2025
Volume 21, Number 4

ACM Journal on

Emerging Technologies in Computing Systems

Association for Computing Machinery

Advancing Computing as a Science & Profession

ACM Journal on Emerging Technologies in Computing Systems

Vol. 21 • No. 4 • 2025

Articles 11–14

ACM Journal on
Emerging Technologies in Computing Systems

ACM
1601 Broadway, 10th Floor
New York, NY 10019-7434
Tel.: (212) 869-7440
Fax: (212) 869-0481
https://www.acm.org

Home Page: https://jetc.acm.org/

The *ACM Journal on Emerging Technologies in Computing Systems* (*JETC*) (ISSN: 1550-4832 ; e-ISSN: 1550-4840) is published quarterly in Spring, Summer, Fall, and Winter by the Association for Computing Machinery (ACM), 1601 Broadway, 10th Floor, New York, NY 10019-7434.

Association for Computing Machinery

Advancing Computing as a Science & Profession

A Persistent Hierarchical Bloom Filter-based Framework for Scalable Authentication and Tracking of ICs

MD MASHFIQ RIZVEE, FAIRUZ SHADMANI SHISHIR, TANVIR HOSSAIN, and TAMZIDUL HOQUE, Electrical Engineering and Computer Science, University of Kansas, Lawrence, Kansas, USA
DOMENIC FORTE, Electrical and Computer Engineering, University of Florida, Gainesville, Florida, USA
SUMAIYA SHOMAJI, Electrical Engineering and Computer Science, University of Kansas, Lawrence, Kansas, USA

Due to the reliance on untrusted supply chain entities, tracking and authentication of Integrated Circuits (ICs) has become crucial to prevent the rapid proliferation of counterfeits. Physically Unclonable Functions (PUFs) can be used for such IC authentication since they generate unique identifiers for individual ICs. However, PUF-generated signatures are often noisy and traditional solutions like Error Correcting Codes (ECC) are expensive and vulnerable to attacks. Moreover, comprehensive PUF-based authentication at multiple locations of the supply chain at any given time suffers from large storage requirements, high query processing time, and security threats. This article proposes a Persistent Hierarchical Bloom Filter (PHBF) to enable fast, storage-efficient and noise-tolerant authentication to track ICs across the supply chain. The proposed framework is demonstrated using 4,000 PUF-generated signatures from several FPGAs and achieved the highest possible authentication accuracy under temperature-induced and synthetic noise of varied degrees without any ECC. Our comparative analysis of storage and query time requirements against four different solutions for detecting wide range counterfeit ICs shows the significant benefit of PHBF, providing up to 10^5 times faster query processing and 39 times lower storage requirement compared to blockchain.

CCS Concepts: • **Hardware → Integrated circuits**; • **Information retrieval → Information retrieval query processing**; • **Security and privacy → Database and storage security**;

Additional Key Words and Phrases: Hierarchical Bloom Filter, Persistent Bloom Filter, Physically Unclonable Function, Electronic Supply Chain, Counterfeit ICs, IC Authentication

ACM Reference format:
Md Mashfiq Rizvee, Fairuz Shadmani Shishir, Tanvir Hossain, Tamzidul Hoque, Domenic Forte, and Sumaiya Shomaji. 2025. A Persistent Hierarchical Bloom Filter-based Framework for Scalable Authentication and Tracking of ICs. *ACM J. Emerg. Technol. Comput. Syst.* 21,4, Article 11 (September 2025), 22 pages.
https://doi.org/10.1145/3748650

Authors' Contact Information: Md Mashfiq Rizvee (corresponding author), Electrical Engineering and Computer Science, University of Kansas, Lawrence, Kansas, USA; e-mail: mashfiq.rizvee@ku.edu; Fairuz Shadmani Shishir, Electrical Engineering and Computer Science, University of Kansas, Lawrence, Kansas, USA; e-mail: shishir@ku.edu; Tanvir Hossain, Electrical Engineering and Computer Science, University of Kansas, Lawrence, Kansas, USA; e-mail: tanvir@ku.edu; Tamzidul Hoque, Electrical Engineering and Computer Science, University of Kansas, Lawrence, Kansas, USA; e-mail: hoque@ku.edu; Domenic Forte, Electrical and Computer Engineering, University of Florida, Gainesville, Florida, USA; e-mail: dforte@ece.ufl.edu; Sumaiya Shomaji, Electrical Engineering and Computer Science, University of Kansas, Lawrence, Kansas, USA; e-mail: shomaji@ku.edu.

ACM 1550-4840/2025/9-ART11
https://doi.org/10.1145/3748650

1 Introduction

The ubiquitous demand for electronic devices and the globalization of device manufacturing exacerbate the difficulty of ensuring device authenticity and a trustworthy supply chain. Attackers can get more room to tamper, re-package, and counterfeit at different stages performed at various untrusted facilities. Additionally, the suppliers are financially affected when large scales of counterfeit **Integrated Circuits (ICs)** are on market and this also poses a security risk to mission-critical systems in the military, avionics, medical applications, and federal cyber infrastructure. Hence, there has been a demand for a large-scale and efficient authentication and tracking system for the development of a traceable supply chain. Such authentication and tracking will also be advantageous for upcoming growth in the domestic semiconductor supply chain bolstered by federal investments through the CHIPS and Science Act [12].

Tracking ICs is important for ensuring the authenticity and quality of digital devices, particularly in high-security and safety-critical applications. **Physically Unclonable Function (PUF)** is a security primitive that can be used to trace ICs by creating a unique digital signature for each chip. PUF works by using the inherent variations in the manufacturing process of ICs to create a unique and unpredictable signature that is difficult to replicate or clone. This signature can be used to identify and trace each IC, allowing for a secure and tamper-proof way to track and record the origin and transfer of ownership throughout their lifecycle. However, environmental and other external factors such as temperature, aging, and voltage variation can cause the output of the PUF (i.e., responses) to change over time for a given input (i.e., challenges). Thus, the PUF-based authentication may not correctly authenticate the ICs under such variations. While solutions like **Error Correcting Code (ECC)** exist to mitigate this challenge, they introduce large overhead and open the door for side-channel attacks to leak the PUF responses [1].

Authentication speed and storage efficiency are another serious concern due to the large volume of tracking data representing the flow of billions of ICs across a large number of vendors throughout long periods of the manufacturing and distribution cycle. In [19], the authors proposed an architecture for semiconductor IC traceability using blockchain. However, the authors did not consider any measure to address the potential error in PUF responses. Moreover, the proposed system only authenticates the ICs and does not track location information in the supply chain. In [9], a distributed ledger-based method is proposed to establish an authentication system and distinguish various counterfeit ICs with a smart contract in the blockchain. However, the framework does not show any solution for cloned and stolen ICs in the supply chain. In [16], the authors focused on an IC traceability solution using blockchain that facilitates fabless Original Design Manufacturers maintaining tracking records along with sales channels. They generated random numbers as unique IDs for the tracking of the ICs and transferred the public key using a cryptographic module. While these blockchain-based solutions may promise strong authentication against diverse counterfeit types, the associated storage and runtime requirements for tracking millions of ICs throughout the supply chain for a long period would be extremely high.

Bloom Filters (BFs) [3] can be used to enable efficient authentication and tracking of a large number of ICs. A BF is a probabilistic data structure that can be used to test whether an element is a member of a set. In the context of IC tracking, a BF can be used to store the **Challenge–Response Pairs (CRPs)** of all manufactured ICs. When a new IC is produced, its responses (i.e., signature) for a specific set of challenges can be enrolled in the BF. To check if an IC is authentic, the responses obtained from that IC using the same set of challenges are queried to the BF with all authentic ICs. If the responses are not found there, the IC is likely to be a counterfeit. Using BFs for IC tracking has the advantage of being significantly fast and storage-efficient, but it does have some limitations. For example, **False Positives (FPs)** are possible, and the size of the BF must be chosen carefully to

ensure a low FP rate. Moreover, BFs cannot deal with noisy data. Therefore, any error in the PUF responses may result in an erroneous membership test. **Hierarchical Bloom Filters (HBFs)** [17], on the other hand, have a minimal FP rate and can check queries even if they are noisy. However, HBFs are inadequate in handling temporal membership query processing that is much needed to enable queries with timestamps (e.g., location of chip X at a specific time). While **Persistent Bloom Filters (PBFs)** [15] can be used to enable such queries, they suffer from the same problem as the BFs; as PBFs cannot deal with noisy data. On the contrary, blockchain or other tracking systems have also been used in the supply chain, but, they are not noise-tolerant either. They can only verify noise-free samples and do not provide fast and area-efficient tracing as suggested by the quantitative analysis presented in later sections.

Utilizing the concepts of the noise tolerance in HBFs combined with the temporal membership query ability in PBFs, in this article, we propose a noise-tolerant and scalable framework, **Persistent Hierarchical Bloom Filter (PHBF)**, that can efficiently store PUF responses in a hierarchical structure and make the IC traceability process faster. PHBF can enable authentication even with erroneous PUF responses, eliminating the need for ECC and associated overhead issues. PHBF also supports temporal membership queries that can be leveraged to enable complex queries to detect a wide range of counterfeits. Unlike blockchain-based tracking, PHBF enables significantly faster query processing and requires smaller storage for tracking large number of ICs. Overall, this article makes the following major contributions:

(1) We have designed an authentication and tracking framework to detect a wide range of counterfeit types by integrating HBF and PBF.
(2) Based on the existing taxonomy of counterfeit ICs, we have developed a set of queries that can be performed using PHBF to identify different counterfeit types.
(3) To demonstrate the error tolerance, we introduced temperature-induced and synthetic errors in 4,000 PUF responses from several FPGAs and showed robust authentication under different error amounts and patterns.
(4) Finally, we evaluated the speed and storage efficiency through experiments with multiple sets of query data. We compared our query and storage complexity with four other tracking frameworks and observed 10^5 times faster query processing and 39 times lower storage requirement compared to blockchain which is considered as the state of the art.

The rest of the article is organized as follows: Section 2 describes the related studies on IC traceability, and Section 3 presents the relevant background. Section 4 presents the proposed PHBF framework to establish IC supply chain traceability. Section 5 shows the authentication results of PHBF, and Section 6 shows an analysis of search and space complexity. Finally, the subsequent one, Section 7 discusses the conclusion and future work.

2 Related Work

Most authentication and traceability frameworks to address counterfeit ICs use various semiconductor chip IDs and PUF CRPs. PUF-based authentication [5] provides every chip with a unique and unclonable ID and authentication can be done in every stage of a supply chain after fabrication and in the field by leveraging this signature. This framework requires the initial enrollment of CRPs from all authentic ICs into a database. A chip can be authenticated by simply querying the server with the comprehensive CRP database of all ICs. However, without a comprehensive authentication and tracking protocol, a PUF alone cannot prevent or detect all forms of counterfeiting that may occur. Authors in [10] proposed a blockchain-based approach for counterfeit IC detection across the entire supply chain. On the other hand, the authors in [18] used a blockchain-based distributed system to authenticate the PUF responses; however, they did not consider the possibility of noise in

Table 1. Comparison between the Proposed PHBF and Related Works

References	Low Storage	Fast Query	Protection of Enrolled Data	In-house Noise Handling Property	Low Cost
[19]	×	×	✓	×	×
[10]	×	×	✓	×	×
[16]	×	×	✓	×	×
[18]	×	×	✓	×	×
[9]	×	×	✓	×	×
[2]	×	×	✓	×	×
Proposed PHBF	✓	✓	✓	✓	✓

the PUF response that may cause authentication to fail. Even though the PUF error can be mitigated using ECC within the chip, it would introduce significant overhead and create the possibility of CRP leakage through side-channel attack [1]. Work in [2] is also based on developing a counterfeit mitigation system which runs inside a blockchain using a smart contract. Similar to [18], this system is not noise-tolerant. Furthermore, the framework does not track location information across the supply chain. A blockchain-based verification system that can store transaction time, chip marking, and IDs is proposed in [19]. Even though their solution can detect overproduced, remarked, and recycled ICs, scalability is a significant concern in the blockchain-based framework. Existing studies present various fast, storage-efficient, and error-tolerant data structures for biometric authentication. For example, Shomaji et al. proposed a novel probabilistic data structure, HBF, that can handle noise in data and provides fast and storage-efficient authentication [17]. However, there was no extended work to leverage HBF for IC authentication, and incorporating timestamps in HBF is still an open research question. Table 1 compares the proposed PHBF with relevant works with respect to storage requirement, query speed, noise tolerance, the privacy of the enrolled template, robustness, and computational cost. The term, "In-house" means that inherently, by the nature of PHBF, the noise can be handled during query. It is within the process of query processing step, and we do not need a separate module to achieve this. Moreover, In a supply chain, myriads of challenges may arise, such as increased time range, storage requirements, changes in location, or even ICs altering their PUF responses due to noise. Our proposed method, PHBF, can sustain these changes and continue to perform optimally. It is designed to handle variations in these parameters without causing inconsistencies in performance which makes it robust."

3 Background

In this section, we introduce some preliminary concepts that are related to our proposed system. Multiple notations have been used in this article which is described in Table 2.

BF. A BF is a probabilistic data structure that is used to test whether an element is a member of a set [3]. The idea behind a BF is to use a fixed-size bit array and a set of hash functions to map elements to positions in the array. When an element is added to the set, the corresponding bits in the array are set to "1." When a query is made to check if an element is in the set, the corresponding bits are checked and if all of them are "1," the element is considered to be in the set (see Figure 1(a)). However, if any of the bits are "0," the element is definitely not in the set. FP rate of a BF is the probability that a membership query for an item returns "probably in set" when the item is actually not in the set. The FP rate is a crucial metric for BF, as it determines the accuracy of the filter in identifying membership in a set. If m number of bits are used to build a BF and k number of hash functions are used on each instance of the n number of elements, the probability of FP is, $Prob\ (p) = (1 - e^{-k(n/m)})^k$. However, BFs are inadequate in handling corrupt or noisy data. While doing a string matching operation, if the query string is almost as similar as the enrolled string but not exactly a duplicate; while querying, it will return a negative result in the membership testing. Thus, one small change in the query string may result in different hash values.

Table 2. Notations and Their Descriptions

Notation	Description
L	Number of locations in supply chain
n	Number of ICs
N	Number of BFs in a HBF
T	Amount of time (in number of days)
M	Length of the responses
K	Number of hash functions in a BF
d	Length of the location name's string representation (for the traditional database)
l	Response length of the string of one IC's PUF
t	Length of the time string representation (for the traditional database)
H	Number of hash tables (for LSH)
P	Bits needed for each transaction (for blockchain)
q	Bits needed to store one character
m	Number of bits in a single BF
p	FP probability of a BF

LSH, Locality-sensitive Hashing.

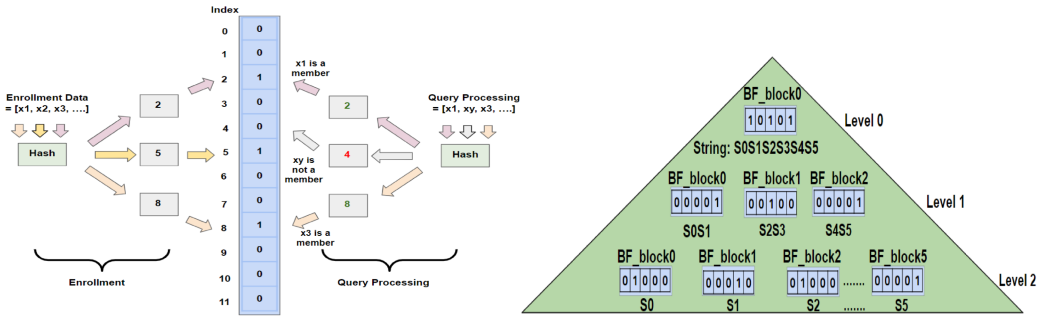


Fig. 1. (a) Enrollment and query processing in BF and (b) structure of a HBF.

HBF. HBFs were introduced to address the problem of substring matching [17]. HBFs consist of multiple levels of BFs to check whether an element is part of a string. HBFs are segmented BFs that are arranged into multiple non-overlapping blocks building a hierarchy of BFs getting stacked on multiple levels. The block sizes can be geometrically increasing. Figure 1(b) provides a clear example of this hierarchy. To insert a string into the hierarchy, the string is divided into $\lceil M \div N \rceil$ segments and inserted into the BF blocks of HBF starting from level 0. Subsequently, at the next level, three segments of the string are inserted into the HBF at level 1, and this process continues. In Figure 1(b), the string "S0S1S2S3S4S5" is first added as a whole at Level 0 at the top of the hierarchy. Then, "S0S1," "S2S3," and "S4S5" are inserted at level 1. Finally, each segment of the string is inserted individually at level 2 in six BF blocks. Authentication is done by matching smaller chunks of BFs. If the query matches a predefined threshold number of BFs, such string will be considered as a member of the database. To keep the FP rate low, the smallest blocks of BFs are inserted in the lower levels of the HBF. In [17], the implementation of the HBF was initialized by defining FP_{BF} as the FP of BF and set FP_{BF} ≈10%. Subsequently, if n is the total elements stored in the HBF, k is the number of hash functions, m is the size of the BF (in bits) and d is the number of levels, then the BFs are arranged in a hierarchy of d levels developing an HBF where FP_{HBF} ≈0%. In order to address the noise tolerance, if N is the number of bits that are flipped by the noise in

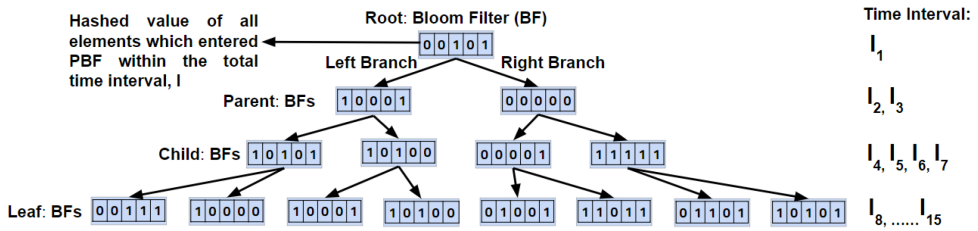


Fig. 2. Schematic of a PBFs. PBFs, Persistent Bloom Filters.

an instance and B is the number of bits in a template, the probability of bits that are flipped in a particular location, $Prob\,(Bit\ Flips) = \frac{N}{B}$. Thus, the probability of at least one bit flipped by the noise for a single block of BF of length l_0 is, $Prob\,(At\ least\ One\ Bit\ Flip) = 1 - (1 - \frac{N}{B})^{l_0} \approx 1$. The size of each BF block l_0 can be calculated using the mentioned equation. In our design, we decided to have an equal length of BFs. However, HBFs may contain BFs with varied lengths which is not considered in this study.

