

LAsset: An LLM-assisted Security Asset Identification Framework for System-on-Chip (SoC) Verification

Md Ajoad Hasan, Dipayan Saha, Khan Thamid Hasan, Nashmin Alam, Azim Uddin, Sujan Kumar Saha, Mark Tehranipoor, Farimah Farahmandi
Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA
 {md.hasan, dsaha, khamthamidhasan, nashminalam, azim.uddin, sujansaha}@ufl.edu, {tehranipoor, farimah}@ece.ufl.edu

Abstract—The growing complexity of modern system-on-chip (SoC) and IP designs is making security assurance difficult day by day. One of the fundamental steps in the pre-silicon security verification of a hardware design is the identification of security assets, as it substantially influences downstream security verification tasks, such as threat modeling, security property generation, and vulnerability detection. Traditionally, assets are determined manually by security experts, requiring significant time and expertise. To address this challenge, we present *LAsset*, a novel automated framework that leverages large language models (LLMs) to identify security assets from both hardware design specifications and register-transfer level (RTL) descriptions. The framework performs structural and semantic analysis to identify intra-module primary and secondary assets and derives inter-module relationships to systematically characterize security dependencies at the design level. Experimental results show that the proposed framework achieves high classification accuracy, reaching up to 90% recall rate in SoC design, and 93% recall rate in IP designs. This automation in asset identification significantly reduces manual overhead and supports a scalable path forward for secure hardware development.

Index Terms—Security Asset Identification, Large Language Model (LLM), Hardware Security, Technical Specification, RTL.

I. INTRODUCTION

Modern system-on-chip (SoC) designs have grown into highly complex systems that integrate processors, memory, bus, and a wide range of peripherals onto a single chip [1], [2]. While this high degree of integration enhances performance and cost-effectiveness, it simultaneously introduces new security challenges by significantly expanding the attack surface [3], [4]. At the same time, increasing market pressure often forces shorter development cycles, leaving limited room for comprehensive pre-silicon security verification. As a result, many vulnerabilities remain undetected until the post-silicon phase [5], [6], where mitigation is substantially more resource-intensive [7]. To ensure robust protection, it is therefore critical to address security issues as early as possible in the design cycle [8]. Early identification of security-critical components not only reduces the risk of costly fixes later in the flow but also builds confidence in the overall design trustworthiness.

In reliable and secure hardware systems design, accurately identifying security assets early on provides the foundation for subsequent tasks, such as threat modeling, test plan generation, security verification, and countermeasure development, as shown in Figure 1 [9], [10], [11], [12]. As underscored by *Security Annotation for Electronic Design Integration (SA-EDI)* standard [13], asset identification is not merely the initial but

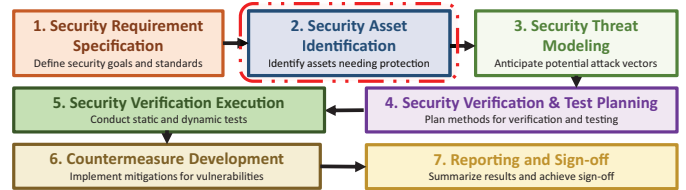


Fig. 1: Security Verification Flow in hardware design

arguably the most important step toward sustainable security assurances in hardware systems. Despite its widely recognized significance, asset identification has traditionally remained a manual process based on industry expertise [14], [15], making it not only inefficient and error-prone but also difficult to scale for large SoC designs. Only a few recent works have explored asset identification, and those that do often consider it in a limited context. For instance, the SAIF tool [16] attempts to identify assets in hardware designs by analyzing vulnerabilities and threat models, while the LASHED flow [17] leverages common weakness enumeration (CWE) vulnerabilities for the same purpose. However, both approaches identify assets only after analyzing design vulnerabilities, whereas asset identification should be the primary step in a security verification flow, as illustrated in Figure 1. Similarly, the method in [18] relies heavily on RTL signal and register naming conventions, using pattern matching against known security-critical identifiers; as a result, it fails to generalize to designs that do not follow such conventions and offers no reasoning for why the detected elements should be considered assets.

Motivated by the challenges in today’s asset identification approaches, we outline the following research questions (RQs):

RQ 1: How can we determine which elements are considered security-relevant assets within a given hardware design architecture?

RQ 2: Can we develop an end-to-end automated methodology for reliably identifying security assets across diverse SoC and IP designs?

RQ 3: How can we validate that an identified asset is really an asset?

