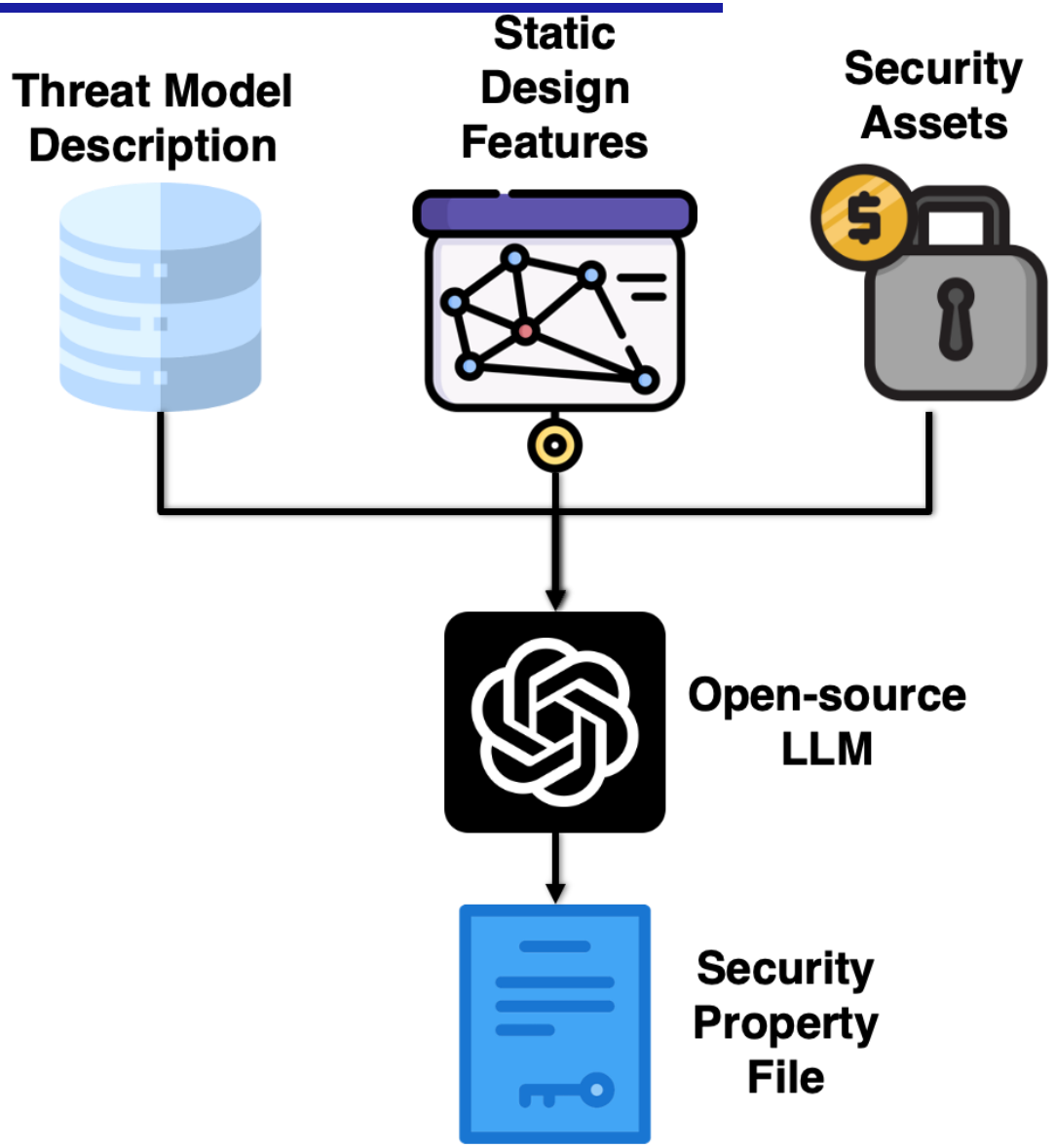


Fig: Natural Language Security Property Generation Step

Step 3: Example RSA FSM Controller

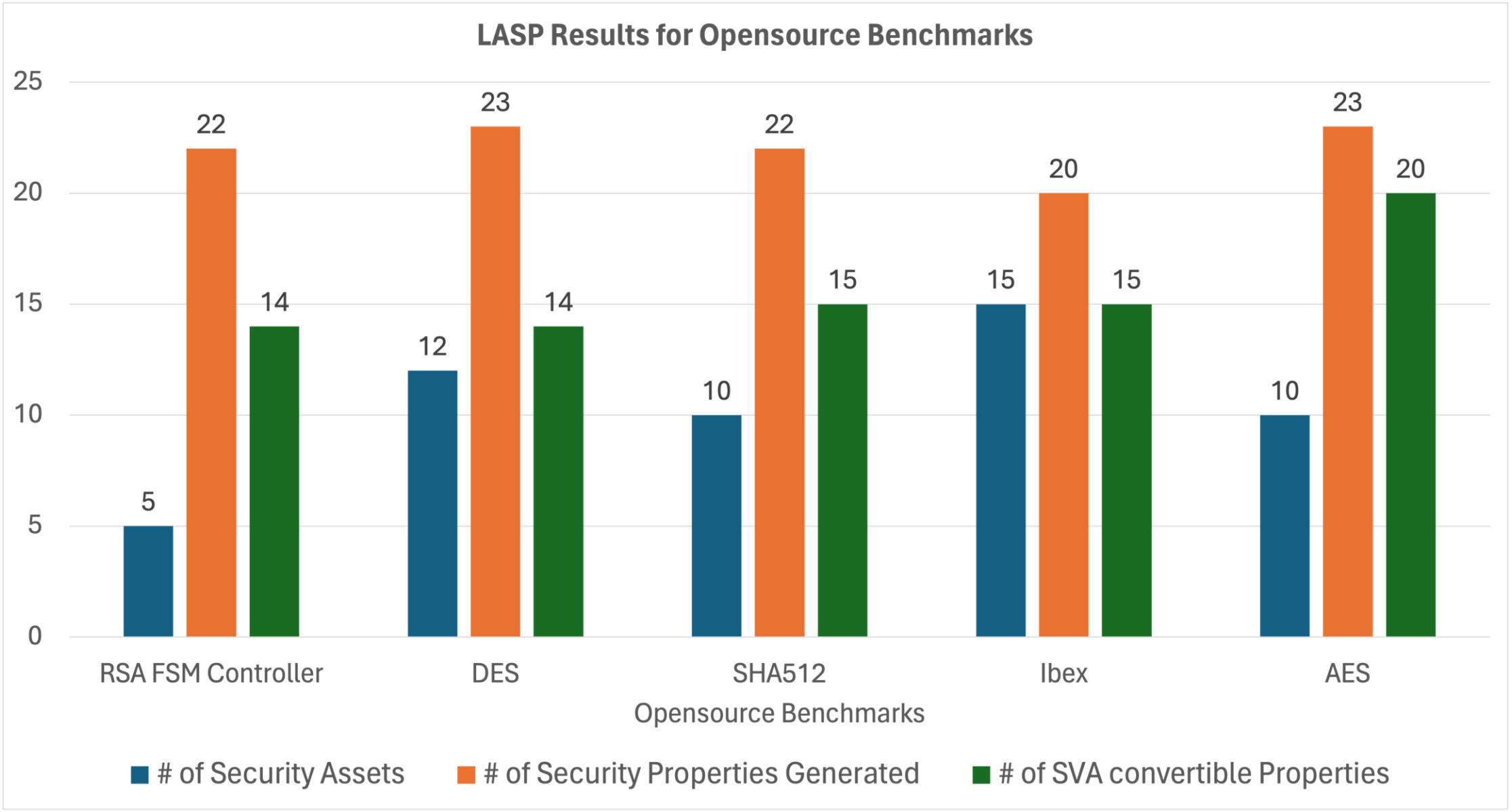
Index	Property	Threat Model Aspect
P1	FSM shall only enter valid states: IDLE (3'b001), INIT (3'b010), LOAD1 (3'b011), LOAD2 (3'b100), MULT (3'b101), SQR (3'b110), RE- SULT (3'b000). Any other state is illegal.	Illegal States.
P2	In IDLE (3'b001), FSM remains in IDLE until 'start' signal transi- tions from low to high.	Unauthorized Access.
P3	Transitions to LOAD1, LOAD2, MULT, SQR, or RESULT states shall occur only from their immediate preceding state as defined. Any other transition is unauthorized.	Illegal States.
P4	FSM remains in SQR state for the number of rounds specified by the RSA algorithm, controlled by 'round_index' register.	Execution Integrity.
P5	FSM transitions from RESULT to IDLE only after 'finished' signal is asserted high, indicating RSA operation completion.	Execution Integrity.
P6	FSM's state register shall never hold a value outside the defined states (3'b000 to 3'b110). Any such value is illegal.	Illegal States.
P7	Complete RSA operation state sequence: IDLE (3'b001) → INIT (3'b010) → LOAD1 (3'b011) → LOAD2 (3'b100) → MULT (3'b101) → SQR (3'b110) → RE- SULT (3'b000) → IDLE (3'b001).	Execution Integrity.

Experimental Results

- We evaluate LASP across various benchmarks.
- For each benchmark we manually evaluate the number of properties translatable to SVAs.
- On an average **80%** of the properties were easily SVA convertible, the remaining can be utilized in non-formal security checks such as side channel assessment etc.

AES Properties	Equivalent SVA
Property P1: The key input signal shall only be modified when the key_mem_ctrl_reg in FSM is in the CTRL_IDLE state and init is asserted	**property p_key_write_auth;** @(posedge clk) disable iff (!reset_n) $rose(key_we) |-> (key_mem_ctrl_reg == CTRL_IDLE) && init; endproperty
Property:P2: The signals round_key, enc_sboxw, and keymem_sboxw shall be scrambled or masked when not in use	No equivalent SVA. However, this property can be utilized for Side Channel Assessment Analysis to ensure security.

Experimental Results



Conclusion

- Introduced a novel approach using LLMs to automate the generation of security properties for SoC verification.
- Streamlined security property generation, reducing manual effort and improving accuracy in detecting vulnerabilities.
- Successfully applied to various threat models and hardware designs, demonstrating versatility and scalability.
- LASP-generated properties are easily convertible into SVA for Formal Property Verification (FPV), fault injection assessments, and other security verification methods.

THANK YOU!

ANY QUESTIONS?