PBF. PBFs were introduced to address the temporal data query in a system [15]. A temporal query is a type of query used to retrieve data based on a specific time or date. It allows a user to search for information within a specific time frame, such as finding all events that occurred between two specific dates. Structurally, a PBF has similarities with a **Binary Search Tree (BST)**. In a PBF, the BFs are organized into a hierarchy of smaller units, with each BF covering a specific time interval such as a day or an hour (see Figure 2). This allows the PBF to track changes in the database over the total time (I) and efficiently handle insertions and query processing. In Figure 2, the root node in PBF stores all elements entering the database within the total time frame I_1, where $I_1 \subseteq I$. Subsequently, As the tree-like structure gets deeper into multiple levels, the binary decomposition of each interval, $(I_2, I_3, \ldots, I_{15})$ correspond to the enrolled element BFs of those specific levels. The PBF also follows a two-step operation: enrollment and membership query processing. In PBFs when an element is inserted with its corresponding timestamp, a **Depth-first Search (DFS)** is used over the binary decomposition tree to find a path from the root node to the leaf node that contains the timestamp. The element is inserted into every BF along the path. While searching for an element, DFS is used to find the canonical cover of the query time range and the query is performed in every BF of the time range intervals. If any of the BFs return TRUE, the membership of that element gets confirmed.

PUF. PUF can be used to detect various forms of counterfeit ICs. It generates authentic response set $\{r_1, r_2, \ldots, r_n\} \in R$ with respect to a given challenge set $\{c_1, c_2, \ldots, c_n\} \in C$ by using the inter- and intra-die variation of the silicon die, V, and environmental noise, E. Hence, these PUFs can be described as $R = F_{PUF}(C, V, E)$. Based on the size of CRPs, PUFs are usually two kinds: strong PUFs and weak PUFs. PUFs with a large number of CRP sets are recognized as strong PUFs. The large set of CRPs enable strong PUFs to be used in challenge-response authentication protocols.

4 Methodology

In this section, the methodology of the proposed PHBF framework has been discussed in detail. We begin with presenting the overview, construction, and system operation of PHBF in Section 4.1, and we discuss its role in the particular type of counterfeit IC detection in Section 4.2.

4.1 Proposed Data Structure

The proposed framework PHBF is a novel data structure which is designed for scalable and robust PUF-based authentication of ICs. PHBF is a fusion of HBF [17] and PBF [15]. By fusing the two

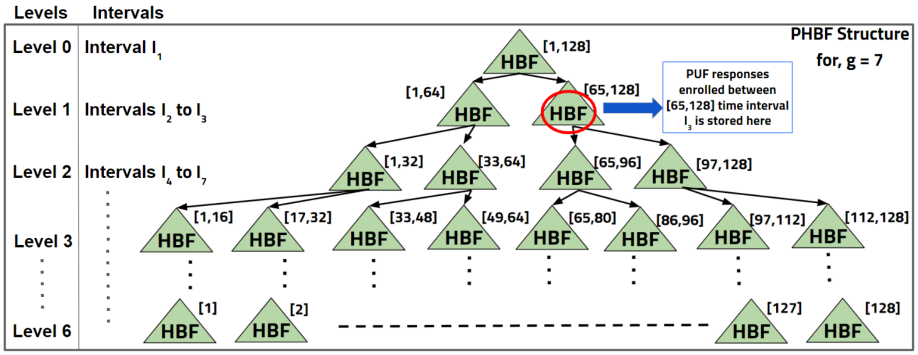


Fig. 3. A schematic of the proposed PHBF framework.

concepts, PHBF combines the noise tolerance property of HBF with the temporal membership testing ability of the PBF. It follows a two-step operation: enrollment and temporal membership query processing. Existing literature in [15] uses the term "Persistent" to indicate the ability to support temporal membership queries using both time range and timestamps. Since this is also one of the features provided by our proposed data structure PHBF, we have included this term in naming the framework. To briefly explain the operations, during the enrollment phase, a PHBF-based database is generated to enroll the ICs with their weak PUF responses whereas, during the query processing stage, a temporal membership query processing is performed at the PHBF to determine the authenticity of an unknown query IC at a given time, e.g., "*Was Chip X at Location-A at 5 PM?*". The reason for considering temporal membership testing is significant here. That is, when an IC is an in-field operation, a traditional membership query processing (which does not involve timestamp during query) may suffice, but when the IC is in a supply chain, the temporal query processing (which considers timestamps) tends to be more effective since it not only assures the authenticity of an IC but also confirms whether the IC was spotted at a particular location at a given time.

Below we describe the construction process of a PHBF followed by the enrollment and the temporal membership query processing details.

PHBF Construction. The construction of the proposed PHBF requires the combination of HBF and PBF. By combining both structures together, we achieve the benefits of both under one framework. A schematic of the PHBF is presented in Figure 3. As shown in the figure, the PHBF is organized in a hierarchy where on each level, HBFs are considered as nodes instead of BFs. Since HBF is the basis of PHBF, and BF is the basis of HBF, we first initialize the associated parameters of BF and HBF. These are the inputs for our proposed framework: number of ICs to be tracked (n), number of locations (L), and number of days (T). For a given n, we initialize the no. of BFs (N), size of each BF (m), no. of hash functions (k) using the equations from [3] and [17]. Such parameter initialization serves as the pre-requisite of the PHBF's initialization. Next, utilizing the T input, we decide the number of levels as well as nodes for the PHBF. Finally, for L no. of locations, we employ L number of PHBFs.

One great feature of the HBF is that it is possible to determine a threshold that helps to achieve a desired authentication performance. As shown in [17], one can tune the FP as well as the associated parameters to determine the optimum threshold. Thus, to define the threshold (FP_{th}) which determines the authentication decision, we consider the following formula: (Please refer to

[17] for the detailed explanation):

$$FP_{th} = (1 - p_{BF}) = \sum_{i=0}^{[N]} \left(\begin{array}{c} N \\ i \end{array} \right) (1 - p_{BF})^{ki} \, (1 - (1 - p_{BF})^{k})^{(N-i)}, \tag{1}$$

where p_{BF} is the probability of having a falsely-set-to-1 bit in a BF, k is the total number of hash functions, and i is the number of levels in HBF.

Once the parameters associated with BF and HBF are determined, the T input is utilized to complete the remaining PHBF initialization process. The consideration of T which is a universal set of days means that the total time frame that we are going to cover is T days. We restrict the time-span using time granularity, g where $g \leq \log_2(T)$ and $2^g \leq T$ to control the number of leaf nodes of intervals in the tree. Binary decomposition of time-span $[1, T]$ consists $Level = \left\lceil (\log_2(\frac{T}{g})) \right\rceil + 1$ levels indexed by $l = 0, 1, 2, \ldots, Level - 1$ where $Level$ 0 is the root node and $Level - 1$ contains the leaf level with $[\frac{T}{g}]$ intervals each of length g. Note that g is used to initialize the upper-bound but it will support any queries up to any number of days that is not a power of two. Level l for each $l \in [0, Level - 1]$ contains 2^l intervals of length $T/2^l$ each. As per the formula mentioned, level 0 or the root node contains longest time interval $\{[1, T]\} \in T$. Subsequently, level 1 contains $\{[1, \frac{T}{2}], [\frac{T}{2} + 1, T]\}$, and level l contains $\{[1, \frac{T}{2^l}], [\frac{T}{2^l} + 1, 2\frac{T}{2^l}], \ldots, [T - \frac{T}{2^l} + 1, T]\}$ time intervals. Thus, the total number of temporal intervals is $u = (2(T/g) - 1)$. A binary tree is formed using temporal intervals. The intervals are indexed in a level-ordered fashion which leads to an interval set $I = \{I_1, I_2, \ldots, I_u\}$. PHBF is organized into $Level$s and consists of HBFs $\{hbf_1, hbf_2, \ldots, hbf_u\}$ such that level l, for $l \in [0, Level - 1]$, consists of HBFs $\{hbf_{2^l}, \ldots, hbf_{2^{l+1}-1}\}$. Each of the HBFs is constructed with l_0 length of BFs with m bits. Now, if we consider the number of enrolled responses n, we achieve the optimal number of hash functions using:

$$k = \left(\frac{m}{n}\right) \ln 2. \tag{2}$$

In order to achieve irreversibility, a cryptographic hash function, SHA256 was used combined with a non-cryptographic modulus function **Fowler-Noll-Vo (FNV)** [14] as done in work [17]. SHA256 has the ability to generate 256-bit outputs with a reasonable level of uniformity. The FNV helps to further reduce the collision probability of the hash functions and helps to limit the output length to m bits. We leverage the complementary advantages after combining both the cryptographic and non-cryptographic functions. Such as the low collision probability, faster processing, and improved security.

Enrollment and Query Process in PHBF. To illustrate the two-step operation, during the enrollment process an enrollment set of PUF responses along with their timestamps are taken into consideration for enrolling the ICs in the PHBF database. These responses are generated by applying a set of challenges which are consistent across all the enrolled ICs. Therefore, only the responses need to be enrolled in the PHBF. During the temporal query processing step, the PHBF is utilized to search for an unknown query IC's presence for a given time. With respect to the application scope of our proposed methodology, the definition of temporal membership testing is based on an upper time limit of e and a lower time limit of s where $s, e \in T$ and a timestamp $t \in T$. The upper and the lower limit of time e and s, where $1 \leq s \leq e \leq T$ can be used to define a range $[s, e]$. A universe of PUF responses, R and an instance of a PUF response, $r \in R$. A temporal membership testing is set to be asked if $(r, t \mid t \in T) \subseteq (R, [s, e])$. This means that, given a PUF response r, and a time range $[s, e]$, the querying user want to know if the response r exists within the time range $[s, e]$. In BFs, a temporal membership test not only takes linear time complexity to perform a search but also raises the FP rate [15]. That being said, PBFs reduce the number of membership test

queries by decomposing a temporal query range using dyadic intervals. A BF is then built for the corresponding element set of each temporal range resulting from the binary decomposition.

In PBFs, the insertion and query processing is similar to segment tree operations. When enrolling an element "x" at a timestamp "t," we use DFS over the binary decomposition tree to find a path from the root to the leaf containing "t" and insert "x" to every HBF along the path. If I is the universal set of time intervals, in case of a query $Q(x, [s, e])$ where $[s, e]$ is a time range and $[s, e] \subset I$, we use DFS to find the binary decomposition of the range $[s, e]$ and query each HBF of the intervals. The query is successful only if at least one HBF returns TRUE for the membership testing of "x." Formally, we first define the canonical cover of range $Cover([s, e]) = \{I_{a_1}, \ldots, I_{a_i}\}$, where $a_i \in [1, u]$ are the index values of the intervals that are contained in the cover. For the sake of simplicity, considering $Cover([s, e])$ cover the query interval exactly, We answer $Q(x, [s, e])$ where $q(x, hbf_i)$ means a membership test of "x" at hbf_i:

$$Q\left(x, [s, e]\right) = q\left(x, hbf_{a_1}\right) \vee q\left(x, hbf_{a_2}\right) \vee \ldots q\left(x, hbf_{a_i}\right). \tag{3}$$

For instance, if we want to find $Cover([17, 48])$, illustrated in Figure 3, the result is $[I_9, I_{10}]$. Since, $I_9 = [17, 32]$, and $I_{10} = [33, 48]$ we will get our membership test answer of "x"s membership test in PHBF using:

$$Q\left(x, [17, 48]\right) = q\left(x, hbf_9\right) \vee q\left(x, hbf_{10}\right). \tag{4}$$

If at least one of the HBFs returns TRUE, the data structure considers "x" to be found. That is, the IC was located at the location associated with the PHBF within the time frame of day 17 to day 48. In the following subsection of the methodology, we discuss a practical application of our framework for PUF-based authentication of ICs.

4.2 Role of PHBF in Counterfeit IC Detection

The IC supply chain is a complex and global system that involves many different players, including IC designers (often fabless companies), foundries, assembly and test companies, distributors, and end users. The IC supply chain interconnection has led to many benefits, such as lower costs, increased efficiency, and more innovation, but it also poses many challenges, such as the risk of counterfeit ICs, the risk of intellectual property theft, and the risk of supply chain disruptions. Traceability and monitoring systems are used to monitor the movement of ICs throughout the supply chain, and to ensure that they are being used as intended.

Supply Chain Configuration. We follow [10] in terms of modeling the supply chain. In Figure 4, we present a schematic of our supply chain. Similar to the work in [10], we assume that all our entities are trusted. In our supply chain configuration, we keep it simple by only having four entities (see Figure 4(a)). It is also important to note that, only the legitimate IP owners registered with a designated consortium can claim the initial ownership of an IC and store the relevant PUF data in the PHBF. **Original Equipment Manufacturer (OEM)**/Manufacturer, Distributor, Retailer—at each end, we assume that there is at least one PHBF-based database to store IC PUF responses (see Figure 4(b)). And at the Manufacturer-end only, there is an HBF that records the PUF responses of the sold ICs. Another assumption is that separate PHBFs are kept to store information related to each die or wafer the IC is produced from. The markings could be inspected to find the corresponding PHBF. Every entity of the supply chain including the customer is connected to a centralized server that records the PHBF's enrollment information of each entity: Manufacturer, Distributor and Retailer. Every entity of the supply chain is able to put information requests into the server and get an informative response.

At this point of this subsection, using the taxonomy of counterfeit ICs, we discuss the types of counterfeits that exist in the market. The taxonomy of counterfeit ICs refers to the different

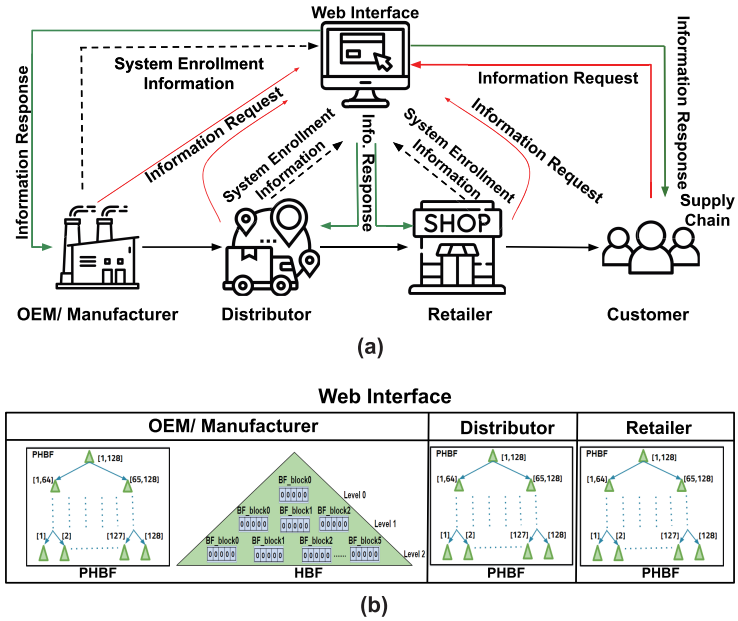


Fig. 4. Supply chain information flow to different entities. (a) Depiction of the proposed system as a whole and (b) information that can be accessed through web interface.

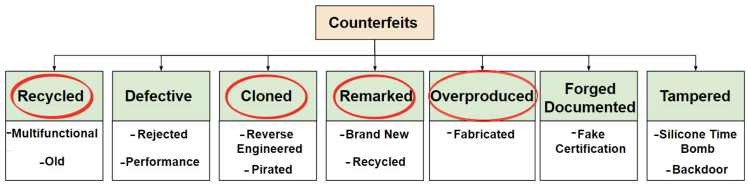


Fig. 5. Taxonomy of counterfeit ICs (red circles: counterfeits that PHBF can detect).

categories or types of IC counterfeits, based on their characteristics, origin, and the methods used to produce them (see Figure 5). Some common categories mentioned in the taxonomy of counterfeit ICs include: Recycled, Remarked, Overproduced, Defective, Cloned, Forge Documented and Tampered. Below we describe four types of counterfeit ICs that are relevant to our proposed system:

Recycled refers to electronic components or ICs that have been used in a device, removed from that device, and then sold as new or "like new." These recycled components may have been harvested from devices that were defective, damaged, or end-of-life. They may also be from devices that were stolen, lost or salvaged. ICs that have been used, may be damaged, degraded, or otherwise altered, which can make them unreliable or unsafe to use, especially in critical systems and infrastructure. Recycled counterfeit ICs can present a significant risk to the end user, as they may not perform as expected or may even cause damage to the equipment they are used in.

Cloned ICs, on the other hand, are copies of existing ICs that are produced by either reverse engineering the original IC or stealing the IP (e.g., intellectual property leaks from the foundry). These clones are often produced by companies in countries with lower labor costs and then sold at a lower price than the original IC. Cloned ICs can be a cost-effective solution for companies looking to reduce their production costs, but they can also pose a risk to the original IC manufacturer, as they may lose revenue due to the sale of cheaper cloned ICs.

Table 3. Relationship between Proposed Query Set and the Types of Counterfeits

No.	Counterfeit Type	Query	Relation between Query and Counterfeit Type Determination
1	Theft	What was ChipX's last location?	ICs are expected to move through specific stages in the supply chain. Therefore, if the chip visits every location of the presumed trajectory and we find it enrolled in every PHBF, we can say that the chip is not missing or stolen.
2	Cloned or overproduced	Is ChipX enrolled at the OEM's PHBF?	In this case, if the chip marking matches but the response do not, we can conclude that the chip is cloned or overproduced. To detect this type of counterfeit, performing a query only at the OEM-end PHBF is enough since it is the starting point of the supply chain.
3	Remarked	Is ChipX present at OEM's PHBF but the external marking is different than that of OEM's?	Remarked ICs could be detected if the chip response matches but the marking does not when a query is performed at the OEM's end.
4	Recycled	Has ChipX been sold to an end user?	In order to keep track of the sold ICs, we keep an HBF at the OEM's end so that the responses could be recorded whenever a chip is sold to an end user. If our query chip response already exists into the HBF, we can say that the chip is recycled since it was previously purchased by someone.

Remarked ICs refers to components that may have been used in a previous application and then modified or repackaged with new markings, such as a different part number, grade, lot, or manufacturer's name. Similar to recycled components, they may not perform as expected due to their inappropriate marking.

An *Overproduced* IC refers to a situation where a counterfeit IC has been manufactured in excess of the original production quantity. This can happen when a third party foundry produces more units than the original designer requested. These chips may be untested and thus can create a risk for the end user, as the counterfeit ICs may not perform as expected or may even cause damage to the equipment they are used in. If ICs are overproduced and these overproduced ICs are distributed throughout the supply chain, there is a risk that an unauthorized party might sell these ICs. If this happens, the overproduced ICs could reach customers through unauthorized channels, potentially causing damage to the revenue and reputation of the original designer. This highlights the importance of ensuring that only authorized entities handle and distribute ICs within the supply chain to prevent such issues.

Role of PHBF to Detect Counterfeit ICs. Using a PHBF to store PUF responses has several advantages. It allows for the efficient storage and retrieval of a large number of PUF responses, while still providing a higher level of security. It also provides a fast and efficient way to verify the authenticity of an IC. Using our data structure, it is possible to detect at least four types of IC counterfeits: Stolen, Cloned/Overproduced, Remarked, and Recycled. We propose a set of queries that helps us determine if a particular IC is a counterfeit or not. If the IC is not authentic, the query set will also help us identify which counterfeit class it belongs to. The proposed set of queries are shown in Table 3. We consider that each of the locations in the supply chain consists of at least one PHBF. Additionally, for the OEM-end only, an HBF is designed separately to record the IC responses of those ICs that were sold to the end users. Before performing any of the queries listed, we make an assumption that, at the OEM-end, for particular types of ICs coming from different dies or wafers, the OEM maintains separate PHBFs to store the information related to each of the respective dies or wafers. While performing the queries, we first identify the markings of the ICs. We try to find the corresponding PHBF first using the die ID or the wafer ID.