After investigating the answers to the above questions, we propose *LAsset*, the first-ever LLM-assisted automated framework for comprehensive security asset identification in SoC and IP designs, motivated by the success of LLM-assisted frameworks in hardware security verification [19], [20], [21], [22], [23], [24]. *LAsset* leverages the SA-EDI standard and

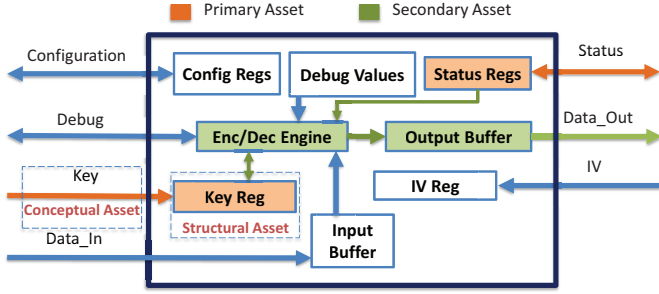


Fig. 2: Overview of Asset Types in an AES Encryption Design

IEEE P3164 guidelines for preliminary identification of assets using LLM. It further validates and refines the asset list by mapping it against the CWE vulnerability database, thereby ensuring its trustworthiness. In summary, the key contributions of our work are as follows:

- **Novelty:** In this paper, we present *LAsset*, the first framework to leverage LLMs for the automated identification of security assets from SoC and IP designs.
- **Robustness and Versatility:** *LAsset* operates with either combined specification and RTL inputs or RTL alone, supports Verilog, SystemVerilog, and VHDL, and applies to both IP and SoC designs, ensuring broad applicability.
- **Comprehensiveness:** Beyond asset identification, *LAsset* associates each asset with threat modeling details and security justifications, providing actionable context for downstream verification tasks.
- **Trustworthiness:** A refinement stage validates each asset by cross-referencing the CWE database and applying self-consistency checks, yielding a high-fidelity asset list with up to 93% recall on real designs.

The rest of this paper is organized as follows. Section II provides a brief account of the background and rationale. Section III describes the proposed methodology. The experimental results and comparative analysis are presented in Section IV before concluding the paper in Section V.

II. BACKGROUND

A. Security Asset

By definition, a security asset in SoCs is any hardware component or data element whose protection is essential to preserve the system’s confidentiality, integrity, or availability (CIA). Security assets can be classified based on two criteria: (i) abstraction and (ii) dependency. According to IEEE P3164 [25], Security assets can be classified as conceptual and structural depending on the abstraction level of the design.

- **Conceptual Assets:** Conceptual Assets are high-level information elements tied to the system’s use-case flows. For example, as shown in Figure 2, in an AES encryption engine, the encryption key is a conceptual asset because its confidentiality must be preserved regardless of where it resides in the design.
- **Structural Assets:** Structural Assets are the hardware elements that physically store or carry conceptual assets. These include registers, buffers, latches, or gates involved in handling or storage of conceptual assets. In the same AES engine, the

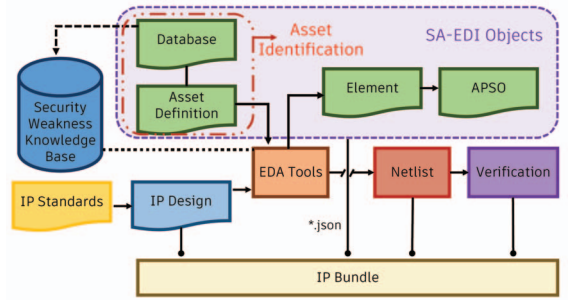


Fig. 3: SA-EDI IP bundle [13]

Key Register is a structural asset because it directly stores the encryption key value.

In addition, security assets can be categorized into Primary and Secondary Assets based on their dependency role in potential security breaches [16].

- **Primary Assets:** Primary Assets are components that serve as the direct target of an attack. Continuing the AES example, the encryption key itself is the primary asset since attackers seek to compromise it directly.
- **Secondary Assets:** Secondary Assets are components that interact with or facilitate the exposure of primary assets. While not directly critical, their compromise can weaken the overall security of the SoC by retrieving the associated primary asset. For example, system buses, peripheral ports, and internal signals/registers that carry the data of the primary asset, either fully or partially. In AES, elements such as the Encryption/Decryption Engine and the Output Buffer qualify as secondary assets because, though they are not the final target, their compromise can indirectly expose the encryption key.

B. Security Annotation for Electronic Design Integration (SA-EDI) standard

The SA-EDI standard provides specification guidelines for documenting the security concerns of hardware IPs as they are integrated into SoCs. As illustrated in Figure 3, security-related information—including asset definitions, security weakness databases, port- and parameter-related elements, and Attack Points Security Objectives (APSO)—is captured alongside the IP design and represented in a SA-EDI object in JSON format. The standard emphasizes that the first step is to identify conceptual assets in the design that require protection under security objectives such as confidentiality, integrity, and availability (CIA), along with their associated structural elements (e.g., ports, parameters) that could compromise these objectives. To support this process, the standard refers to the IEEE P3164 white paper [25], which introduces the *Conceptual and Structural Analysis (CSA)* methodology. CSA begins by identifying conceptual assets tied to the CIA triad and then mapping them to their corresponding RTL representations, i.e., structural assets that are passed to the next Asset Definition objects in SA-EDI. This methodology highlights the distinction between conceptual assets—the ‘what’ that requires protection, and structural assets—the ‘where’ in the design that embodies that asset—ensuring that IP developers document security-critical assets comprehensively before integrating IPs into larger

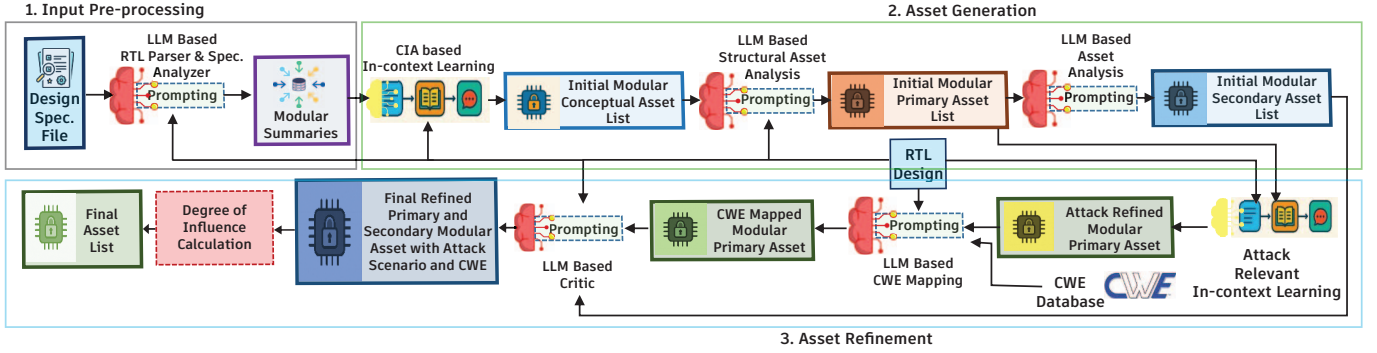


Fig. 4: Overview of the proposed *LAsset* framework for security asset identification

SoC designs. Accordingly, to address our **RQ1** in Section I, we build on the insights provided in the SA-EDI guidelines.

III. LASSET FRAMEWORK

Identifying security assets as posed in **RQ1-RQ3**, mentioned in Section I, requires a solution with several key qualities. Following the SA-EDI principle of first defining conceptual assets based on CIA objectives and then mapping them to structural design elements, it must understand both natural-language specification and RTL to map conceptual assets to their structural counterparts; reason over design semantics, hierarchy, and inter-module dependencies to generalize across diverse SoC and IP designs; and provide verifiable, standards-aligned justifications to ensure the fidelity of identified assets. Such a task requires semantic comprehension, contextual reasoning, modular task decomposition, and explainable decision-making capabilities that traditional rule-based approaches lack. LLMs inherently offer these properties: they can jointly interpret specification and RTL, infer security relevance from context rather than naming heuristics, be organized into specialized agents for sub-tasks like specification analysis, RTL analysis, asset mapping, and CWE-based rationalization, and generate human-readable rationales with self-consistency checks. Leveraging these strengths, we design *LAsset* as an agentic framework driven by an LLM capable of performing automated, scalable, and reliable asset identification.

As shown in Figure 4, *LAsset* operates through three sequential agents: (i) *Input Pre-processing*, which parses the available specification and RTL resources and aligns them under the SA-EDI-based conceptual–structural asset model; (ii) *Asset Generation*, where specialized LLMs analyze the pre-processed inputs to identify candidate conceptual assets and map them to their structural counterparts; and (iii) *Asset Refinement*, which validates and rationalizes each candidate through cross-referencing with the CWE database and self-consistency checks, producing a final, standards-aligned asset list. This modular agentic design enables *LAsset* to perform automated, scalable, and verifiable security asset identification across diverse SoC and IP designs.

A. Input Pre-processing

Between the two inputs of *LAsset* framework, the RTL repository provides the internal functionalities as well as in-

formation flow tracking, whereas the corresponding Spec. primarily outlines the module’s integration within the broader SoC architecture, providing additional design insights and interface details in natural language. Therefore, we pre-process the inputs in terms of the security concerns of the design so that LLMs can better understand the true security context rather than hallucinating due to token limitations and constrained long-context retention. The *Input Pre-processing* agent performs the following three major tasks, as shown in Algorithm 1 (line 3-4):

1) *Design Modules Extraction*: In this task, LLMs’ reasoning is employed to (i) prune non-relevant modules and (ii) select security-critical modules among the extracted module names from the RTL repository. For example, within the NE-ORV32 SoC, the *neorv32_application_image* IP module is not considered security-critical and was therefore excluded during this task, whereas the security-sensitive *neorv32_cpu_pmp* IP module, among others, was retained. This task is important, for it makes the flow much more resource-efficient (both cost and time) by reducing the security non-relevant design overhead in the beginning.

2) *Modular Spec. Summarization*: This task is built on a Retrieval-Augmented Generation (RAG) in-context learning-based model, where the SoC-level design Specs. is used as the knowledge source. For each security-relevant module, a user-guided query augments the prompt to produce a *Technical Summary* via LLMs, which contains all the security-relevant information both within the module and across modules in the broader SoC context. Overall, this *summary* acts as a concise Spec. input for the *Asset Generation* agent for the respective module.

3) *Modular RTL Parsing*: To extract the design elements, we implement two LLM-based parsers: one for I/O ports and another for internal signals/registers, along with their types and functions at the module level. This is done to ensure that LLMs do not overlook any of the parsed design elements for asset decision, thereby reducing the chance of false negatives.