The type of counterfeit and the association of each query is listed in Table 3. In the enrollment phase, we store the PUF responses of the ICs which have visited a particular location. For our first query, "What was Chip X's last location?". We simply go to each location and search using the PUF response to make sure that the IC has visited those particular places at least once within a range of

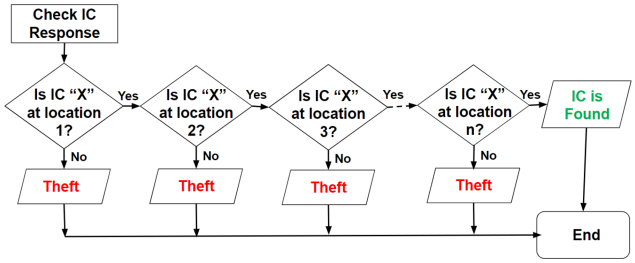


Fig. 6. Proposed approach to detect theft counterfeit ICs.

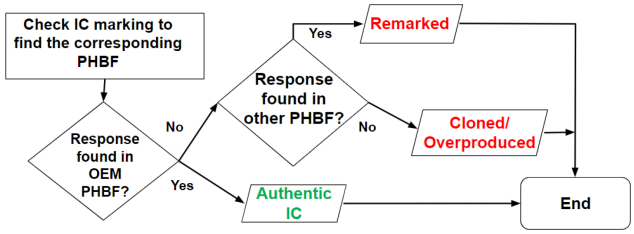


Fig. 7. Proposed approach to detect remarked and cloned/ overproduced counterfeit ICs.

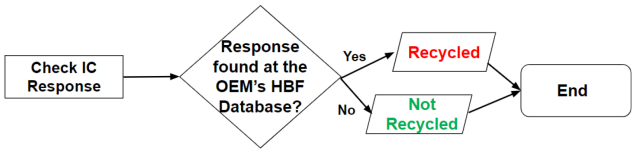


Fig. 8. Proposed approach to detect recycled counterfeit ICs.

days. If any of the locations fail to authenticate the response, we can consider that particular IC being a stolen chip (see Figure 6) within the supply chain.

The second and the third type in the list of queries initiates at the OEM-end. The process is illustrated in Figure 7. Whenever a chip is manufactured, it gets stored at the OEM's PHBF database. Since there are separate PHBFs that stores information related to each of the dies or wafers, for the second and third type of query, we first inspect the markings to find the corresponding PHBF. If our query IC response finds a match in the database, we can say that our query IC response is manufactured at the OEM-end and the IC is authentic. On the other hand, if the IC response could not be found at the OEM-end database, we proceed our search to other location's PHBFs. Now, if there exist any IC responses that are not in the OEM database, but in any of the other location's databases, it is possible that the query IC has entered the supply chain from a different entity other than the OEM and could be identified as a remarked type of counterfeit IC. Cloned ICs, on the other hand, has the same markings as the genuine ICs. However, unlike the genuine ones, the responses of cloned ICs have not been registered/stored in any PHBF. Benefitting from this fact, if our search does not match any responses from separate locations of PHBFs, we can conclude that the IC is cloned or overproduced.

The fourth in our list is the simplest among all the queries that are listed above. To detect the recycled type of counterfeit, we only check for responses that are stored at the OEM-end's HBF. If the IC was previously sold to an end user, we find a match. Thus, it can be concluded that the IC is recycled. The query process for this type of query is illustrated in Figure 8.

5 Experimental Results

In this section, we present a comprehensive analysis of the experimental results. The subsequent discussions encapsulate the intricacies of our experimental setup, the metrics utilized for evaluation, and a detailed examination of the authentication performance. Through a systematic exploration of these key components, we aim to provide a thorough understanding of the effectiveness and reliability of our proposed approach within the PHBF framework.

5.1 Experimental Setup

We conducted two types of experiments in this article: (1) authentication performance verification of the HBFs, and (2) authentication using PHBFs. For both the experiments, **Ring-oscillator (RO)**-based PUF with MicroBlaze microprocessor have been deployed in the Basys-3 FPGA development board to generate the CRPs with environmental variations. A RO PUF [11] is a delay-based PUF that generates unique responses based on the given challenges and the frequency variations of the ROs. The RO frequency varies due to the process variation of the silicon die (V) and the environmental variations (E). In a supply chain, ICs travel through different entities that are in different environmental conditions. This environmental variation can create bit flips in responses of the PUFs, which will may cause **False Negative (FN)** identifications. Our proposed HBF-based framework can identify **True Positive (TP)** queries even with noisy query (response) data. So, we have generated PUF 32-bit responses in different temperatures to impose noise and test our proposed model. A total of 5,000 responses have been collected to create an enrollment dataset. Furthermore, two sets of 1,000 noisy responses each with 32-bit length have been generated at $\approx 75^{\circ}C$ and $\approx 100^{\circ}C$ temperatures each. Due to the increase of temperatures from $\approx 26^{\circ}C$ (room temperature) to $\approx 75^{\circ}C$ and $\approx 100^{\circ}C$, the reliability of PUF has decreased from 98.51% to 98.1% and 97.54% respectively. Also, the response uniformity increased from 53.3% to 60.1% and 60.4% as well. Additionally, to check the robustness of our framework, from the original 5,000 enrollment data, we randomly picked 1,000 PUF responses. Then, within these 1,000 PUF responses, we flipped 2% bits for the first 400 instances, 5% for the next 300 instances, and 7% for the final 300 instances causing them to have synthetically generated noise.

We maintained two sets of these synthetic noisy data. Since it's not possible to mimic the original noise distribution in a manual way, in one of the sets, the synthetic noise was distributed uniformly across the 32-bit responses (see Figure 9(a)). Uniform noise can be defined as a type of noise that is evenly distributed throughout the entire range of response values in a PUF. This uniform distribution implies that any given response value has an equal likelihood of being influenced by the noise. In the case of the other set, the noise was put in together in a clustered way (see Figure 9(b)). Clustered noise is characterized by a non-uniform distribution that tends to concentrate around certain regions or values in the response space of the PUF. This suggests that certain values or regions are more susceptible to noise than others, which can be attributed to specific environmental or manufacturing factors that affect particular areas of the response space.

Until this point, we have had four sets of 1,000 noisy data. Two sets from the original PUF data were generated at 75°C and 100°C, and the other two sets of synthetic noisy data. Using these four sets of noisy data, we created four sets of query data (see Figure 10). Each of the query datasets had 3,000 PUF responses. The first 1,000 responses were directly taken from the enrolled dataset. These responses are genuine and serve as the baseline for identifying valid, enrolled PUF responses. The second set of 1,000 responses was derived from the original dataset of 5,000 responses. The second set of 1,000 responses, in order to introduce variability, were randomly selected from the 5,000 and then subjected to controlled noise injection as mentioned above. This noise included alterations, mimicking real-world conditions where responses might deviate from the enrolled

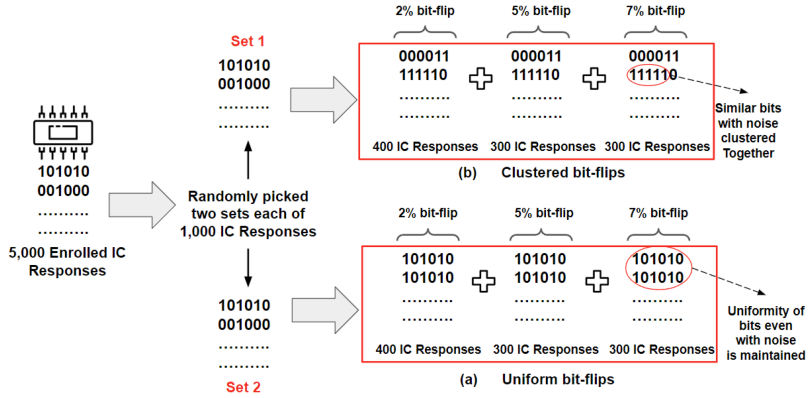


Fig. 9. Incorporating uniform and clustered noise into random PUF response samples.

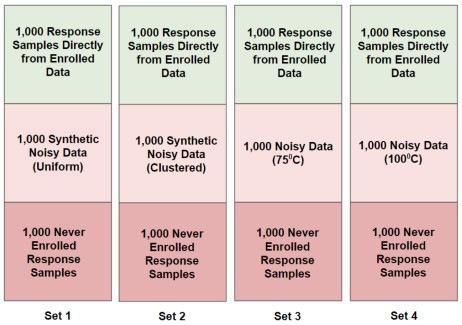


Fig. 10. Design of four different query sets for performing membership query processing.

data. The intention behind this segment was to simulate scenarios where the PUF response is still recognized as legitimate but exhibits some level of deviation from the original enrolled data. The final set of 1,000 responses were crafted to simulate completely unenrolled, or non-genuine, PUF responses which were generated synthetically. This was achieved by taking the original enrolled responses and flipping approximately 50% of the bits. This substantial modification ensured that these responses would not match the enrolled dataset [6] and would be distinctly recognized as unenrolled. The purpose of this segment was to test the system's robustness in identifying and rejecting responses that are sufficiently different from the enrolled data, and are completely new, unique and reproducible unenrolled PUF.

5.2 Evaluation Metric

Using four of the prepared query sets, we performed multiple PUF response authentication experiments on the HBF and the PHBF. Results are represented using the **Receiver Operating Characteristics (ROC)** curve. The ROC curve is a graphical representation of the performance of a binary classifier system as its discrimination threshold is varied. It plots the TP rate against the FP rate at different thresholds, allowing for an assessment of how well different models are able to distinguish between classes. In our case, the ROC curve shows how the **False Acceptance Rate (FAR)** and **False Rejection Rate (FRR)** rates are behaving when the threshold for authentication is set to different values. The FAR and FRR score is calculated using the equations $FAR = \frac{\text{Number of False Acceptance}}{\text{Total Number of Identification Attempts}}$ and $FRR = \frac{\text{Number of False Rejection}}{\text{Total Number of Identification Attempts}}$ respectively.

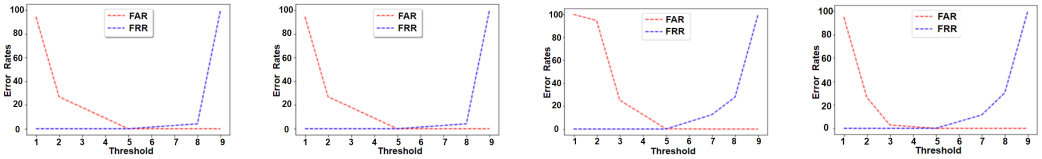


Fig. 11. Performance evaluation of HBF using ROC curve with four query datasets and demonstrating the optimal number of BF required to authenticate ICs: (a) dataset containing real noise (75℃), (b) dataset containing real noise (100℃), (c) dataset containing synthetic noise (clustered), and (d) dataset containing synthetic noise (uniform).

Table 4. Authentication Performance of HBF and PHBF at $th = 5$

Query Samples = 3,000	**Positive**	**Negative**
Positive	2,000	0
Negative	0	1,000

Since we had similar results for both the stages—(a) authentication performance verification of HBF and (b) authentication performance of PHBF, a common table of the results is shown.

Specificity measures how accurately it can classify negatives. A perfect test would have 100% sensitivity and 100% specificity—it would be able to detect all cases with no FP whatsoever. We can also understand the same thing using **Equal Error Rate (EER)**, where $FAR = FRR$. We evaluated our framework using the confusion matrix and presented our results in terms of TP, FP, True Negative, and FN.

5.3 Experimental Results on Authentication Performance

We conducted our experiments in two stages as mentioned in the earlier section: (1) authentication performance verification of the HBFs, and (2) authentication performance of the PHBF. Since PHBF is constructed using HBFs, it was important to verify the performance of the HBFs. Using our HBF architecture, after the enrollment dataset is enrolled, we used the four query sets (see Figure 10) to perform the query operations. Using the concepts from Section 4, an HBF of level 3 and BF count of 9 was set for the enrollment of 5,000 response data each of 32 bits. The threshold (th) was set starting from 1 to 9 and FAR and FRR data points were calculated for evaluating HBF (see Figure 11). Ideally, a threshold should be chosen depending on external conditions (e.g., extreme heat or cold weather). In our experiment, quite similar to the synthetically generated clustered noise (see Figure 11(c)), in the uniformly distributed noisy data (see Figure 11(d)), the FAR and FRR follow the same trend. The robustness of our system is evident when responses with original noise data (initiated at 75°C) (see Figure 11(a)) and responses with original noise data (initiated at 100°C) (see Figure 11(b)) provide us similar trend as the synthetic noisy data experiment. At each of the experiments, regardless of the query sets that were used, the threshold parameter, $th = 5$ is where we get our EER of 0. This means, at $th = 5$, we get no FPs or FNs. The confusion matrix (Table 4) shows our query results at $th = 5$.

From Table 4, we can see that our framework correctly identified the 2,000 PUF responses accurately even though 1,000 amongst them were noisy.

Experiment on the Proposed Queries. As we proceed to the next stage, using PHBF, we conducted experiments on the proposed set of queries mentioned in Section 4.2. As our query dataset, we

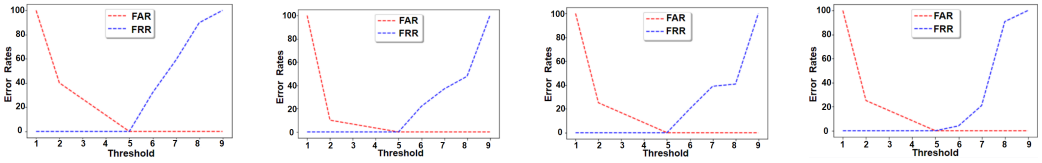


Fig. 12. Authentication performance evaluation of PHBF using ROC curve for detecting different types of counterfeits and demonstrating the optimal number of BF required to authenticate ICs: (a) theft, (b) clones/overproduced, (c) remarked, and (d) recycled type of counterfeits (see details about the queries in Table 3).

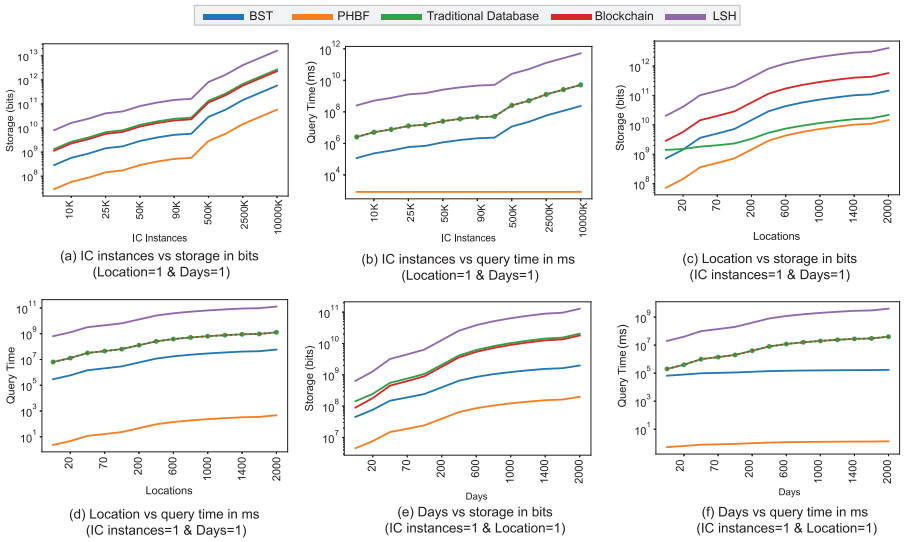


Fig. 13. Comparison of storage and query time efficiency in different data structures: (a), (c), and (e) demonstrate the storage requirement when no. of ICs, no. of locations, and no. of days vary (respectively); and (b), (d), and (f) demonstrate the query processing time when no. of ICs, no. of locations, and no. of days vary (respectively).

use set 4 from Figure 10 (original noisy response initiated at 100°C). After experimenting on all of the query sets that were proposed, we get EER = 0 at the threshold, $th = 5$ (see Figure 12). For this experiment as well, we got the same confusion matrix as shown in Table 4. In this experiment also, the PHBF correctly identifies the enrolled data (2,000) and the never enrolled (1,000) PUF responses. Regardless of the query sets that were used to conduct our experiments, it is evident that if the parameters of our proposed framework are tuned accordingly, it is possible to have a 100% detection rate of ICs using PUF responses even when the responses are prone to change.

6 Scalability Analysis

In this section, we analyze the performance of our proposed PHBF with the relevant existing frameworks (see Figure 13) in terms of their storage and query processing time. Our data structure's key feature is its time and storage efficiency. To clearly demonstrate its scalability, we conducted a comprehensive analysis of time and storage complexity, followed by a series of experiments using values derived from this analysis. We tested the structure's performance across a wide range of scenarios, varying the number of locations up to 2,000, the number of items up to 10 million, and

Table 5. Overhead Comparison between Proposed PHBF and Other Approaches

Overhead		Proposed PHBF	BST	Blockchain	LSH	Traditional Database
Storage	Total	$L \cdot (2^{\log_2(T+1)} - 1) \cdot N \cdot m$	$L \cdot n \cdot (2^{\log_2(T+1)} - 1) \cdot l \cdot q$	$n \cdot L \cdot T \cdot P$	$n \cdot L \cdot T \cdot H \cdot d$	$n \cdot q \cdot t \cdot ((T \cdot d) \cdot (L \cdot l))$
	Worst Case	$O(n \cdot L \cdot T)$	$O(L \cdot n \cdot T)$	$O(n \cdot L \cdot T)$	$O(n \cdot L \cdot T \cdot H)$	$O(n \cdot L \cdot T)$
Time	Total	$L \cdot ((\log_2(T) + 1) \cdot N \cdot k$	$n \cdot L \cdot (\log_2(T) + 1)$	$n \cdot L \cdot T$	$n \cdot L \cdot T \cdot H$	$n \cdot L \cdot T$
	Worst Case	$O(L \cdot (\log_2(T)))$	$O(n \cdot L \cdot (\log_2(T)))$	$O(n \cdot L \cdot T)$	$O(n \cdot L \cdot T)$	$O(n \cdot L \cdot T)$

the duration up to 2,000 days (graphs shown in Figure 13 are in log scale). Comparison is done using the existing data structures and demonstrated that the PHBF does better in terms of scalability in terms of both time and storage. The considered structures are BST [4], Traditional Database (relational database) [8], Blockchain [13], and **Locality-sensitive Hashing (LSH)** [7]. Note that, all the expressions for the complete representation as well as the worst-case complexity are listed in Table 5.