B. Asset Generation

Using the *Technical Summary* and RTL as input, we employ the few-shot in-context learning paradigm for asset generation. We define the hardware security context, supplemented by the Q/A related to CIA security objectives. To further illustrate the task, we provide several case studies featuring widely recognized hardware IP blocks, e.g., AES, GPIO, Gaussian Noise

Generator, each annotated with asset listings and corresponding justifications explaining why some design elements are security assets and others are not. Through this step-by-step approach, the LLM is systematically guided to learn and identify security assets.

After this in-context learning, through a stepwise reasoning and appropriate constraints approach, we identify the conceptual assets at the module level. These assets are then systematically mapped to their corresponding structural RTL references, derived from the parsed design elements- these are the *primary assets* at the module level. In addition, for each *primary asset*, we find the internal signals/registers that influence/violate its security objective(s)- these are termed as the *secondary assets* at the module level. *Asset Generation* steps are shown in Algorithm 1 (line 5-6).

C. Asset Refinement

When applied to security asset identification, LLMs often lean toward listing a broad set of possible security and non-security assets. This tendency, while helpful in maximizing coverage, can also introduce false positives by including design elements that are hardly security-relevant. For this reason as well as to answer our **RQ3** in section I, a rigorous three-stage back-to-back refinement agent is designed to filter out the false positives and validate the remaining results, ensuring that the final asset list at the module level is accurately substantiated. The refinement agent comprises the following stages, also shown in Algorithm 1 (line 7-17):

1) *Attack Scenario Analysis*: We evaluate each candidate primary asset for susceptibility to seven hardware attack classes: side-channel, fault injection, secure-to-nonsecure information leakage, unauthorized access, privilege escalation, hardware Trojan, and denial-of-service. Using in-context training, LLMs generate high-level abstractions of plausible attack scenarios for each asset; assets without such scenarios are excluded. This analysis justifies asset inclusion to verification experts and, through the zero-shot reasoning capabilities of LLMs, extends beyond modular boundaries to capture system-wide security implications.

2) *CWE Mapping*: To provide additional context for the generated assets as well as refine the assets further, we analyze whether and to what extent the CWEs from the MITRE database can be mapped to each primary asset [26]. Given the substantial size of the database, the analysis is structured to examine CWEs under each security objective one at a time, ensuring that no CWE is overlooked. Besides, we focus on only the security asset-relevant information in the database to prevent LLMs from being overwhelmed by extraneous information during the CWE mapping process. The rigorous mapping ensures that only the absolutely critical CWE(s), if any, are mapped to each asset. Finally, Assets with no mappable CWE are removed.

3) *Self-critique Revision*: The asset annotations generated at the module level so far are challenged by a self-critique prompt and revised further. Using the input resources again, we re-check asset relevance, RTL references, security-objective cor-

Algorithm 1 *LAsset* Framework

```

Require: Design Spec  $S$ ; RTL repo  $R = \{m_1, m_2, m_3, \dots\}$ ; DBASEcwe;
Require: Function Library: SpecRAG(Spec), SecAsset(Asset, Signals,
    RTL), UnrollAsset(Asset_List, Path), ICLasset, ICLattack;
Require: Prompt Library: LLMparse, LLMAsset, LLMattack, LLMcwe, LLMref.
Ensure: Group of modular asset annotations  $Asset_g = \{Ref\_Asset_m \mid m \in M\}$ .
Ensure: List of DoI per secondary asset  $DOI = \{DoIsec\_asset_h \mid h \in H\}$ 
1:  $M \leftarrow MOD\_LISTING(R)$ 
   Step: Modular Assets Identification
2: for each  $m \in M$  do
   /* Input pre-processing */
3:    $MOD\_SPEC S_m \leftarrow SPECRAG(S)$ 
4:    $MOD\_RTLPARSE \{Prim_m, Sec_m\} \leftarrow LLMPARSE(R(m))$ 
   /* Assets generation */
5:    $Asset_m \leftarrow LLMASSSET(S_m, Prim_m, R(m), ICLASSET)$ 
6:    $Exp\_Asset_m \leftarrow SECASSET(Asset_m, Sec_m, R(m))$ 
   /* Assets refinement */
7:    $Ref\_Asset-1_m \leftarrow LLMATTACK(Exp\_Asset_m, ICLATTACK)$ 
8:    $Ref\_Asset-2_m \leftarrow LLMCWE(Ref\_Asset-1_m, DBASEcwe)$ 
9:    $Ref\_Asset_m \leftarrow LLMREF(Ref\_Asset-2_m)$ 
10: end for
   Step: DoI calculation
11:  $H \leftarrow HIERARCH\_PATH(R)$ 
12: for path  $h \in H$  do
   /* Enumerate linked modules' assets */
13:    $\mathcal{A}_{h,g} \leftarrow UNROLLASSETS(A_g, h)$ 
14:   let  $\mathcal{A}_{h,g} \triangleq \{Amh_1, Amh_2, Amh_3, \dots\}$ 
15:    $DoIsec\_asset_h \leftarrow DOI(\mathcal{A}_{h,g})$ 
16:   let  $DoIsec\_asset_h \triangleq \{DoIsec\_asset_{h,1}, DoIsec\_asset_{h,2}, \dots\}$ 
17: end for
18: return  $Asset_g, DOI$ 

```

rectness, attack scenario coherence, and CWE appropriateness, thereby improving both accuracy and reliability.

4) *Degree of Influence (DoI) Calculation for Secondary Assets*: After enumerating assets at the module level, we assess how these assets influence one another within the design. As modular assets interact across boundaries, their interdependencies determine how local vulnerabilities may escalate into broader system risks. RTL analysis is used to extract hierarchical linkages among security-critical modules. Along each path, the most tamper-prone asset is designated as the *primary* asset, while the remaining assets are treated as *secondary*. Bit-level RTL connectivity is then analyzed to quantify the relationship between primary and secondary assets, and the influence is backtracked multiplicatively along each path to derive the final *DoI* metric for each secondary asset.

$$DoI = \frac{\# \text{ of bits of the secondary asset connected to the primary asset}}{\text{Total \# of bits of the primary asset}} \times 100\% \quad (1)$$

We compute DoI as an empirical metric to quantify the hardness of the identified secondary assets, which can also guide subsequent research within this scope.

IV. EXPERIMENTAL RESULTS AND EVALUATIONS

We present case studies on the NEORV32 RISC-V SoC and several hardware IP blocks, under two configurations, to demonstrate the efficacy and generality of the *LAsset* workflow in identifying security assets across diverse architectures. Results are available online.¹

LAsset employs OpenAI's GPT-5 model for in-context learning-based asset identification. In the *Modular Spec. Generation* task, retrieval is performed with 1,000-character chunks and

¹<https://github.com/Ajoad/LAsset-Security-Assets>

```

{ "Primary Asset": "CSR Read Data (Control/Status)",
  "Primary Asset Structural": "csr_rdata (neorv32_cpu)",
  "Security Objective": "Integrity",
  "Assets": [{
    "Secondary Asset": "CSR Write Enable",
    "Secondary Asset RTL": "csr_we_i",
    "Sub-secondary Assets": "csr_frm, csr_fflags, fflags",
    "Asset Linkage Path":
"csr_we_i.csr_o(fpu_csr_we).csr_rdata(xcsr_alu)",
    "Degree of Influence": "25%",
    "VHDL Entity": "neorv32_cpu_cp_fpu",
    "Functionality": "Input control that gates write access to FPU
CSRs (frm/fflags/fcsr); when asserted allows CSR updates from
csr_wdata_i to internal CSR registers.",
    "Attack Analysis": [{
      "Attack Vector 1": {
        "Attack": "Fault injection attacks",
        "Attack Scenario": "Inducing transient faults by injecting
voltage or clock glitches could result in unauthorized writes to the
CSR registers, such as 'csr_frm' and 'csr_fflags', altering the
rounding modes or exception flags. The RTL snippet 'if (csr_we_i =
'1') then' in the 'csr_write' process is critical, as it determines
whether the CSR registers are updated.}}],...],
    "System-level Impact": "The integrity of the 'CSR Write Enable'
is crucial for ensuring that only authorized writes are performed on
the FPU control registers.",
    "CWEs": ["CWE-1262", "CWE-1247"],
    "CWEs Reasoning": {
      "CWE-1262": "If the CSR/register interface is not privilege-
checked, untrusted software can execute CSR write instructions that
cause csr_we_i to assert and push attacker-controlled data into
csr_frm/csr_fflags.",... }}],...}]

```

Fig. 5: A snippet of *LAsset* output for NEORV32 processor (*Spec. + RTL* approach)

a 200-character overlap to preserve context. Embeddings are generated using OpenAI’s *text-embedding-ada-002* model, and similarity search is conducted with FAISS to retrieve the top 20 relevant chunks per query.

A. SoC: NEORV32 RISC-V processor

For the SoC case study, we select the NEORV32 RISC-V processor for evaluating the *LAsset* framework.

After a full *LAsset* run, assets are identified across NEORV IPs, and their interconnections yield the *Degree of Influence* of secondary assets on their primary counterparts. For example, the primary asset *csr_rdata* in *neorv32_cpu* is linked to several secondary assets, including *csr_we_i* in *neorv32_cpu_cp_fpu*. This asset controls only 8 of 32 FPU CSR bits (*csr_fflags[4:0]*, *csr_frm[2:0]*), giving $8/32 = 25\%$. As *csr_o* maps fully to *csr_rdata*, the overall influence remains 25%. Associated threats include fault injection, unauthorized access, and hardware Trojans (CWE-1262, CWE-1247).

To validate the outputs, we manually create a reference list of security assets for each NEORV32 IP, based on our knowledge of hardware security assurance. We further cross-validate the manually annotated assets using mainstream LLM chatbots, i.e., GPT, Gemini, and Grok. This is carried out to assess the reliability of the manual annotations against the ground truth, while also introducing an additional evaluation dimension beyond human review, which might be subject to error. We compare these manual annotations with the responses from *LAsset* whose summary is shown in Table III. Since *LAsset* deterministically computes *DoI* based on the relationships among the already validated primary and secondary assets, we do not perform a separate validation for this.