6.1 Storage Efficiency

For analyzing the storage efficiency, we first determine the expression for the total storage requirement for each of the data structures considering n ICs visiting L locations for T days. Next, using the same expression and ignoring the lower order and constant terms, we deduce the worst-case complexity analysis. Finally, considering the same supply chain model, we conduct some simulation-based experiments with the original total storage requirement equation by varying n, L, and T to observe the scalability of each of the structures.

PHBF Total Storage. To initiate determining the total storage, we focus on assessing the storage requirements for n ICs starting at a singular location. Within each location of the supply chain, a PHBF is appointed to manage the enrollment of n ICs. The storage requirement for this PHBF is defined in terms of the "bits required to store the IDs for n ICs" at that specific location. Consequently, we compute the storage requirement for a single PHBF and then multiply it by L to account for the total L locations and the corresponding appointments of L PHBFs. Next, we delve into understanding how the storage requirement for a single PHBF is determined. As explained in the methodology, the foundations of a PHBF are HBFs, with each node of the PHBF housing an HBF. Given that there are $\log_2(T + 1)$ nodes in a PHBF, the same number of HBFs are present within a PHBF. Additionally, it's important to note that an HBF is also a hierarchical structure where each node incorporates a BF. Thus, an HBF consists of N BFs organized in a hierarchical manner [17]. When dissecting the structure of one BF, we find m bits, where m is contingent upon the number of IC inputs denoted as n. Total storage requirement for n ICs in the supply chain can be computed by:

Step 1: The total number of levels in a PHBF is calculated, which is equal to, $\log_2(T + 1)$
Step 2: Next, the total number of nodes in a PHBF is determined, which is equal to, $2^{\log_2(T+1)} - 1$.
Step 3: Storage for a single PHBF is defined, which is equal to, No. of nodes × Size of a single HBF $= (2^{\log_2(T+1)} - 1) \times N \times m$.
Step 4: The resultant value is further multiplied by the total count of L locations, which is equal to, No. of Locations × No. of nodes × No. of BFs in an HBF × Size of a single BF $= L \times (2^{\log_2(T+1)} - 1) \times N \times m$.

Thus, total storage (in bits) required for enrolling all the ICs for all the days in all the locations across the supply chain can be mentioned as $Storage_{phbf}$ and can be expressed as:

$$Storage_{phbf} = L \cdot (2^{\log_2(T+1)} - 1) \cdot N \cdot m$$

$$\implies L \cdot (2^{T/2} - 1) \cdot N \cdot \lceil n \cdot log_2(p)/log_2(1/2^{log_2(2)}) \rceil$$

$$\implies L \cdot (2^{T/2} - 1) \cdot N \cdot \lceil ((n \cdot log_2(p))/0.81) \rceil . \quad (5)$$

Therefore, The worst-case storage complexity is $O(n \cdot L \cdot T)$.

BST. Similar to PHBF, in order to calculate the total storage for BSTs can be done by following these steps:

Step 1: The total no. of nodes in BST is defined, which is equal to, $2^{\log_2(T+1)} - 1$.
Step 2: Within the nodes, ICs of l response length of q bits are kept, which is equal to, Total no. of nodes × Response length × Response size = $(2^{\log_2(T+1)} - 1) \times l \times q$.
Step 3: Since there are n number of ICs, the resultant value will be multiplied, which is equal to, Total no. of ICs × Total no. of nodes × Response length × Response size = $n \times (2^{\log_2(T+1)} - 1) \times l \times q$.
Step 4: Finally, since there are L number of locations, which is equal to, No. of locations × Total no. of nodes × Response length × Response size = $L \times n \times (2^{\log_2(T+1)} - 1) \times l \times q$. Thus, we can write total storage as:

$$Storage_{bst} = L \cdot n \cdot (2^{\log_2(T+1)} - 1) \cdot l \cdot q$$

$$\implies L \cdot n \cdot (2^{\log_2(T+1)} - 1) \cdot l. \tag{6}$$

Therefore, The worst-case storage complexity is $O(L \cdot n \cdot T)$.

Traditional Database. Relational databases are commonly used thus we consider that as traditional database. A critical feature of relational databases is the concept of a primary key, which ensures the uniqueness and integrity of data within a table. Relationships between tables are established through foreign keys, allowing for complex data structures. Access and manipulation of data in relational databases are carried out using SQL. In our supply chain setting, we had multiple variable dependencies while calculating the complexity for traditional databases. We enroll the n number of ICs row-wise in a T number of column database to organize the ICs. By this, we can understand the enrollment time of each ICs. Now, in order to calculate the total storage, we considered the following steps:

Step 1: For a single database in a single location, the total storage (in bits) can be calculated for n ICs, which is equal to, Total no. of ICs × Bits needed to store one character = $n \times l$.
Step 2: Similarly, to store enrollment time for each of the n ICs, t is calculated for the total of T times, which is equal to, Length of time string represented × Total time = $t \times T$.
Step 3: Thus, within the database, the total space needed can be calculated for n ICs enrolled in t time, which is equal to, Bits ICs (in bits) × Bits needed to store one character = $(n \times l) \times (T \times t)$.
Step 4: Finally, since there are L number of locations with each of them having the name of string d length, the same structure that is mentioned above could be copied to every location, which is equal to, Bits ICs × Bits needed to store one character × Total no. of locations × Length of the location name's string representation= $(n \times l) \times (T \times t) \times (L \times d)$. As a result, we can compute the total storage as:

$$Storage_{traditional} = (n \cdot l) \cdot (T \cdot t) \cdot (L \cdot d). \tag{7}$$

Therefore, the worst-case storage complexity is $O(n \cdot L \cdot T)$.

Blockchain. The structure of a blockchain is the foundational architecture that defines how this distributed ledger technology is organized and functions. At its core, a blockchain is a sequence of blocks, with each block containing a collection of transactions or data. These blocks are interconnected in a linear, chronological order, creating an unbroken chain. Each block is divided into two primary components: the header and the body. The immutability of the blockchain is a

defining feature of its structure. Once data is added to a block and confirmed by the network, it becomes exceedingly difficult to alter or delete. This characteristic ensures the trustworthiness and reliability of the ledger. From a technical perspective, blockchain systems employ a combination of cryptographic methods, consensus algorithms, and distributed network protocols. These technical components work in union leading to a complex and intricate technological landscape. In our case:

Step 1: Due to the linear nature of the structure of the blockchain, the total storage is calculated by multiplying the variables n, L and T, which is equal to, Total no. of ICs × Total no. of locations × Total time = $n \times L \times T$.

Step 2: Moreover, blockchain-based IC management infrastructure requires a granular evaluation of the number of transactions needed to oversee a single IC, denoted as P, which is equal to, Total no. of ICs × Total no. of locations × Total time × Bits needed for each transaction = $n \times L \times T \times P$:

$$Storage_{blockchain} = n \cdot L \cdot T \cdot P. \tag{8}$$

Therefore, the worst-case storage complexity for blockchain is: $O(n \cdot L \cdot T)$.

LSH. Quite similar to the previous variables defined on the other data structures, Total storage for LSH can be calculated when:

Step 1: The number of bits required for all the hash tables for all ICs, denoted as H is assessed and length of location name (d) for all hash tables for all ICs are then multiplied, which is equal to, No. of hash tables × length of location name = $H \times d$.

Step 2: Calculate the total storage required for managing ICs using the parameters n, L and T just like we did for blockchain, which is equal to, Total no. of ICs × Total no. of locations × Total time × No. of hash tables × length of location name = $n \times L \times T \times H \times d$. Therefore, the calculation can be expressed as:

$$Storage_{lsh} = n \cdot L \cdot T \cdot H \cdot d. \tag{9}$$

Therefore, the worst-case storage complexity for LSH is: $O(n \cdot L \cdot T \cdot H)$.

Scalability Analysis in Terms of Storage Requirement. The complexity expressions of different data structures are summarized in Table 5. By analyzing the storage complexity of different data structures theoretically, it is expected that the storage requirement for PHBF does not increase rapidly as it does for other structures. To offer better visualization of our interpretation, we ran multiple simulation-based experiments where we vary n, L, and T in different combinations and present the storage requirement. As shown in Figure 13(a), (c), and (e), the PHBF is the most storage-efficient one.

6.2 Query Time Efficiency

Similar to the previous subsection, we did the expression derivation for total query processing time, worst-case time complexity, and simulation-based experiments for time efficiency observation. The total query processing time refers to the number of operations. As a query, we considered what will be the total time requirement if an IC is looked up for all the days throughout all the locations.

PHBF. When assessing the time complexity of this system, we find that it relies on several factors to gauge its efficiency. Each of these elements plays a distinct role in determining the overall time complexity, much like the storage considerations previously discussed. Just as the storage requirements varied with parameters such as n (number of ICs), L (number of locations), T (Total time), and N (number of BFs in an HBF), the time complexity analysis takes into account various parameters to evaluate the efficiency of query operations. Moreover:

Step 1: The total number of levels are considered in which the searching is done for all the locations which is equal to, Total no. of locations × Total no. of levels = $L \times ((\log_2(T) + 1)$.
Step 2: In all the levels of the PHBF, there are N BFs the search is required, which is equal to, Total no. of locations × Total no. of levels × No. of BFs in an HBF = $L \times ((\log_2(T) + 1) \times N$.
Step 3: Before the data is enrolled at the PHBF, it is hashed using k hash functions. Thus, the hashing time is considered, which is equal to, Total no. of locations × Total no. of levels × No. of BFs in an HBF × No. of hash functions = $L \times ((\log_2(T) + 1) \times N \times k$. Finally, we get the following expression:

$$QueryTime_{PHBF} = L \cdot ((\log_2(T) + 1) \cdot N \cdot k. \tag{10}$$

Therefore, the worst-case time complexity is $O(L \cdot log_2(T))$.

BST. The total query time in a BST structure can be calculated by doing the following:

Step 1: The total query time of the BST can be calculated when we try to find all the ICs in all the locations, which is equal to, Total no. of ICs × Total no. of locations = $n \times L$.
Step 2: Thus, to quantify the time complexity concerning there are multiple levels in a BST, the following formula can be employed, which is equal to, Total no. of ICs × Total no. of locations × Total no. of levels = $n \times L \times ((\log_2(T) + 1)$. Thus, the formula can be expressed as:

$$QueryTime_{BST} = n \cdot L \cdot (log_2(T) + 1). \tag{11}$$

Therefore, the time complexity of BST is: $O(n \cdot L \cdot (\log_2(T)))$.

Traditional Database. The total query time is determined by accessing the total number of IC inputs, searching through the number of locations, and calculating the total time required, i.e., variables n, L, and T. This results in the following:

$$QueryTime_{traditional} = n \cdot L \cdot T. \tag{12}$$

Therefore, the worst-case time complexity is: $O(n \cdot L \cdot T)$.

Blockchain. Quite similar to the traditional database, in the context of blockchain technology, the complexity analysis calculation is achieved through the multiplication of the number of ICs (n), the number of locations (L) and total time (T) and resulting in:

$$QueryTime_{blockchain} = n \cdot L \cdot T. \tag{13}$$

Therefore, the worst-case time complexity is $O(n \cdot L \cdot T)$.

LSH. When assessing the total query time in the context of LSH, a structured analysis is essential to understand the implications. The calculation is based on the following enumerate,

Step 1: The total query time for all ICs is calculated by multiplying the factors n, L, T just like the ones that are calculated for blockchain and traditional database, which is equal to, Total no. of ICs × Total no. of Locations × Total time = $n \times L \times T$.
Step 2: Moreover, there is one additional parameter H is factored for the multiple hash tables, which is equal to, Total no. of BFs in an HBF × Total no. of Locations × Total time × Total no. of hash tables = $n \times L \times T \times H$:

$$QueryTime_{LSH} = n \cdot L \cdot T \cdot H. \tag{14}$$

Therefore, the worst-case time complexity is $O(n \cdot L \cdot T)$.

In accordance with our methodology, analogous to the approach employed for assessing storage complexity, we conducted a series of multiple simulations aimed at evaluating query complexity. The outcomes of these simulations are presented in Figure 13(b), (d), and (f). Our findings collectively

indicate that the PHBF emerges as the most temporally efficient solution among the considered alternatives.

Time Requirement for Proposed Query Sets. To detect theft counterfeits, our proposed method comprises the querying into every location (see Figure 6). Quite similar to the theft counterfeit, if the IC cannot be found at the OEM-end, to detect the cloned and overproduced ICs (see Figure 7), query needs to be performed in remaining location PHBFs as well. Thus, the query complexity to detect theft, cloned and overproduced IC can be written as, $QueryTime = L \cdot (\log_2(T))$.

The time complexity of detecting recycled ICs (see Figure 8) is same as query time complexity in an HBF. We calculate the query time using, number of BFs in an HBF × number of hash functions. Mathematically, the equation can be represented as, $QueryTime = N \cdot K$.

7 Conclusion and Future Work

We have presented a novel data structure PHBF which is a fast, storage-efficient and noise-tolerant framework to offer a scalable and robust authentication and tracking of ICs. We discussed the design principles, the flow of IC enrollment and query process for detecting four major types of counterfeits. We also analyzed the authentication performance and scalability in terms of storage and time efficiency in both theoretically and experimentally. Furthermore, we compared our results with four existing state-of-the-art approaches and observed that our proposed PHBF performs significantly better in terms of authentication performance and scalability properties.

In our future work, we will focus on several key areas to further enhance the PHBF framework. These include the integration of a deletion and insertion property within the PHBF structure, allowing for more flexible and dynamic data management by dynamically adding or removing items. This will enable our proposed framework to be a Dynamic PHBF framework. Furthermore, we intend to do a thorough security analysis to assess the framework's resilience against potential adversarial models which will ensure that it can withstand various types of attacks. Finally, we will explore improvements to the overall framework to address any identified security or privacy issues.

References

[1] Abdulrahman Alaql, Tamzidul Hoque, Domenic Forte, and Swarup Bhunia. 2019. Quality obfuscation for error-tolerant and adaptive hardware IP protection. In *Proceedings of the 2019 IEEE 37th VLSI Test Symposium (VTS)*. IEEE, 1–6.

[2] Leonardo Aniello, Basel Halak, Peter Chai, Riddhi Dhall, Mircea Mihalea, and Adrian Wilczynski. 2021. Anti-BlUFf: Towards counterfeit mitigation in IC supply chains using blockchain and PUF. *International Journal of Information Security* 20, 3 (2021), 445–460.

[3] Burton H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM* 13, 7 (1970), 422–426.

[4] Nathan G. Bronson, Jared Casper, Hassan Chafi, and Kunle Olukotun. 2010. A practical concurrent binary search tree. *ACM SIGPLAN Notices* 45, 5 (2010), 257–268.

[5] Wenjie Che, Fareena Saqib, and Jim Plusquellic. 2015. PUF-based authentication. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '15)*. IEEE Press, 337–344.

[6] Yansong Gao, Said F. Al-Sarawi, and Derek Abbott. 2020. Physical unclonable functions. *Nature Electronics* 3, 2 (2020), 81–91.

[7] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity search in high dimensions via hashing. In *Proceedings of the 25th VLDB Conference*, Vol. 99, 518–529.

[8] Jan L. Harrington. 2016. *Relational Database Design and Implementation*. Morgan Kaufmann, Cambridge, MA.

[9] Md Nazmul Islam and Sandip Kundu. 2019. Enabling IC traceability via blockchain pegged to embedded PUF. *ACM Transactions on Design Automation of Electronic Systems* 24, 3 (2019), 1–23.

[10] Md Nazmul Islam, Vinay C. Patii, and Sandip Kundu. 2018. On IC traceability via blockchain. In *Proceedings of the 2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. IEEE, 1–4.

[11] Abhranil Maiti and Patrick Schaumont. 2011. Improved ring oscillator PUF: An FPGA-friendly secure primitive. *Journal of Cryptology* 24 (2011), 375–397.

[12] Judy Meiksin. 2022. US "CHIPS and science act" gives semiconductor R&D and industry a multibillion dollar boost. *MRS Bulletin* 47 (2022), 890–892.

[13] Michael Nofer, Peter Gomber, Oliver Hinz, and Dirk Schiereck. 2017. Blockchain. *Business & Information Systems Engineering* 59, 3 (2017), 183–187.
[14] Landon Curt Noll. 1994. FNV HASH. Retrieved February 23, 2023 from http://www.isthe.com/chongo/tech/comp/fnv/
[15] Yanqing Peng, Jinwei Guo, Feifei Li, Weining Qian, and Aoying Zhou. 2018. Persistent bloom filter: Membership testing for the entire history. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 1037–1052.
[16] S. S. Rekha, K. Suraj, and K. Sudeendra Kumar. 2021. A holistic blockchain based IC traceability technique. In *Proceedings of the 2021 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS)*. IEEE, 307–310.
[17] Sumaiya Shomaji, Fatemeh Ganji, Damon Woodard, and Domenic Forte. 2019. Hierarchical bloom filter framework for security, space-efficiency, and rapid query handling in biometric systems. In *Proceedings of the 2019 IEEE 10th International Conference on Biometrics Theory, Applications and Systems (BTAS)*. IEEE, 1–8.
[18] Nidish Vashistha, Muhammad Monir Hossain, Md Rakib Shahriar, Farimah Farahmandi, Fahim Rahman, and Mark M. Tehranipoor. 2021. eChain: A Blockchain-Enabled ecosystem for electronic device authenticity verification. *IEEE Transactions on Consumer Electronics* 68, 1 (2021), 23–37.
[19] Xiaolin Xu, Fahim Rahman, Bicky Shakya, Apostol Vassilev, Domenic Forte, and Mark Tehranipoor. 2019. Electronics supply chain integrity enabled by blockchain. *ACM Transactions on Design Automation of Electronic Systems* 24, 3 (2019), 1–25.

Received 11 November 2023; revised 30 August 2024; accepted 8 July 2025

Impedance Leakage Vulnerability and Its Utilization in Reverse-Engineering Embedded Software

MD SADIK AWAL and MD TAUHIDUR RAHMAN, Electrical and Computer Engineering, Florida International University, Miami, Florida, USA

Discovering new vulnerabilities and implementing security and privacy measures are important to protect systems and data against physical attacks. One such vulnerability is impedance, an inherent property of a device that can be exploited to leak information through an unintended side-channel, thereby posing significant security and privacy risks. Unlike traditional vulnerabilities, impedance is often overlooked or narrowly explored, as it is typically treated as a fixed value at a specific frequency in research and design endeavors, leaving its potential for information leakage largely unexplored. This article demonstrates that the impedance of an embedded device is not constant and directly relates to the programs executed on the device. We define this phenomenon as *impedance leakage* and use this as a side-channel to extract software instructions from protected memory. Our experiment on the ATmega328P microcontroller and the Artix 7 FPGA indicates that the impedance side-channel can detect software instructions with 96.1% and 92.6% accuracy, respectively. Furthermore, we explore the dual nature of the impedance side-channel, highlighting the potential for beneficial purposes and the associated risk of intellectual property theft.

CCS Concepts: • **Security and privacy** → *Embedded systems security*; *Security in hardware*; *Side-channel analysis and countermeasures*;

Additional Key Words and Phrases: Hardware security, impedance leakage, switching activity, instruction reverse engineering

ACM Reference format:
Md Sadik Awal and Md Tauhidur Rahman. 2025. *Impedance Leakage* Vulnerability and Its Utilization in Reverse-Engineering Embedded Software. *ACM J. Emerg. Technol. Comput. Syst.* 21, 4, Article 12 (October 2025), 20 pages.
https://doi.org/10.1145/3764931

1 Introduction

Integrated circuits and systems have become essential components in various domains, managing sensitive data and attracting adversaries aiming to extract unauthorized access or information [43]. Intriguingly, the internal operations of these systems give rise to physical side-channel signals—subtle traces of information that can be exploited to obtain sensitive information. In this context, physical **side-channel analysis (SCA)** arises as a potent technique based on the physical properties of a system [38]. The phenomenon of physical side-channel leakages resulting from the

Authors' Contact Information: Md Sadik Awal (corresponding author), Electrical and Computer Engineering, Florida International University, Miami, Florida, USA; e-mail: mawal003@fiu.edu; Md Tauhidur Rahman, Electrical and Computer Engineering, Florida International University, Miami, Florida, USA; e-mail: mdtrahma@fiu.edu.

ACM 1550-4840/2025/10-ART12
https://doi.org/10.1145/3764931

computation and storage operations on integrated circuits reveals itself through measurable quantities such as power consumption, **electromagnetic (EM)** radiation, and even thermal radiation. The attackers exploit these quantities to compromise the security of cryptographic implementations [38]. Although numerous countermeasures have been proposed and developed to limit SCA-based attacks, SCA has demonstrated their effectiveness despite standard security measures [12, 44, 46]. The persistence and adaptability of side-channel vulnerabilities continue to pose new challenges [34, 46].

Contrary to conventional security measures, which often focus on mitigating vulnerabilities, SCA presents an intriguing perspective: It can be employed to enhance hardware security [14, 22, 23, 37]. By integrating side-channel information into device design, intrusion detectors can be developed to monitor the side channels of a device for indications of an attack. These detectors can identify deviations from expected patterns and initiate preventive measures if an attack is detected [37]. Moreover, external detectors can discreetly observe device activity without relying on its inputs and outputs, thus foiling an attacker's attempts to escape detection by controlling those elements.

While many vulnerabilities resulting from the physical properties of digital systems have been explored, one often overlooked characteristic is impedance, a fundamental property of electrical circuits [6]. Despite its seemingly innocuous nature, impedance can act as an unintended side-channel, posing significant security and privacy risks. Traditionally, impedance is considered a fixed value at a specific frequency [6, 30], which has limited its exploration. In contrast to the prevailing assumption, our research studies how impedance variations can leak sensitive information from embedded systems. We define this phenomenon as *impedance leakage* and demonstrate that it can reveal details about software execution [6, 8] and even the contents of emerging memories that encode data through impedance variations [7]. In our prior work, we demonstrated that different instruction types can be distinguished using impedance profiles [8]. Subsequent research, such as that conducted by [27], follows our foundational contributions in recognizing impedance as a side-channel to recover AES encryption key. In this article, we take a step further by showing that individual instructions, not just instruction types, can be uniquely identified based on their impedance signatures. This finer level of granularity enables more precise reverse engineering and opens new possibilities for both validating and monitoring software integrity. To support this advancement, we introduce a refined two-stage frequency selection framework that isolates instruction-sensitive impedance points with higher discriminative power. In addition, we develop a gate-level mathematical formulation that explains how logic switching activity contributes to impedance variation. This gate-level mathematical framework establishes that impedance variations are intrinsically linked to the logic states of digital circuits, providing a theoretical foundation for the instruction-dependent impedance leakage.

Building on these insights, this article investigates the feasibility of using impedance leakage to reverse engineering executed software instructions. Disassembling software instructions provides insights into the internal behavior of an embedded device, making it a potential defense against a wide range of software-based attacks, e.g., software modification-based attacks. We observe that the impedance of an embedded device changes when it executes different software instructions. Leveraging this phenomenon, we investigate the feasibility of using runtime impedance variations across selected frequency channels as an emerging side-channel for instruction disassembly. Thus, this work seeks to answer the fundamental question: "Does device impedance leak information of the executed software instructions and emerge as a promising side-channel?"—or in other words: *Can we use the device runtime impedance to reverse engineering executed software instructions*? The major contributions of this work are summarized as follows:

—We explore a novel runtime impedance side-channel arising from instruction-dependent switching activity.
—We develop a gate-level and system-level mathematical framework that models how logic state transitions influence impedance, providing a theoretical basis for impedance leakage.
—We present a frequency selection framework that identifies instruction-sensitive impedance points with high discriminative potential.
—We demonstrate and validate the existence of impedance leakage by reverse-engineering individual software instructions across two different embedded platforms using multiple **machine learning (ML)** models, achieving high classification accuracy.

The remaining sections of the article are organized as follows. Section 2 briefly addresses existing side channels along with impedance as a side-channel. Section 3 discusses the research motivation. The experimental setup of this study has been discussed in Section 4, which includes the hardware setup, software setup, and impedance signal measurement. In Section 5, the analysis of the signals and the classification results are presented. Section 6 discusses the related works. The conclusion of the work is drawn in Section 7.

2 Background

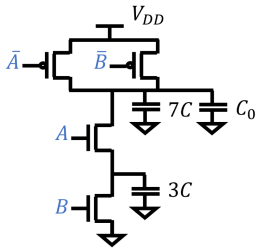
The security and privacy of digital systems are not solely dependent on the efficacy of mathematically secure algorithms and protocols, as the physical properties of a system can unintentionally leak internal information. These unintended physical characteristics, known as physical side channels, can be exploited by attackers to gain access to sensitive data and assets. Two widely studied physical side channels are the power SCA and EM SCA.

Power SCA involves analyzing variations in power consumption during system operations. An attacker can deduce sensitive information like cryptographic keys by measuring the power consumption of the target component, such as a processor. Differential power analysis is commonly used in this context to detect subtle changes in power consumption, providing insights into data-dependent variations that may otherwise be imperceptible. Alternatively, EM SCA relies on capturing the EM radiation emanating from a device during its functioning. As the device executes its operations, the dynamic changes in the current flow within its components give rise to distinct EM waves. Through the careful analysis of these EM leakages, attackers can glean valuable information about the events occurring throughout each clock cycle, allowing them to potentially infer internal processes and sensitive data. Despite their effectiveness, both power and EM SCAs present practical limitations. Power analysis often requires hardware modification (e.g., inserting a shunt resistor), while EM analysis demands specialized equipment like near-field probes and advanced signal processing. Additionally, devices with aggressive power filtering or low power footprints may exhibit weak or masked side-channel signatures.

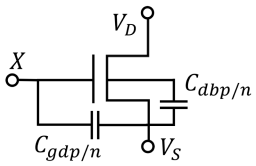
Despite their effectiveness, both power and EM SCAs present limitations. Power analysis often requires hardware modification (e.g., inserting a shunt resistor), which may not always be feasible or practical. EM analysis, while non-invasive, can be challenging due to the need for specialized equipment like near-field probes and advanced signal processing to extract exploitable information. While these techniques are well-studied, other less-explored physical phenomena, potentially more subtle and difficult to defend against, may also serve as effective side channels. Such overlooked modalities can be exploited by adversaries to compromise system privacy and security in unforeseen ways.

2.1 Impedance Leakage

Impedance side channels, in contrast to EM and power side channels, emerge from changes in impedance caused by switching circuits within a **complementary metal–oxide semiconductor**



(a)



(b)

A	B	$\overline{AB}$	Z
0	0	1	Z_{00}
1	0	1	Z_{01}
0	1	1	Z_{10}
1	1	0	Z_{11}

(c)

Fig. 1. (a) 2-input NAND gate, (b) PMOS/NMOS with the parasitic capacitance, (c) truth table with relative impedance.

(CMOS) device. This unique side-channel presents a modulated signal of internal device events, wherein the two-state impedance of transistors in the CMOS device undergoes changes during operation. Our previous works [6, 8] represent a mathematical formulation of this phenomenon in *transistor-level*. In this work, we extend the analysis to the gate level by examining a two-input NAND gate, as illustrated in Figure 1(a), to present how logical inputs affect impedance. We model the behavior of PMOS and NMOS transistors in different operating regions (cutoff, linear, saturation) and derive their impedance contributions. We consider the NMOS and PMOS to have widths of 2 and 3, respectively, and C_0 denotes the fanout capacitance. Depending on the input A and B, the **metal oxide semiconductor field effect transistors (MOSFETs)** can be in the active or cutoff region. Let Z_{3c} be the impedance of $3C$ capacitance and $Z_{eq,0}$ be the impedance of the combined capacitance of $7C$ and C_0.

The impedance in the cutoff region is governed by leakage currents and can be expressed using the reverse-biased P-N junction characteristics. In the cutoff mode, a MOSFET functions as an open switch with very high resistance between the source and drain. Despite this state, a low leakage current exists due to the presence of reverse-biased P-N junctions between the source and drain. The impedance in cutoff mode is dependent on specific device specifications and operating conditions, which can be calculated by measuring the leakage current ($I_{DS,L}$) while considering the drain-to-source voltage (V_{DS}). The leakage current $I_{DS,L}$ has an expression of [9]:

$$I_{DS,L} = W.C.B^{-B.L_G} \exp\left(\frac{\Delta\Phi}{V_t}\right)\left[\exp\left(\frac{A.V_{DS}}{V_t}\right) - 1\right]. \tag{1}$$

Thus, the cutoff region impedance, Z_C, is expressed as:

$$\begin{aligned} Z_C &= \frac{V_{DS}}{I_{DS,L}} \\ &= \frac{V_{DS}}{W.C.B^{-B.L_G} \exp\left(\frac{\Delta\Phi}{V_t}\right)\left[\exp\left(\frac{A.V_{DS}}{V_t}\right) - 1\right]}. \end{aligned} \tag{2}$$

In active regions, the impedance is a combination of on-resistances and parasitic capacitances. In the linear and saturation mode, let $R_{lin,p/n}$ and $R_{sat,p/n}$ be the effective on-resistance for the PMOS/NMOS, respectively. The expression of $R_{lin,p/n}$ and $R_{sat,p/n}$ can be represented as [42],

$$R_{lin,p/n} = \frac{\frac{1}{2}(V_D - V_S - V_{t,p/n})}{\frac{3}{8}k'_{p/n}(\frac{W}{L})_{p/n}(V_D - V_S - V_{t,p/n})^2} \tag{3}$$

$$R_{sat,p/n} = \frac{V_D - V_S}{\frac{1}{2}k'_{p/n}(\frac{W}{L})_{p/n}(V_D - V_S - V_{t,p/n})^2}. \tag{4}$$

Here, $k'_{p/n}$ represents the trans-conductance, $(W/L)_{p/n}$ represents the aspect ratio, and $V_{t,p/n}$ represents the threshold voltage of the PMOS/NMOS. The voltage at the drain is denoted by V_D, while the voltage at the source is denoted by V_S. Let $R_{p/n}$ be the effective on-resistance of the PMOS/NMOS. Using Equations (3) and (4), R_p and R_n can be estimated as:

$$R_{p/n} = \frac{1}{2}(R_{lin,p/n} + R_{sat,p/n}). \tag{5}$$

Figure 1(b) presents an equivalent circuit of PMOS/NMOS containing parasitic capacitances. Based on the input X at the gate, the MOSFET changes its operating state from cutoff to saturation following the linear state. Let $C_{gdp/n}$ represent the gate to drain capacitance, $C_{dbp/n}$ represent the drain to bulk parasitic capacitance (the diffusion capacitance) for the PMOS/NMOS [35, 36]. The equivalent capacitance value, $C_{eq,p/n}$, aggregating the parasitic capacitance for the PMOS/NMOS, respectively [42], can be expressed as,

$$C_{eq,p/n} = \sum (C_{gdp/n}, C_{dbp/n}). \tag{6}$$

Let $X_{p/n}$ be the equivalent reactance of NMOS/PMOS, respectively. Interestingly, the parasitic capacitance of the PMOS/NMOS dominates the equivalent reactance $X_{p/n}$ [35]. Hence, the equivalent reactance $X_{p/n}$ follows,

$$X_{p/n} \approx \frac{-1}{\omega C_{eq,p/n}}. \tag{7}$$

Here, $\omega = 2\pi f$ and f is the signal frequency. The equivalent impedance, $Z_{p/n}$ for PMOS/NMOS, consists of equivalent resistance $R_{p/n}$ and equivalent reactance $X_{p/n}$. The expression of the equivalent impedance $Z_{p/n}$ can be expressed as,

$$Z_{p/n} = R_{p/n} + jX_{p/n}. \tag{8}$$

Using Equation (8), we can calculate the impedance for the PMOS and NMOS working in the linear and saturation regions. The physical parameters for the PMOS and NMOS are distinct. Therefore, the impedance values for the PMOS and NMOS operating in the linear and saturation regions are not identical.

To demonstrate how information is leaked through impedance in the *gate level*, we consider a simple two-input NAND gate (Figure 1(a)). Let $Z_{P,a}$, $Z_{N,a}$ be the active region impedance and $Z_{P,c}$, $Z_{N,c}$ be the cutoff region impedance for the PMOS and NMOS of the NAND gate, respectively. Equation (8) can be used to estimate $Z_{P,a}$ and $Z_{N,a}$. Equation (2), on the other hand, can be used to estimate $Z_{P,c}$ and $Z_{N,c}$. Using $Z_{P/N,a/c}$, the impedance values for all four possible input patterns to the two-input NAND gate can be estimated as presented in Figure 1(c).

Case $A = 0$, $B = 0$: Here, the two PMOS remain active while the two NMOS remain inactive. Let, the impedance between V_{DD} and ground node be Z_{00} and it can be expressed as,

$$\begin{aligned} Z_{00} &= Z_{P,a} || Z_{P,a} + (Z_{N,c} || Z_{3C} + Z_{N,c}) || Z_{eq,0} \\ &= \frac{Z_{P,a}}{2} + \frac{(Z_{N,c}Z_{3C} + Z_{N,c}(Z_{N,c} + Z_{3c}))Z_{eq,0}}{Z_{N,c}Z_{3C} + (Z_{N,c} + Z_{eq,0})(Z_{N,c} + Z_{3c})}. \end{aligned} \tag{9}$$

Similarly, for the input combinations *Case* $A = 0$, $B = 1$, *Case* $A = 1$, $B = 0$, and *Case* $A = 1$, $B = 1$, the resulting impedances from the supply voltage side can be represented by Z_{01}, Z_{10}, and

Z_{11}, respectively, as defined in Equations (10)–(12).

$$Z_{01} = Z_{P,a}||Z_{P,c} + (Z_{N,c}||Z_{3C} + Z_{N,a})||Z_{eq,0} \tag{10}$$

$$Z_{10} = Z_{P,c}||Z_{P,a} + (Z_{N,a}||Z_{3C} + Z_{N,c})||Z_{eq,0} \tag{11}$$

$$Z_{11} = Z_{P,c}||Z_{P,c} + (Z_{N,a}||Z_{3C} + Z_{N,a})||Z_{eq,0} \tag{12}$$

This section considers a simple two-input NAND gate to demonstrate the concept of different input combinations producing distinct impedance values. The equivalent impedance between V_{DD} and ground nodes as expressed by Equations (9)–(12) is distinct for all the possible input combinations of A and B. Although this is a simplified gate-level model, it provides physical justification for using impedance as a leakage source. Different input combinations across logic gates induce unique impedance values, which aggregate at the system level and form the basis of measurable leakage. This foundation allows us to interpret observed signals as instruction-dependent phenomena, bridging the physical circuit behavior with high-level software execution.

To extend the gate-level analysis to system-level observations, we introduce a comprehensive chip-level impedance model, as expressed in Equation (13). This model characterizes how the impedance characteristics of a digital system evolve in response to instruction execution and frequency-dependent behavior. In this formulation, the total impedance of the system, $Z_{\text{chip}}(I, f)$, is modeled as a function of the executed instruction $I \in S$, where S is the instruction set of the system, and the probing frequency f. The term $R_{\text{dc}}(I)$ denotes the DC resistance arising from the instruction-dependent switching paths active during logic transitions. The components $X_L(I, f)$ and $X_C(I, f)$ represent the inductive and capacitive reactances, respectively, which also vary based on the specific instruction and frequency. The term $Z_{\text{pkg}}(f)$ accounts for the impedance introduced by the packaging and interconnect structures. Finally, $Z_{\text{f.only}}(f)$ captures additional impedance contributions from purely frequency-dependent sources such as decoupling capacitors or transmission line effects.

$$Z_{\text{chip}}(I, f) = R_{\text{dc}}(I) + jX_L(I, f) + \frac{1}{jX_C(I, f)} + Z_{\text{pkg}}(f) + Z_{\text{f.only}}(f). \tag{13}$$

As different instructions toggle distinct sets of logic gates and switching paths, they induce variations in these impedance components. These variations manifest as frequency-dependent signatures that can be captured during program execution. To quantify this behavior, we define the instruction-dependent impedance difference in Equation (14).

$$\Delta Z = [Z_{\chi_i} - Z_{\chi_j}] = Z(I_i, f) - Z(I_j, f), \quad \forall i, j \in S. \tag{14}$$

Equation (14) formalizes the concept of impedance leakage as a differentiable signal resulting from the execution of two distinct instructions. It captures the measurable shift in impedance that occurs when the system transitions from one instruction state, I_i, to another, I_j. This differential model links physical circuit-level activity to system-level impedance responses. By integrating this chip-level formulation with our earlier gate-level model, we establish a theoretically grounded pathway for instruction-level identification based on impedance analysis. This connection demonstrates that instruction-specific switching activity induces distinct and traceable changes in the system's impedance spectrum.

3 Exploiting Impedance Leakages to Reverse-Engineer Embedded Software

Unlike conventional side channels such as power and EM emissions that monitor fluctuations in power consumption or EM radiation, impedance side channels directly characterize the intrinsic impedance of a device across a range of frequencies. This novel perspective of impedance side-channel transcends the conventional understanding of impedance, which typically focuses on

its role in circuit design and signal transmission. Instead, the impedance side-channel considers impedance as a dynamic and sensitive parameter that varies with internal switching activity, as detailed in Section 2.1. Impedance variations, driven by the switching of Logic gates, form patterns that can be instruction-specific and measurable. This offers a new path for observing software execution externally, including scenarios where power or EM channels can be less effective [4, 31].

Consequently, the impedance side-channel can detect counterfeit hardware and printed circuit boards by profiling authentic components and identifying impedance mismatches [30, 45]. Variations in device impedance during cryptographic operations can also be exploited to extract secret encryption keys [20]. Furthermore, discernible impedance shifts caused by the execution of different program instructions open new possibilities for monitoring and analyzing software behavior [6]. By observing these variations during instruction execution, it becomes feasible to identify the exact sequence of instructions running on a device. This instruction-level insight enables a wide range of security applications, including verifying program integrity, detecting injected or malicious code, and identifying unauthorized or tampered systems lacking legitimate developer fingerprints. Collectively, these features position impedance SCA a promising, underexplored method to enhance embedded system security.