TABLE I: Summary of Assets for *Spec.+RTL* vs *RTL* approaches across IP repositories. Each block reports Reference Asset List (Golden), Total Design Elements (Elem.), Total Initial Assets after *Asset Generation* (Init.), Total Refined Assets after *Asset Refinement* (Ref.), True Positive Assets (TP), False Negative Assets (FN), and False Positive Assets (FP).

Repository	IP	Golden	Elem.	Spec + RTL				RTL					
				Init.	Ref.	TP	FN	FP	Init.	Ref.	TP	FN	FP
OpenTitan	hmac	11	268	15	11	8	3	3	14	12	8	3	4
	keymgr	42	688	46	44	37	5	7	44	42	36	6	6
	ot_gpio	5	114	8	7	5	0	2	7	7	5	0	2
OpenCores	sha3	7	71	13	12	7	0	5	10	8	7	0	1
	simple_gpio	2	17	3	3	2	0	1	5	4	2	0	2
	tiny_aes	4	141	10	8	4	0	3	13	13	4	0	8
	ahb_m_wishbone	7	31	12	9	7	0	2	9	8	6	1	2
	rc4	3	12	4	4	3	0	1	5	4	3	0	1
	robust_axi2apb	10	87	15	13	10	0	3	11	11	9	1	2
NEORV32	CPU	14	35	15	14	12	2	2	15	13	11	3	2
	PMP (CPU)	6	23	8	6	5	1	1	6	6	4	2	2
	Debug Transport	5	25	7	7	5	0	2	8	6	4	1	2
	TRNG	7	39	9	8	6	1	2	7	7	5	2	2
	UART	10	44	12	11	10	0	1	9	9	8	2	1
Total		133	1595	177	157	121	12	35	163	150	112	21	37

B. IP: OpenTitan and OpenCores IPs

For validating *LAsset* at IP-level security asset identification, we have selected a total of 21 open-source crypto IPs, Interface-GPIO IPs, and Interface-peripheral IPs, from OpenTitan [27], OpenCores [28], and other repositories. A comprehensive primary-secondary classified assets listing for a 128-bit AES architecture is shown in Table II. We compare our asset outputs with those from Nath et. al [18], as shown in Table III, since both their tool-generated list of primary assets and the benchmark datasets for the same IPs are publicly available [29].

TABLE II: Security Assets for AES-128 Design

Primary Asset		Security Objective	Secondary Asset (Design Module)	CWE-ID
Conceptual Asset	Structural Asset			
Key Expansion Logic Output	key	Confidentiality	out-1 (expand-key-128)	1300, 1191, 1239, 1258
			out-2 (expand-key-128)	1300, 1191, 1313
			key (one-round)	1300, 1247, 1191, 1239, 1258, 1263
			key (final-round)	1300, 1247, 1191, 1239, 1258, 1263
Final Encrypted Output	out	Availability	out (AES-128)	1247, 1319, 1384
			state-out (one-round)	1247, 1261, 1313
			state-out (final-round)	1247, 1261, 1313
Initial Data Processing State	state	Confidentiality	state	1300, 1191, 1258
			state-in (one-round)	1300, 1247, 1323, 1313, 1263
			state-in (final-round)	1300, 1247, 1323, 1313, 1263
			state (AES-128)	1300, 1191, 1258
			state (table-lookup)	1300, 1247, 1319
	P0, P1, P2, P3 (table-lookup)	1300, 1247, 1319		

C. Performance Comparison and Analysis

In this sub-section, we compare the performance of *LAsset*, as shown in Figure 6, to evaluate whether it aligns with the expected behavior and to analyze the underlying reasons for any observed discrepancies. A detailed, IP-wise summary of the results obtained by comparing the asset outputs along with the reference list, are presented in Table I.

1) *SoC*: From Table III, we find that the *Spec. + RTL* approach yields better results than the *Only RTL* approach in SoC-level. This signifies that RTL is the primary contributing resource, since RTL explicitly maps out all necessary design components and their functionalities, which LLMs can interpret

effectively. However, the Spec.’s natural-language description of operational behavior and interconnectivity helps the LLM better understand the design, more so than using RTL alone. Therefore, the takeaway from this analysis is that although an industry-standard Spec. alone is not enough to identify the security assets in design, it can work as a supplementary factor to the *LAsset* framework.