Our previous work [6] demonstrates the feasibility of distinguishing instruction types, such as data transfer, arithmetic, logic, and control, using impedance-based SCA. Distinct impedance profiles are observed during the execution of each instruction type. Building on this foundation, the present study significantly extends the analysis by exploring whether individual instructions can be distinguished and reverse-engineered based on their unique impedance profiles. To achieve this, we introduce a frequency-domain signal analysis framework that extracts fine-grained runtime impedance features and applies ML to classify instructions with high accuracy. This links physical-layer signals to software semantics, turning impedance from a passive parameter into a powerful tool for software disassembly.

The motivation of this research is to evaluate the reliability and potential of adopting impedance SCA, a hardware-based approach, for disassembling individual software instructions. By transforming impedance from a static, passive property into an active information source, we can bridge the gap between hardware characteristics and software security. The ability to link impedance profiles to instruction behavior can revolutionize reverse engineering and greatly empower the protection of connected systems against evolving cyber threats [33, 41].

4 Experimental Setup

The experimental setup evaluates the feasibility of using device impedance as a side-channel for reverse-engineering individual software instructions. The process consists of three stages: hardware configuration, software setup, and signal collection. First, we establish a testbed using two embedded platforms—the ATmega328P microcontroller and a custom CPU on a Xilinx Artix-7 FPGA—to demonstrate the generality of the approach. A **vector network analyzer (VNA)** is used to inject test signals and measure reflected impedance, with the setup optimized for signal integrity and consistency. In the second stage, instruction-level programs are designed to repeatedly execute specific opcodes. These programs are compiled, deployed using platform-specific toolchains, and executed. Finally, the VNA captures impedance responses across a wide frequency range during program execution. The collected signals are then pre-processed and prepared for feature extraction and classification. Further details on each stage are provided in the following subsections.

4.1 Hardware Setup

Our goal in this experiment is to evaluate whether impedance side-channel measurements can be used to reverse-engineer program instructions. We also to study the impact of hardware complexity

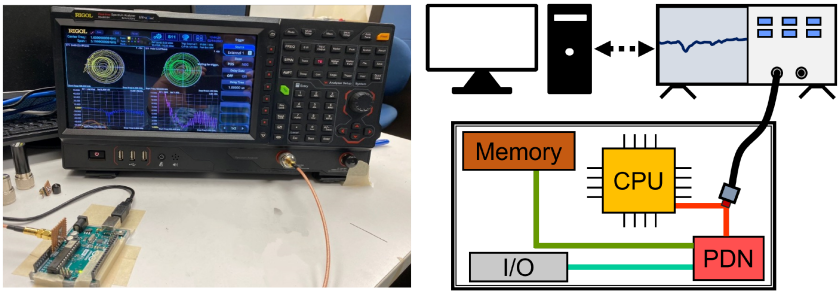


Fig. 2. Hardware setup: measurement of *impedance* signals.

on this new side-channel. To do this, we conduct experiments on two different representative platforms: the Arduino Uno [2], which features an 8-bit AVR ATmega328P microcontroller [3], and the Alchitry AU board, which contains an Artix-7 FPGA [1]. We choose these platforms due to their contrasting levels of complexity and configurability. The primary differences between the two are summarized as follows:

—FPGAs support parallel execution and hardware reconfiguration, whereas microcontrollers operate sequentially with fixed hardware.
—FPGA hardware can be reconfigured and customized, but microcontroller hardware has limited flexibility.

We acknowledge that these platforms are less complex than industry-standard processor architectures. However, the ATmega328P is widely used in both commercial embedded systems and educational environments, making it a practical and relevant baseline for evaluating new side-channel modalities. The Artix-7 FPGA, on the other hand, enables implementation of a custom instruction set architecture with full control over system behavior, allowing for precise isolation and characterization of instruction-specific leakage. These attributes make the chosen platforms well-suited for foundational studies of impedance-based analysis in a controlled and reproducible setting. Moreover, the core physical phenomenon investigated, i.e., impedance variation resulting from gate-level switching activity, is intrinsic to digital logic and is architecture agnostic. As such, they are expected to persist across more advanced processors. Nonetheless, applying the proposed method to complex ISAs, such as ARM Cortex-M or RISC-V, introduces additional challenges related to pipelining, concurrency, and instruction overlap. Exploring these scenarios represents a promising area for future research. However, such an investigation lies outside the scope of the current work, which is centered on establishing a foundational understanding of impedance-based fingerprinting.

Figure 2 illustrates the hardware setup for our experiments on the Arduino and Alchitry systems. For both devices, we target the 3.3V power domain since it is closer to the CPU voltage level. The Arduino and Alchitry both have onboard 3.3V pins, so no physical modification or tampering is required to tap into this domain and monitor impedance changes near the core of each device. By leveraging the 3.3V pins, we observe the impedance side-channel on both platforms non-invasively. This highlights a key benefit of impedance side-channel—the ability to monitor signals without direct hardware access or modification. The use of native 3.3V pins enables aboveboard measurements, avoiding tampering.

To collect impedance data, we use a Rigol RSA5032N spectrum analyzer operating in VNA mode. The VNA is connected to the **power delivery network (PDN)** of each device via a coaxial cable. The coaxial cable consists of an inner conductor surrounded by a concentric conducting shell,

which enables signal transmission in the center while shielding against interference. The cable's design minimizes signal loss and EM interference. We measure impedance fluctuations with better precision by directly attaching the VNA to the PDN. The VNA-to-PDN connection enables us an accurate means to monitor impedance changes related to processor activity in each device.

While VNAs are typically more expensive than the equipment used in conventional power or EM SCA, they offer distinct advantages. Unlike power analysis, which captures a single, aggregated time-domain trace of current consumption and is often vulnerable to noise or attenuation from power-distribution filtering, VNAs operate in the frequency domain and measure a system's impedance response across a range of frequencies. It provides a much richer multi-dimensional view of internal switching dynamics. Impedance is inherently frequency-dependent, and by sweeping through discrete frequency points, the VNA captures a detailed profile that reflects the device's internal switching behavior under different operating conditions. These impedance measurements at multiple frequencies provide diverse and complementary perspectives on the system's state, effectively enriching the observable side-channel information. This level of granularity is particularly valuable for fine-grained classification tasks, such as instruction-level fingerprinting, as well as for scenarios where conventional power analysis fails due to low power consumption or aggressive power filtering [4]. Although high-end VNAs can be expensive, more cost-effective alternatives are increasingly available. In resource-constrained environments, similar impedance characteristics can be probed by injecting RF signals and analyzing the reflected signals with spectrum analyzers or oscilloscopes. While such setups may offer reduced measurement precision, prior work has demonstrated their capability in capturing impedance variations resulting from internal switching activity [30–32]. Furthermore, on-chip impedance sensing using FPGAs can provide scalable and low-cost implementations, eliminating the need for external instrumentation.

4.2 Software Setup

We program the test devices in assembly language to maintain control over instruction execution. Assembly directly corresponds to hardware-level operations through opcodes and operands, allowing precise specification of data manipulation, branching, and register-level interactions. The opcode indicates the operation, while the operands provide the data. Assembly language allows writing code with the CPU's basic instructions. This low-level control is essential for isolating and characterizing instruction-specific behavior during impedance measurements.

For the Artix-7 FPGA, we implement a simple 8-bit CPU with a custom 12-instruction set. We also develop a custom assembler to translate the instructions into memory writes for the FPGA. The architecture of the custom CPU is illustrated in Figure 3. In total, we define 12 instruction types: data transfer (LOAD, STORE, SET), arithmetic/logic (ADD, SUB, AND, OR, XOR), rotate (SHL, SHR), and branch (BEQ, BNEQ). This custom 12-instruction set enables basic program execution on our FPGA CPU for testing the impedance side-channel. Table 1 summarizes the instruction operations, which are briefly described below.

LOAD, STORE, SET: LOAD and SET transfer data into registers, while STORE transfers data from a register to memory. LOAD and STORE use the syntax: *Opcode R1, R2, K*. For LOAD, R1 receives the value. For STORE, R1 provides the output value. R2 contains the base address, and K is an offset constant. SET uses the syntax: *Opcode R1, K*. R1 is the target register and K is the 8-bit value to assign.

ADD, SUB, AND, OR, XOR: The arithmetic and logic instructions (ADD, SUB, AND, OR, XOR) follow the syntax: *Opcode R1, R2, R3*. They perform operations between registers R2 and R3, storing the result in R1. Specifically, ADD does addition, SUB does subtraction, AND performs a logical AND, OR does a logical OR, and XOR executes an exclusive OR.

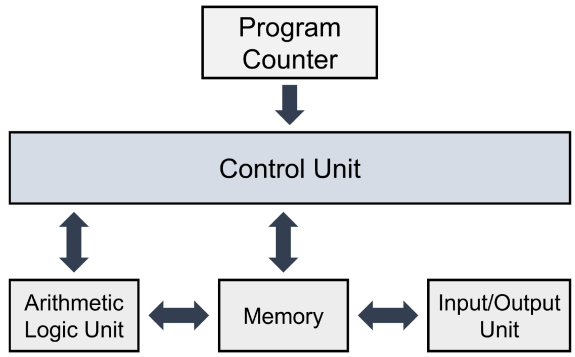


Fig. 3. Architecture of custom FPGA CPU.

Table 1. Program Instruction Set Summary

Instruction type	Description	Artix 7 FPGA			ATmega328P Microcontroller		
		Opcode	Operands	Operation	Opcode	Operands	Operation
Data transfer	Load between registers	LOAD	R1, R2, K	R1 ← [R2 + K]	MOV	R1, R2	R1 ← R2
	Store from register to memory	STORE	R1, R2, K	R1 → [R2 + K]	—	—	—
	Load constant into register	SET	R1, K	R1 ← K	LDI	R1, K	R1 ← K
Arithmetic and logic	Addition	ADD	R1, R2, R3	R1 ← R2 + R3	ADD	R1, R2	R1 ← R1 + R2
	Subtraction	SUB	R1, R2, R3	R1 ← R2 − R3	SUB	R1, R2	R1 ← R1 − R2
	Bit-wise AND	AND	R1, R2, R3	R1 ← R2 & R3	AND	R1, R2	R1 ← R1 & R2
	Bit-wise OR	OR	R1, R2, R3	R1 ← R2 | R3	OR	R1, R2	R1 ← R1 | R2
	Bit-wise XOR	XOR	R1, R2, R3	R1 ← R2 ⊕ R3	EOR	R1, R2	R1 ← R1 ⊕ R2
Rotate	Binary left shift	SHL	R1, R2, R3	R1 ← R2 ≪ R3	LSL	R1	R1[n + 1] ← R1[n] R[0] ← 0
	Binary right shift	SHR	R1, R2, R3	R1 ← R2 ≫ R3	LSR	R1	R1[n] ← R1[n + 1] R[7] ← 0
Branch	Branch if equal	BEQ	R1, K	if R1 = K: PC ← PC + 2	BREQ	K	if Z = 1: PC ← PC+K+ 1
	Branch if not equal	BNEQ	R1, K	if R1: ≠ K PC ← PC + 2	BRNE	K	if Z = 0: PC ← PC+K+1

SHL, SHR: These two instructions have the syntax: *Opcode R1, R2, R3*. The SHL and SHR instructions shift the bits in the R2 register value left or right, respectively, by the number of bit positions specified in the R3 register and store the result in the R1 register. The **least significant bits (LSBs)** and **most significant bits (MSBs)** are padded with zeros following the SHL and SHR, respectively.

BEQ, BNEQ: The BEQ and BNEQ instructions follow the syntax: *Opcode R1, K* and allow conditional branching based on comparing a register, R1, and constant, K. BEQ skips the next instruction by increasing programming counter, *PC*, by 2 if they are equal. BNEQ does the opposite (skips if they are not equal).

Similarly, for the Arduino's ATmega328P AVR microcontroller, we adhere to the conventional assembly instructions provided by the manufacturer [3]. We select 11 representative instructions,

summarized in Table 1, covering similar categories as the FPGA design: data transfer, arithmetic/logic, rotate, and branching. The 8-bit ATmega328P microcontroller is based on RISC architecture with 1 KB EEPROM, 2 KB SRAM, 23 GPIO, and 32 general registers. The chosen instruction set mirrors the functionality of the custom FPGA instructions, leveraging the ATmega328P's standard feature set. The following paragraph briefly outlines the operations of the instructions.

MOV, LDI: MOV instruction is used to transfer data from one register to another, while LDI instruction is used to load an immediate to the designated register.

ADD, SUB, AND, OR, EOR: These instructions execute the arithmetic and logical operations. ADD, SUB, AND, OR, and EOR perform the addition, subtraction, logical AND, OR, and XOR operations, respectively.

LSL, LSR: LSL and LSR shift one bit of the specified register value to the left or right, respectively. The LSBs and MSBs are padded with zeros following the LSL and LSR instructions, respectively.

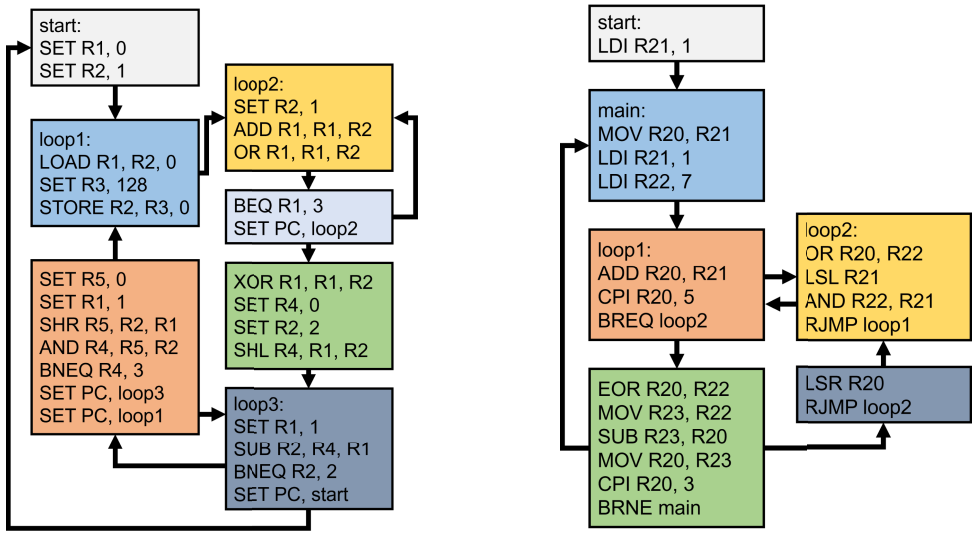
BREQ, BRNE: The BREQ (branch if equal) and BNEQ (branch if not equal) instructions control program flow based on the zero flag, Z. They are often used after arithmetic operations that set or reset Z. If Z is 0, BREQ will branch to a different instruction sequence instead of the next immediate instruction. BNEQ branches if Z is not 0. By manipulating the program counter (*PC*), these instructions create conditional loops and branching logic.

To characterize the impedance response of each instruction, we design controlled test sequences, such as the example illustrated in Figure 4, which execute every instruction in the defined set. These sequences are not intended to serve any functional or algorithmic purpose but are structured to isolate and repeat individual instructions for stable impedance signal acquisition. Conditional branches and jump instructions are included to enable repeated execution of specific instruction blocks. Importantly, these sequences do not contain any special structural properties that facilitate reverse engineering. On the contrary, their simplicity is intentional, ensuring that the collected signals reflect instruction-specific features. In practice, we generate multiple randomized instruction streams using different combinations of instructions, including varied control flows, and observed consistent classification performance across these variants. For clarity of presentation, we present results for the representative sequence illustrated in Figure 4. Since our method operates at instruction-level granularity, once individual instruction fingerprints are learned, the classifier can generalize to arbitrary instruction flows, regardless of their length or structure. Additional control instructions, such as *CPI* and *RJMP*, are used in the ATmega328P to facilitate branching and loop formation during code execution. The *CPI* instruction compares an immediate value to a register value and sets the zero flag Z. The *RJMP* instruction performs a relative jump to a specified location in the code.

4.3 Signal Collection

In our study, we use the Rigol RSA5032N spectrum analyzer configured as a VNA. The RSA5032N contains built-in circuitry to perform VNA measurements and supports **standard commands for programmable instruments (SCPI)**. SCPI instructions transfer the trace data over Ethernet in complex format. Therefore, the received complex trace values need to be transformed into impedance values for analysis.

Let $T_{Re,m}$ and $T_{Im,m}$ represent the real and imaginary portions of the trace at frequency m. First, we convert the real and imaginary components to resistance R_m, and reactance, X_m using Equations (15) and (16), respectively [15]. All raw traces are collected during software instruction execution



(a) Artix-7 FPGA. (b) ATmega328P microcontroller.

Fig. 4. Representative instruction sequences for characterizing instruction-level impedance leakage.

and transformed into impedance signals.

$$R_m = Z_{ref} * \frac{(1 - T_{Re,m}^2 - T_{Im,m}^2)}{(1 - T_{Re,m})^2 + T_{Im,m}^2} \tag{15}$$

$$X_m = Z_{ref} * \frac{2 * T_{Im,m}}{(1 - T_{Re,m})^2 + T_{Im,m}^2}. \tag{16}$$

Here, Z_{ref} is the reference impedance of the measurement instrument. The overall complex impedance, $Z_{T,m}$, at the trace point m is calculated as,

$$Z_{T,m} = R_m + iX_m. \tag{17}$$

We specify 10,001 linearly spaced frequency points from 500 kHz to 3.2 GHz. We set the VNA's internal averaging factor to 100 to minimize the measurement noise floor, ensuring precision in our data collection. We control the clock and use triggering method to collect signals for any specific instructions. Using a custom Python script with SCPI commands, we collect 700 traces per FPGA instruction (8,400 total) and 500 traces per microcontroller instruction (5,500 total). These raw traces are in complex form. The complex trace values are converted to impedance using Equation (17), where Z_{ref} is the 50 Ω reference impedance. By transforming the complex trace values into impedance, we obtain the necessary data for analysis. The resulting impedance trace signals are then labeled for supervised learning. We use 70% of the data for training and 30% for testing the ML classifiers. While training the ML classifiers, we use a validation dataset through 10-fold cross-validation on the training dataset.

5 Evaluation

5.1 Signal Analysis

The impedance profiles for each instruction execution are visibly distinct, as seen in Figure 5(a) for three Artix 7 operations (LOAD, AND, BNEQ) and Figure 5(b) for the ATmega328P (MOV, ADD,

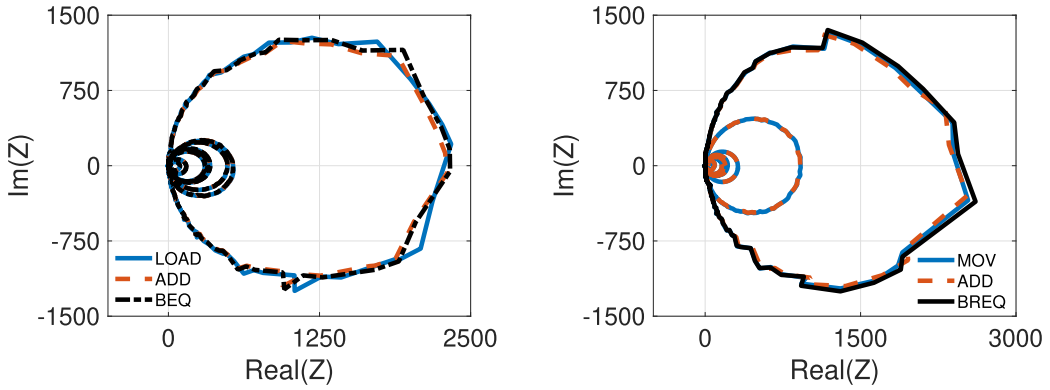


Fig. 5. Impedance profile of program instructions. (a) Artix 7: LOAD, ADD, BEQ, (b) ATmega328P: MOV, ADD, BREQ.