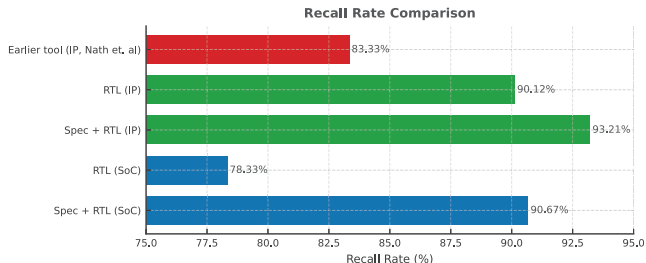


Fig. 6: Graphical Comparison between *RTL + Spec.* and *Only RTL* Approaches at SoC and IP level

2) *IP*: Table III shows that the *Spec.+RTL* approach outperforms the *Only RTL* approach at the IP-level. Compared with the work of Nath et al. [29], *LAsset* achieves a higher recall value of 93.21%, whereas their approach has 83.33%. These values are measured against their manually tailored golden asset list. Upon closely reviewing this list, however, we found that a number of actual assets are missing. For instance, in the case of the SHA-3 IP block, the round constants `rc1` and `rc2` should be included, as tampering with them would err the transformation steps and undermine the integrity of the algorithm. Because the golden list in Nath et al. [29] overlooks a significant number of such critical assets, those identified by *LAsset* are incorrectly flagged as false positives, even though they must be true positives. Hence, a direct comparison of accuracy and F-1 score between *LAsset* and Nath et al. [29] would not be a meaningful assessment; instead, recall rate provides a more appropriate metric for comparison.

TABLE III: Comparison across Different Input Configurations at SoC- and IP-levels.

Input Config	Actual Class	Predicted Positive	Predicted Negative	Performance Metric
SoC-level (NEORV32)				
RTL + Spec.	Positive	272	30	Accuracy = 93.75%
	Negative	59	1029	
RTL	Positive	235	67	Accuracy = 91.16%
	Negative	59	992	
IP-level (21 IPs from OpenTitan, OpenCores, etc.)				
RTL + Spec.	Positive	148	15	Recall = 93.21%
	Negative	50	1735	
RTL	Positive	143	20	Recall = 90.12%
	Negative	50	1735	

D. Time-Cost-Performance Comparison

Although GPT-5 model has been employed to obtain the results of *LAsset* presented in this paper, we consider the framework’s model-agnostic behavior as well. To this end, we executed the end-to-end flow for five sample IP blocks,

i.e, AES, SHA-3, GPIO, AXI-Adapter, and AHB3LITE, and compared three outcomes: recall(%), runtime, and monetary cost, among five different LLM models for each IP, as shown in Figure 7.



Fig. 7: Performance-Time-Cost Comparison among different LLM models for *LAsset*

To quantify the trade-offs, we define a model-agnostic utility (MAU) that aggregates the normalized metrics under user-specified priorities:

$$MAU = \alpha \frac{100 - R}{100} + \beta \frac{T}{T_{\max}} + \gamma \frac{C}{C_{\max}}, \quad (2)$$

where $\alpha, \beta, \gamma \geq 0$, $\alpha + \beta + \gamma = 1$.

Here, T_{\max} and C_{\max} denote the maximum observed time and cost for a design; R, T and C represents the observed recall percentage, time and cost for the corresponding LLM model, respectively; α, β , and γ indicate the selection preference (e.g., performance-centric, latency-sensitive, or cost-constrained). A lower MAU therefore corresponds to a more efficient model. For our experiments, we prioritize performance over cost and time, so we tune the equation (2) with $\alpha = 0.6$, $\beta = 0.1$, and $\gamma = 0.3$. Under this configuration, GPT-5-nano is found to be the best candidate with the least (MAU).

V. CONCLUSION

This paper introduces *LAsset*, an automated AI-assisted methodology for identifying security-critical assets in both hardware IPs and SoCs, drawing on the inherent capabilities of LLMs. *LAsset* not only identifies security-relevant primary and corresponding secondary assets in hardware designs but also associates them with relevant attack vectors and CWEs, as well as the degree of influence per secondary asset, thus aiding design and verification engineers to better understand and utilize these assets in subsequent security stages. This development further standardizes existing IP and SoC design flows while enhancing the overall hardware security verification landscape.

ACKNOWLEDGMENT

This research was funded by the U.S. National Science Foundation (NSF) under the Faculty Early Career Development (CAREER) Program, Grant No. 2339971.