BREQ). Though subtle, variations between instructions become evident at certain frequencies. This motivates the removal of redundant frequency points from the raw impedance signals. Further analysis uses the impedance magnitude values, as they capture most of the relevant information [5, 8]. At each measured frequency, the signals follow a Gaussian distribution, representing additive white Gaussian thermal noise. Figure 6 histograms illustrate the normal distribution of impedances at two of the selected frequencies. These frequency-dependent impedance variations enable the identification of instruction-specific signatures. To enhance the robustness of our analysis, we apply statistical filtering to exclude outlier values and average across multiple runs. This helps to suppress random fluctuations and accentuate instruction-dependent variations. Additionally, we verify the repeatability of impedance patterns by reprogramming the same instruction sequences across different devices and measurement sessions, confirming the consistency of the observed trends.

5.2 Signal Processing

To determine relevant frequency points, we compute the Pearson correlation coefficient between the impedance signal at each frequency and the program instruction set. This allows us to select a subset of the 10,001 total frequency points where the impedance responses are highly correlated with changes in instruction execution. Frequency points with lower correlation to instruction changes are rejected.

The impedance value at a particular frequency is denoted as X and the instruction set as I. For n sample pairs $\{(X_1, I_1),\ldots,(X_n, I_n)\}$, the Pearson correlation coefficient ρ_{XI} is calculated as Equation (18). We select the largest set of frequencies where the impedance responses are fairly orthogonal, allowing us to further reduce the number of points. Using Equation (18), we calculate the Pearson correlation coefficients between X_i (impedance at one frequency) and Y_i (impedance at another frequency). We choose the maximum number of frequencies where the intra-group impedance responses are less than 85%. In other words, the impedance variations at rejected frequencies can be explained by the impedance at selected frequencies. Since impedance at the chosen frequencies can explain the rejected points, we termed these the *Dominant frequency points*.

$$\rho_{XI} = \frac{n\sum_{i=1}^{n} X_i I_i - \sum_{i=1}^{n} X_i \sum_{i=1}^{n} I_i}{\sqrt{n\sum_{i=1}^{n} X_i^2 - \left(\sum_{i=1}^{n} X_i\right)^2}\sqrt{n\sum_{i=1}^{n} I_i^2 - \left(\sum_{i=1}^{n} I_i\right)^2}}. \tag{18}$$

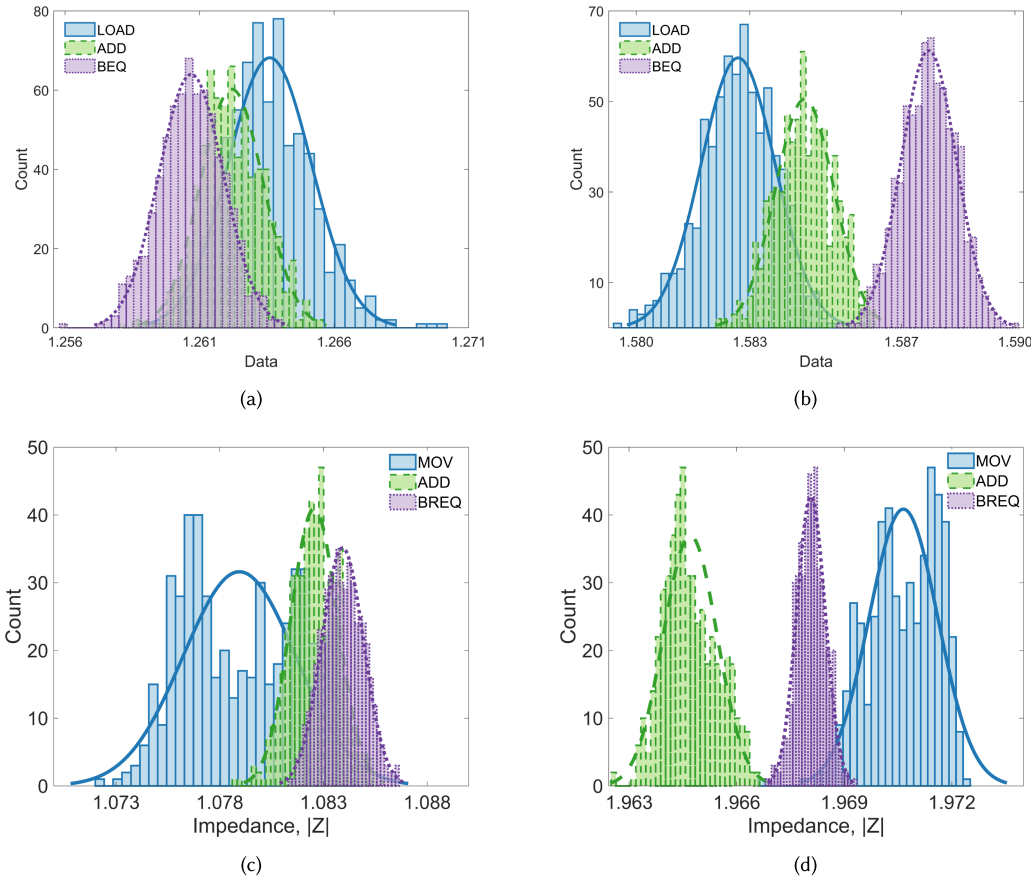


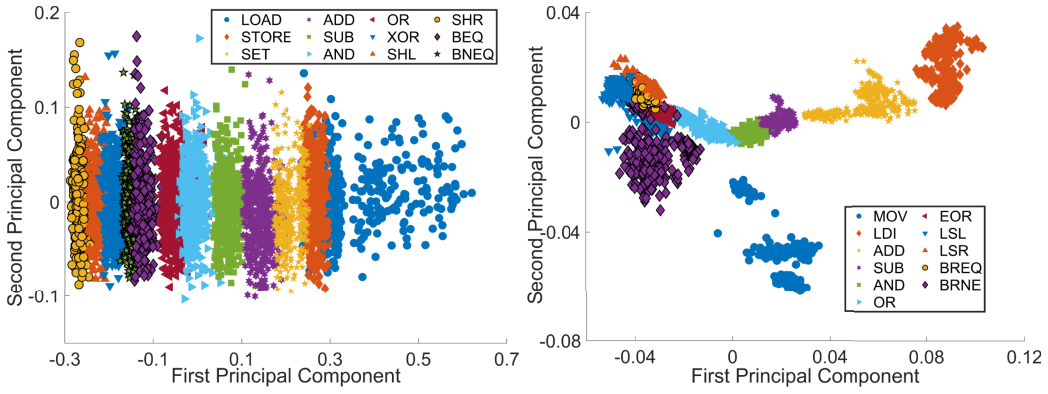
Fig. 6. Impedance profile distribution: Artix 7 at (a) 1 GHz, (b) 2.3 GHz, and ATmega328P at (c) 1 GHz, (d) 2.3 GHz.

Using this two-step frequency selection, we locate approximately 3,600 and 2,700 frequency points for the FPGA and ATmega328P, respectively, that adequately characterize the software instructions. Notably, these represent just 36% and 27% of the total observable frequency points. We then collect additional impedance signals during various program executions at the specified frequencies to construct the test dataset. By selecting dominant orthogonal frequencies, we drastically reduce the data required while retaining the information needed to reverse-engineer software instructions.

5.3 Feature Extraction

We use the training data to identify features associated with each instruction's execution. To extract the most informative components from the high-dimensional impedance measurements, we apply **principal component analysis (PCA)** [21]. PCA transforms the raw frequency-domain data into a lower-dimensional space, capturing the directions of greatest variance while discarding noise and redundancy. We use the training data to identify features associated with each instruction's execution.

To optimize performance, we retain only those principal components that collectively explain 95% of the variance in the training data. This dimensionality reduction not only enhances classifier accuracy but also significantly reduces training time and computational complexity. The resulting



(a) Artix 7.

(b) ATmega328P.

Fig. 7. Distribution of the two most significant principal components for multiple program instructions.

feature vectors serve as compact inputs to ML classifiers for instruction identification. Figure 7(a) and (b) illustrates the distribution of data projected onto the two most significant principal components for the FPGA and microcontroller instruction sets, respectively. Each instruction exhibits distinct clustering in the reduced feature space. These observations support the feasibility of instruction classification based on impedance-derived features.

5.4 Performance Metrics

To evaluate classifier performance, we observe five key metrics: recall rate (**true-positive rate (TPR)**), specificity (true-negative rate), precision, accuracy, and F1-score [26] for the test dataset. All classes are weighted equally when calculating the macro-average of these metrics. Let TP_i, TN_i, FP_i, and FN_i represent the true positives, true negatives, false positives, and false negatives predicted by a given classifier for class i out of N classes. The metrics are defined as:

$$Recall = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{TP_i}{TP_i + FN_i}\right) \tag{19}$$

$$Specificity = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{TN_i}{TN_i + FP_i}\right) \tag{20}$$

$$Precision = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{TP_i}{TP_i + FP_i}\right) \tag{21}$$

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i}\right) \tag{22}$$

$$F1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall}. \tag{23}$$

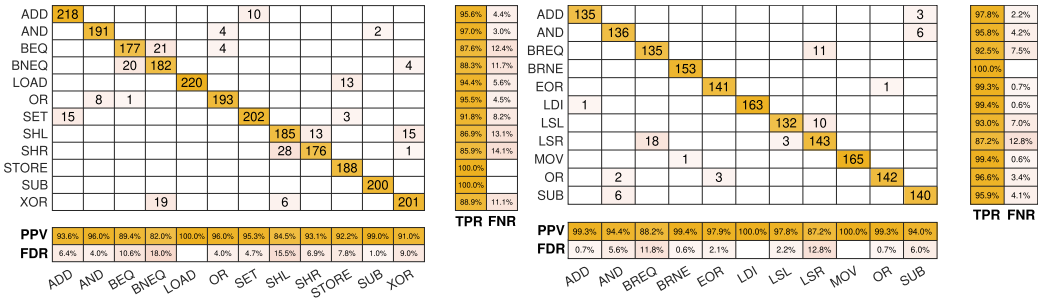
We use Equations (19)–(23) to calculate the performance metrics and compare the performance of the classifiers.

5.5 Detection and Classification

We use several ML models, including **support vector machine (SVM)**, **adaptive boosting (AdaBoost)**, **k-nearest neighbors (kNN)**, linear discriminant, and Gaussian Naive Bayes [26], to

Table 2. Classification Scores for Detecting Program Instruction

Device	Classifier [26]	Validation Score	F1-Score	Recall	Specificity	Precision	Accuracy
FPGA	SVM (Kernel: Linear)	92.8%	92.6%	92.7%	99.3%	92.7%	92.6%
	AdaBoost	91.8%	92.0%	92.0%	99.3%	92.1%	92.0%
	Linear Discriminant	86.6%	87.2%	87.3%	98.8%	87.3%	87.3%
ATmega328P	SVM (Kernel: Quadratic)	95.0%	96.1%	96.1%	99.6%	96.1%	96.1%
	Bagged trees	92.5%	91.6%	91.7%	99.2%	91.6%	91.6%
	Linear Discriminant	88.9%	92.2%	92.0%	99.2%	92.8%	92.2%



(a) (b)

Fig. 8. Confusion matrix of reverse-engineered program instructions: (a) Artix 7 FPGA, (b) ATmega328P microcontroller.

detect and classify program instructions. Each model is trained using 10-fold cross-validation to enhance stability and prevent overfitting. The best-performing classifier is selected based on the average cross-validation accuracy.

After extracting instruction-specific features from the impedance traces as outlined in previous sections (Sections 5.2 and 5.3), the models are evaluated on a 30% held-out test set. The SVM performs exceptionally well and achieves top classification performance, with F1-score, precision, recall, specificity, and accuracy all over 92% on the test data. This demonstrates the viability of our proposed approach using impedance side-channel for reverse-engineering program instructions. The findings of the classification of individual program instructions are presented in Table 2. The confusion matrix in Figure 8 provides further insight into the performance of the top classifier, SVM, along with the **positive predictive value (PPV)**, **false-discovery rate (FDR)**, TPR, and false-negative rate. PPV represents the probability that a predicted result is actually correct, whereas FDR is the complement of PPV. A high PPV indicates the reliability of the trained model, and the false positive rate is the complement of specificity. Among the other classifiers, SVM demonstrates the ability to accurately differentiate between the various instructions with minimal confusion and a top 10-fold cross-validation score.

In summary, by leveraging ML algorithms and extracted signal features, our method of using impedance as a side-channel can reliably identify and reverse-engineer program instructions executed on a chip. The exceptional classification results from the SVM highlight the promise of this technique. While our ML pipeline-comprising PCA for feature extraction and standard classifiers such as SVM, kNN, and AdaBoost-is not novel, its application to impedance-based side-channel signals is unexplored. Existing studies primarily rely on power analysis or EM emissions.

In contrast, we demonstrate that even well-established ML models can extract meaningful instruction-specific patterns from impedance data. This practical demonstration validates impedance as a viable side-channel modality.

6 Related Works

SCA in Hardware Defense: Side channels in hardware have been leveraged both for extracting sensitive information and securing systems. For example, the authors in [25] use timing-based information leakage and simple power analysis to analyze vulnerabilities in embedded neural networks. In [19], an EM side-channel-based hardware Trojan detection is proposed by spectrum modeling and analyzing. Impedance can be employed as a tool for detecting malicious system as well as other physical side-channel attacks, e.g., power side-channel attacks. Authors in [47] detect board-level modifications by measuring impedance at various locations. Another approach in [17] detects Trojans by monitoring impedance changes in integrated circuit wires. Authors in [29] present a detection system that monitors battery impedance to identify malicious probing attempts in power side-channel attacks. By detecting impedance changes, their approach provides a new way to thwart such attacks through impedance-based sensing. Thus, prior works have demonstrated side channels like timing, power, EM emissions, and impedance can be exploited for analyzing hardware vulnerabilities, detecting Trojans, and identifying malicious modifications. Our research similarly uses impedance as a side-channel, but with the novel goal of reverse-engineering software instructions.

SCA in Software Defense: Recent work has leveraged physical side channels produced by embedded microcontrollers to infer internal behavior and operations. For example, authors in [10] present an analytical approach to assess the security of **hardware performance counters (HPCs)** for malware detection. Their framework analyzes the probability of malware detection given a program, set of HPCs, and sampling rate. In [13], the authors present a method utilizing power side-channel and ML on statistical trace features to detect runtime malware in medical devices. Analyzing EM emissions is another approach explored in prior art [18] detected malicious **programmable logic controller (PLC)** code by training a sequential neural network model on pre-processed frequency data to encode control flow transitions. Other works include detecting behavioral changes in industrial control systems [40] and identifying malicious PLC code [11]. Beyond EM and power analysis, recent work has investigated backscattering side channels [30]. In these attacks, adversaries analyze subtle variations in the reflected signal, which are modulated by impedance changes within the device.

SCA in Code Reverse Engineering: The authors in [16] implement an instruction classifier for the PIC16F687 microcontroller by performing a template attack using the power side-channel. With statistical models like **hidden Markov models (HMMs)**, they achieve 70.1% accuracy over 35 test instructions. By modeling control flows with HMMs on an 8-bit AVR, the authors in [24] demonstrate how voltage drops on the power pin can be measured to track code execution and detect anomalies. Similarly, the authors in [28] utilize instruction-level power side-channel leakage profiles to identify program instructions executed on an embedded processor. Other works include the use of localized EM probes to capture high-resolution side-channel traces for dynamic code recognition, as demonstrated in [39].

The extensive prior art demonstrates side channels can reveal rich details about a device's internal operations. Our work similarly leverages impedance as a side-channel, but with the unique goal of reverse-engineering software instructions on FPGA and microcontroller platforms. By contrast, most existing techniques focus on malware detection, behavior identification, and code profiling rather than reverse engineering the actual opcodes. The ability to reliably reconstruct sequences

of software instructions will open up new possibilities for observation, verification, and reverse engineering of embedded and cyber-physical systems.

7 Conclusion

In this article, we investigated the feasibility of using impedance as a side-channel signal to identify program instructions. Experimental validation was performed on two platforms with distinct instruction sets, an Artix-7 FPGA and an ATmega328P microcontroller, demonstrating instruction classification accuracies of 92.6% and 96.1%, respectively. By analyzing impedance variations induced by instruction-level switching activity, we extracted discriminative signal patterns across selected frequency channels and applied classical ML models to achieve high-accuracy classification. While the classifiers used are well-established, their application to impedance-based side channels is novel, revealing a previously unexplored attack surface. To support our findings, we developed a refined two-stage frequency selection method and introduced a gate-level mathematical formulation linking switching activity to impedance behavior. These results demonstrate that runtime impedance can effectively reflect executed instructions with high accuracy, opening new possibilities for code recognition, control flow analysis, reverse engineering, and embedded systems security. Importantly, the underlying leakage arises from fundamental switching behavior in digital logic gates and is therefore expected to generalize across architectures. Although this study focused on low-complexity platforms, future work will extend the methodology to more advanced architectures, including ARM Cortex-M and RISC-V. Such extensions present additional challenges, such as instruction overlap, increased background activity, and limited PDN access. Addressing these complexities may require advanced triggering, signal processing, or on-chip impedance sensing techniques. By establishing impedance as an active side-channel modality rather than a passive electrical property, this work introduces a new perspective on system observability and highlights the importance of exploring unconventional leakage sources to enhance cyber resilience in increasingly interconnected and security critical computing environments.

Acknowledgments

We would like to thank the reviewers for providing their valuable feedback.