REFERENCES

- [1] C. Rowen, *Engineering the complex SOC: fast, flexible design with configurable processors*. Pearson Education, 2008.
- [2] V. S. Chakravarthi and S. R. Koteswar, "Soc advanced architectures," in *System on Chip (SOC) Architecture: A Practical Approach*. Cham: Springer Nature Switzerland, 2023, pp. 127–138, ISBN: 978-3-031-36242-2. DOI: 10.1007/978-3-031-36242-2_9. [Online]. Available: https://doi.org/10.1007/978-3-031-36242-2_9.
- [3] W. Chen, S. Ray, J. Bhadra, M. Abadir, and L.-C. Wang, "Challenges and trends in modern soc design verification," *IEEE Design Test*, vol. 34, no. 5, pp. 7–22, 2017. DOI: 10.1109/MDAT.2017.2735383.
- [4] P. Kocher, "Complexity and the challenges of securing socs," in *Proceedings of the 48th Design Automation Conference*, 2011, pp. 328–331.
- [5] J. Knechtel et al., "Towards secure composition of integrated circuits and electronic systems: On the role of eda," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 508–513. DOI: 10.23919/DATE48585.2020.9116483.
- [6] H. Pearce, R. Karri, and B. Tan, "High-level approaches to hardware security: A tutorial," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 3, pp. 1–40, 2023.
- [7] S. Mitra, S. A. Seshia, and N. Nicolici, "Post-silicon validation opportunities, challenges and recent advances," in *Proceedings of the 47th Design Automation Conference*, 2010, pp. 12–17.
- [8] L. Arm, "Arm security technology-building a secure system using trustzone technology," *PRD-GENC-C. ARM Ltd. Apr.(cit. on p.), Tech. Rep, Tech. Rep.*, 2009.
- [9] J. Portillo, E. John, and S. Narasimhan, "Building trust in 3pip using asset-based security property verification," in *2016 IEEE 34th VLSI Test Symposium (VTS)*, IEEE, 2016, pp. 1–6.
- [10] P. Slpsk, J. Cruz, S. Ray, and S. Bhunia, "Protects: Secure provisioning of system-on-chip assets in untrusted testing facility," in *2023 IEEE International Test Conference India (ITC India)*, IEEE, 2023, pp. 1–6.
- [11] G. K. Contreras, A. Nahiyan, S. Bhunia, D. Forte, and M. Tehranipoor, "Security vulnerability analysis of design-for-test exploits for asset protection in socs," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2017, pp. 617–622.
- [12] E. Peeters, "Soc security architecture: Current practices and emerging needs," in *Proceedings of the 52Nd Annual Design Automation Conference*, 2015, pp. 1–6.
- [13] Accellera Systems Initiative, *Security Annotation for Electronic Design Integration Standard*, https://www.accellera.org/images/downloads/standards/Accellera_SA-EDI_Standard_v10.pdf, Apr. 2021. [Online]. Accessed: 2024-02-04.
- [14] A. Meza and R. Kastner, "Information flow coverage metrics for hardware security verification," *arXiv preprint arXiv:2304.08263*, 2023.
- [15] W. Hu, A. Althoff, Ardeshiricham, et al., "Towards property driven hardware security," in *2016 17th International Workshop on Microprocessor and SOC Test and Verification (MTV)*, IEEE, 2016, pp. 51–56.
- [16] N. Farzana, A. Ayalasomayajula, F. Rahman, F. Farahmandi, and M. Tehranipoor, "SAIF: Automated Asset Identification for Security Verification at the Register Transfer Level," in *2021 IEEE 39th VLSI Test Symposium (VTS)*, 2021, pp. 1–7. DOI: 10.1109/VTS50974.2021.9441039.
- [17] B. Ahmad, H. Pearce, R. Karri, and B. Tan, "LASHED: LLMs And Static Hardware Analysis for Early Detection of RTL Bugs," *arXiv preprint arXiv:2504.21770*, 2025.
- [18] S. K. D. Nath and B. Tan, "Toward automated potential primary asset identification in verilog designs," in *2025 26th International Symposium on Quality Electronic Design (ISQED)*, IEEE, 2025, pp. 1–7.
- [19] D. Saha et al., "Llm for soc security: A paradigm shift," *IEEE Access*, vol. 12, pp. 155 498–155 521, 2024.
- [20] D. Saha et al., "Sv-llm: An agentic approach for soc security verification using large language models," *arXiv preprint arXiv:2506.20415*, 2025.
- [21] S. Tarek, D. Saha, et al., "Socurellm: An llm-driven approach for large-scale system-on-chip security verification and policy generation," in *2025 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2025, pp. 335–345.
- [22] D. Saha, H. Al Shaikh, S. Tarek, and F. Farahmandi, "Special session: Threatlens: Llm-guided threat modeling and test plan generation for hardware security verification," in *2025 IEEE 43rd VLSI Test Symposium (VTS)*, 2025, pp. 1–5. DOI: 10.1109/VTS65138.2025.11022932.
- [23] S. Tarek, D. Saha, S. K. Saha, and F. Farahmandi, "Bug-whisperer: Fine-tuning llms for soc hardware vulnerability detection," in *2025 IEEE 43rd VLSI Test Symposium (VTS)*, IEEE, 2025, pp. 1–5.
- [24] D. Saha, K. Yahyaei, S. K. Saha, M. Tehranipoor, and F. Farahmandi, "Empowering hardware security with llm: The development of a vulnerable hardware database," in *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2024, pp. 233–243.
- [25] IEEE P3164 Working Group, *Asset Identification for Electronic Design IP*, <https://ieeexplore.ieee.org/document/10496567>, Apr. 2024. [Online]. Accessed: 2024-02-04.
- [26] *CWE - Common Weakness Enumeration*, <https://cwe.mitre.org/index.html>, Accessed: 2024-02-04, MITRE, 2024.
- [27] *OpenTitan: Open Source Silicon Root of Trust (RoT)*, <https://opentitan.org/>, Accessed: 2024-02-04.
- [28] *OpenCores Projects*, <https://opencores.org/projects>, OpenCores. [Online]. Accessed: 2024-02-04.
- [29] CalgaryISH, *Asset Dataset for Crypto, GPIO, and Peripheral IPs using Partial Keyword and Signal Category-based Automatic Tool*, https://github.com/CalgaryISH/Asset_Dataset_using_PKG, GitHub. [Online]. Accessed: 2024-02-04.