References

[1] Alchitry. 2024. Alchitry Au. Retrieved September 9, 2024 from https://alchitry.com/boards/au
[2] Arduino UNO. 2024. Arduino Uno Rev3. Retrieved September 9, 2024 from https://docs.arduino.cc/hardware/uno-rev3
[3] ATmega328P Specification and Datasheet. 2024. ATmega328P Specification and Datasheet. Retrieved September 9, 2024 from https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
[4] Md Sadik Awal, Buddhipriya Gayanath, and Md Tauhidur Rahman. 2024. Impedance vs power side-channel vulnerabilities: A comparative study. arXiv:2405.06242. Retrieved from https://arxiv.org/abs/2405.06242
[5] Md Sadik Awal, Arjuna Madanayake, and Md Tauhidur Rahman. 2022. Nearfield RF sensing for feature-detection and algorithmic classification of tamper attacks. *IEEE Journal of Radio Frequency Identification* 6 (2022), 490–499.
[6] Md Sadik Awal and Md Tauhidur Rahman. 2023. Disassembling software instruction types through impedance side-channel analysis. In *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 227–237. DOI: 10.1109/HOST55118.2023.10133318
[7] Md Sadik Awal and Md Tauhidur Rahman. 2025. DExiM: Exposing impedance-based data leakage in emerging memories. In *2025 58th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE.
[8] Md Sadik Awal, Christopher Thompson, and Md Tauhidur Rahman. 2022. Utilization of impedance disparity incurred from switching activities to monitor and characterize firmware activities. In *2022 IEEE Physical Assurance and Inspection of Electronics (PAINE)*. IEEE, 1–7.
[9] R. Jacob Baker. 2019. *CMOS: Circuit Design, Layout, and Simulation*. John Wiley & Sons.
[10] Kanad Basu, Prashanth Krishnamurthy, Farshad Khorrami, and Ramesh Karri. 2019. A theoretical study of hardware performance counters-based malware detection. *IEEE Transactions on Information Forensics and Security* 15 (2019), 512–525.

[11] Nathaniel Boggs, Jimmy C. Chau, and Ang Cui. 2018. Utilizing electromagnetic emanations for out-of-band detection of unknown attack code in a programmable logic controller. In *Cyber Sensing 2018*, Vol. 10630, SPIE, 84–99.
[12] Martin Brisfors, Michail Moraitis, and Elena Dubrova. 2022. Side-channel attack countermeasures based on clock randomization have a fundamental flaw. *Cryptology ePrint Archive*. Retrieved from https://eprint.iacr.org/2022/1416
[13] Shane S. Clark, Benjamin Ransford, Amir Rahmati, Shane Guineau, Jacob Sorber, Wenyuan Xu, and Kevin Fu. 2013. {WattsUpDoc}Power side channels to nonintrusively discover untargeted malware on embedded medical devices. In *2013 USENIX Workshop on Health Information Technologies (HealthTech '13)*.
[14] Stephen Dunlap, Jonathan Butts, Juan Lopez, Mason Rice, and Barry Mullins. 2016. Using timing-based side channels for anomaly detection in industrial control systems. *International Journal of Critical Infrastructure Protection* 15 (2016), 12–26.
[15] Joel P. Dunsmore. 2020. *Handbook of Microwave Component Measurements: With Advanced VNA Techniques*. John Wiley & Sons.
[16] Thomas Eisenbarth, Christof Paar, and Björn Weghenkel. 2010. Building a side channel based disassembler. In *Transactions on Computational Science X: Special Issue on Security in Computing*, Part I. Springer, 78–99.
[17] Daisuke Fujimoto, Shota Nin, Yu-Ichi Hayashi, Noriyuki Miura, Makoto Nagata, and Tsutomu Matsumoto. 2018. A demonstration of a HT-detection method based on impedance measurements of the wiring around ICs. *IEEE Transactions on Circuits and Systems II: Express Briefs* 65, 10 (2018), 1320–1324.
[18] Yi Han, Sriharsha Etigowni, Hua Liu, Saman Zonouz, and Athina Petropulu. 2017. Watch me, but don't touch me! Contactless control flow monitoring via electromagnetic emanations. In *2017 ACM SIGSAC Conference on Computer and Communications Security*, 1095–1108.
[19] Jiaji He, Yiqiang Zhao, Xiaolong Guo, and Yier Jin. 2017. Hardware trojan detection through chip-free electromagnetic side-channel statistical analysis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 10 (2017), 2939–2948.
[20] Kengo Iokibe, Tomonobu Kan, and Yoshitaka Toyota. 2020. A study on evaluation board requirements for assessing vulnerability of cryptographic modules to side-channel attacks. In *2020 IEEE International Symposium on Electromagnetic Compatibility & Signal/Power Integrity (EMCSI)*. IEEE, 528–531.
[21] J. Edward Jackson. 2005. *A User's Guide to Principal Components*. John Wiley & Sons.
[22] Jingfei Kong, Onur Aciiçmez, Jean-Pierre Seifert, and Huiyang Zhou. 2009. Hardware-software integrated approaches to defend against software cache-based side channel attacks. In *2009 IEEE 15th International Symposium on High Performance Computer Architecture*. IEEE, 393–404.
[23] Lang Lin, Markus Kasper, Tim Güneysu, Christof Paar, and Wayne Burleson. 2009. Trojan side-channels: Lightweight hardware trojans through side-channel engineering. In *11th International Workshop on Cryptographic Hardware and Embedded Systems (CHES '09)*. Springer, 382–395.
[24] Yannan Liu, Lingxiao Wei, Zhe Zhou, Kehuan Zhang, Wenyuan Xu, and Qiang Xu. 2016. On code execution tracking via power side-channel. In *2016 ACM SIGSAC Conference on Computer and Communications Security*, 1019–1031.
[25] Saurav Maji, Utsav Banerjee, and Anantha P. Chandrakasan. 2021. Leaky nets: Recovering embedded neural network models and inputs through simple power and timing side-channels—attacks and defenses. *IEEE Internet of Things Journal* 8, 15 (2021), 12079–12092.
[26] Stephen Marsland. 2011. *Machine Learning: An Algorithmic Perspective*. Chapman and Hall/CRC.
[27] Saleh Khalaj Monfared, Tahoura Mosavirik, and Shahin Tajik. 2023. LeakyOhm: Secret bits extraction using impedance analysis. In *2023 ACM SIGSAC Conference on Computer and Communications Security*, 1675–1689.
[28] Mehari Msgna, Konstantinos Markantonakis, and Keith Mayes. 2014. Precise instruction-level side channel profiling of embedded processors. In *10th International Conference on Information Security Practice and Experience (ISPEC '14)*. Springer, 129–143.
[29] Rowshon Munny and John Hu. 2021. Power side-channel attack detection through battery impedance monitoring. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.
[30] Luong N. Nguyen, Chia-Lin Cheng, Milos Prvulovic, and Alenka Zajić. 2019. Creating a backscattering side channel to enable detection of dormant hardware trojans. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 7 (2019), 1561–1574.
[31] Luong N. Nguyen, Chia-Lin Cheng, Frank T. Werner, Milos Prvulovic, and Alenka Zajic. 2020. A comparison of backscattering, EM, and power side-channels and their performance in detecting software and hardware intrusions. *Journal of Hardware and Systems Security* 4 (2020), 150–165.
[32] Luong N. Nguyen, Baki Berkay Yilmaz, Milos Prvulovic, and Alenka Zajic. 2020. A novel golden-chip-free clustering technique using backscattering side channel for hardware trojan detection. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 1–12.
[33] Marwan Omar. 2022. *Defending Cyber Systems through Reverse Engineering of Criminal Malware*. Springer Nature.

[34] Max Panoff, Honggang Yu, Haoqi Shan, and Yier Jin. 2022. A review and comparison of AI-enhanced side channel analysis. *ACM Journal on Emerging Technologies in Computing Systems* 18, 3 (2022), 1–20.
[35] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic. 2002. *Digital Integrated Circuits*. Prentice hall.
[36] A. S. Smith and K. C. Smith. 2004. *Microelectronic Circuits* (5th. ed.). Oxford University Press.
[37] Devin Spatz, Devin Smarra, and Igor Ternovskiy. 2019. A review of anomaly detection techniques leveraging side-channel emissions. *Cyber Sensing 2019* 11011 (2019), 48–55.
[38] François-Xavier Standaert. 2010. Introduction to side-channel attacks. In *Secure Integrated Circuits and Systems*. Springer, 27–42.
[39] Daehyun Strobel, Florian Bache, David Oswald, Falk Schellenberg, and Christof Paar. 2015. Scandalee: A side-channel-based disassembler using local electromagnetic emanations. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 139–144.
[40] Pol Van Aubel, Kostas Papagiannopoulos, Łukasz Chmielewski, and Christian Doerr. 2018. Side-channel based intrusion detection for industrial control systems. In *12th International Conference on Critical Information Infrastructures Security (CRITIS '17)*, Vol. 12, Springer, 207–224.
[41] Zhenting Wang, Kai Mei, Hailun Ding, Juan Zhai, and Shiqing Ma. 2022. Rethinking the reverse-engineering of trojan triggers. In *Advances in Neural Information Processing Systems*, Vol. 35, 9738–9753.
[42] W. Wolf. 2004. *FPGA-Based System Design*. Pearson Education.
[43] Jean-Paul A. Yaacoub, Ola Salman, Hassan N. Noura, Nesrine Kaaniche, Ali Chehab, and Mohamad Malli. 2020. Cyber-physical systems security: Limitations, issues and future trends. *Microprocessors and Microsystems* 77 (2020), 103201.
[44] Yuan Yao, Mo Yang, Conor Patrick, Bilgiday Yuce, and Patrick Schaumont. 2018. Fault-assisted side-channel analysis of masked implementations. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 57–64.
[45] Fengchao Zhang, Andrew Hennessy, and Swarup Bhunia. 2015. Robust counterfeit PCB detection exploiting intrinsic trace impedance variations. In *2015 IEEE 33rd Vlsi Test Symposium (Vts)*. IEEE, 1–6.
[46] YongBin Zhou and DengGuo Feng. 2005. Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. *Cryptology ePrint Archive*. Retrieved from https://eprint.iacr.org/2005/388
[47] Huifeng Zhu, Haoqi Shan, Dean Sullivan, Xiaolong Guo, Yier Jin, and Xuan Zhang. 2023. PDNPulse: Sensing PCB anomaly with the intrinsic power delivery network. *IEEE Transactions on Information Forensics and Security* 18 (2023), 3590–3605.

Received 9 September 2024; revised 9 June 2025; accepted 22 August 2025

SHIFT: Selective Hardware Information Flow Tracking Driven by Deterministic Constraints

HAODONG SUN and ZHI YANG, PLA Information Engineering University, Zhengzhou, China
SHUYUAN JIN, Sun Yat-sen University, Guangzhou, China
ZHENLONG ZHANG, PLA Information Engineering University, Zhengzhou, China

Information flow tracking technology is commonly used in the security analysis of hardware design. This technology protects the confidentiality and integrity of essential assets by instrumenting trace logic on each operation unit to detect whether critical information has been leaked or tampered with. However, as hardware design becomes increasingly large-scale and complex, the significant performance overhead introduced by instrumentation has become a major challenge. This article proposes Selective Hardware Information Flow Tracking (SHIFT), a constraint-driven optimization technique. The core idea of SHIFT includes selective monitoring of operations and selective optimization of propagation logic. In the intermediate representation of the hardware design, SHIFT scans taint sources in the code statically using a conservative analysis algorithm to determine whether logic structures require monitoring and assigns optimization tags based on known conditions. During the synthesis process, these optimization tags are passed to the netlist, thereby enabling selective instrumentation of the trace logic on the cell. In the Trust-Hub AES test bench, SHIFT reduces the deployment time of the tracking model by 12.1%, decreases the number of cells by 19.9%, and reduces the synthesized area by 35.7%, Additionally, the security verification time of the flow model was reduced by 10.5%. In general, SHIFT reduces the overhead of deploying trace logic without introducing false positives.

CCS Concepts: • **Security and privacy → Security in hardware**;

Additional Key Words and Phrases: Hardware Security, Dynamic Analysis, Information Flow Tracking, Model Optimization

ACM Reference format:
Haodong Sun, Zhi Yang, Shuyuan Jin, and Zhenlong Zhang. 2025. SHIFT: Selective Hardware Information Flow Tracking Driven by Deterministic Constraints. *ACM J. Emerg. Technol. Comput. Syst.* 21, 4, Article 13 (October 2025), 16 pages.
https://doi.org/10.1145/3765906

1 Introduction

For a long time, hardware design in the industry has primarily focused on ensuring correct functionality and computational performance, while neglecting the potential for design flaws and

This work was supported by the National Natural Science Foundation of China under Grants 62176265 and 62472456.
Authors' Contact Information: Haodong Sun, PLA Information Engineering University, Zhengzhou, China; e-mail: sunhd365@163.com; Zhi Yang (corresponding author), PLA Information Engineering University, Zhengzhou, China; e-mail: zynoah@163.com; Shuyuan Jin, Sun Yat-sen University, Guangzhou, China; e-mail: jinshuyuan@mail.sysu.edu.cn; Zhenlong Zhang, PLA Information Engineering University, Zhengzhou, China; e-mail: zhenl0ng_zhang@sina.com.

ACM 1550-4840/2025/10-ART13
https://doi.org/10.1145/3765906

malicious vulnerabilities. Design flaws inadvertently introduced during the development process can compromise the security of the entire hardware system. For instance, Meltdown [17] and Spectre [14] in 2018, along with various attack variants, have exposed mainstream processor to risks by exploiting security flaws in the hardware design. Concurrently, as industrial companies increasingly outsource hardware design to third parties to meet growing market demands, the third parties may insert hardware trojans into design with ulterior motives, leading to cause catastrophic system failures. Above security issues pose significant threats to critical sectors such as military, aerospace, and communications, drawing increasing attention from both academia and industry to the security analysis in hardware design.

Hardware Information Flow Tracking (HIFT) is utilized to detect and analyze information flows within hardware designs. By monitoring the state of critical assets in a flow tracking model and assessing compliance with relevant security policies, HIFT can identify unintentional design flaws, malicious circuit modifications, timing side channels, access control violations, and other insecure hardware behaviors [11]. However, implementing the **Information Flow Tracking (IFT)** model necessitates the integration of trace logic into each operational unit, resulting in considerable deployment overhead.

Raising the abstraction layer of IFT can effectively reduce instrumentation overhead. However, it may compromise the precision of information flow propagation. Additionally, the complex semantics associated with high abstraction layers increase the difficulty in automating the instrumentation of trace logic. To mitigate the instrumentation overhead associated with low-abstraction layer flow tracking techniques, common optimizations include employing inaccurate trace logic or reducing the hardware design area of tracking. However, the former approach may introduce false positives that burden security personnel, while the latter may overlook critical dataflow behaviors. To address these challenges, we propose **Selective Hardware Information Flow Tracking (SHIFT)** based on deterministic constraints. This technique converts deterministic constraints into optimization tags for the intermediate hardware design structure by employing static analysis algorithms. It propagates these optimization tags to the cell during synthesis and ultimately inserts lightweight trace logic according to optimization tags. We preserve the original detection precision while reducing hardware instrumentation overhead, without introducing additional false positives. Specifically, we present the following contributions.

- We introduce SHIFT, the first SHIFT technique based on optimization tags, which reduces deployment overhead while ensuring no new false positives;
- We propose a static analysis algorithm for high-level logical structures that conservatively identifies logical structures that never propagate taint and assigns corresponding optimization tags to logical structures based on known conditions;
- SHIFT framework has been restructured using the open source hardware design synthesis tool Yosys. It incorporates lightweight tracking logic under various constraints into the original information flow trace logic library. Experiments were conducted on common cryptographic cores and other functional components to demonstrate its precision and lightness.

The following sections of this article are summarized below. Section 2 provides a brief review of related work in the field of HIFT. Section 3 introduces the trace logic optimization principles and the model generation process of SHIFT. Section 4 details the specific implementation of SHIFT's deployment of IFT models. Section 5 presents the security verification results and assesses the overhead for mainstream hardware design. Finally, Section 6 discusses the summary and outlook of this article.

2 Related Work

This section provides a concise overview of the research advancements in HIFT (Section 2.1), the trace logic with different accuracies (Section 2.2), and the optimization of IFT (Section 2.3).

2.1 HIFT

Denning [9] first introduced the concept of IFT in 1976. IFT is a security analysis technique that creates an information flow model for the target system and analyzes how information propagates within it. Security labels are initially assigned to the information-carrying objects in the system to represent specific security attributes, and then security labels trace logic is introduced to enable tracking from the sensitive port to the release port. Security researchers can monitor the status of the output security labels and then determine whether the information complies to the specified security policy.

The confidentiality of the system is ensured by monitoring the status of security labels in the public domain to determine whether sensitive information has leaked into unclassified areas. The integrity of the system is verified by assessing whether the security label of an untrusted entity affects the label of a trusted entity, thereby indicating whether the trusted entity has been compromised.

Hardware information flow tracing can be employed to detect security vulnerabilities, including design flaws and bypass channels [14, 17, 27]. This approach provides an effective solution for verifying and testing hardware security. To achieve a balance between precision and scalability in hardware information flow, prior research has proposed security analysis at various abstraction levels of hardware design. These levels encompass gate-level [13, 19, 20, 26], cell-level [23], RTL level [4, 5], HDL level [3, 15, 16, 21, 28], and HLS level [8, 10, 22].

Information flow models implemented at various abstraction layers have different detection precision and verification performance. Gate-level information flow technology [26] is implemented by inserting trace logic into the gate-level netlist, providing high-precision tracking of information flow. However, the complexity of the model results in low verification performance, fails to meet scalability requirements, and presents certain issues with false positives. RTL level information flow technology [4] focuses on the integration of trace logic within **Hardware Description Languages (HDLs)**. This approach improves verification performance while maintaining precision. However, the complexity and abundance of high-level language structures may present challenges in fully processing these statements. Cell-level information flow technology [23] is deployed on the macrocell netlist between the above two abstraction layers and performs well in precision and verification performance.

2.2 Accuracy and Complexity of Trace Logic

HIFT technology integrates trace logic into the operational unit that processes hardware signals. Consider a two-input AND logic cell in the tracking model, where a_t, b_t, and o_t represent the security labels of input and output, respectively. When the data input to port a contains confidential information, set a_t to 1. The security label o_t can be expressed using the appropriate trace logic according to the desired precision.

Information flow propagation logic can be represented as $o_t = a_t \mid b_t$, which conservatively captures all potential confidential information flows from input ports a or b to output port o. However, this propagation logic may be imprecise. For instance, if port a carries public data 0, which is a_t is 0 and port b carries confidential information 1, means b_t is 1, the output o will be dominated by a, and the other confidential input b has no effect on the output o. However,

this propagation logic conservatively expressed $o_t = 1$, classifying the output port as confidential. This conservative approach results in a significant number of false alarms, thereby increasing the workload for security personnel.

CellIFT [23] and GLIFT [26] provide a more precise method for computing the security label of output. The flow of information from input to output occurs only when the input value influences the output value. For an AND logic cell, its output port o security label can be expressed as $o_t = b \,\&\, a_t \mid a \,\&\, b_t \mid a_t \,\&\, b_t$. According to the above input values and security label, it is deduced that the output port security label o_t is 0, indicating that the exact propagation logic does not produce false alarms.

The higher the required tracking precision, the more complex the propagation logic becomes. To address this, Hu et al. [12] propose relaxing precision by progressively disregarding certain inputs, thereby moving toward a lower level of precision. This optimization results in a loss of precision in the propagation of hardware information flow. However, under specific constraints, functionally equivalent substitutions between imprecise and precise propagation strategies can be achieved.

2.3 IFT Optimization

Although hardware information flow tracing techniques can effectively detect logical vulnerabilities and verify the security properties of hardware designs, their inherent instrumentation overhead hinders the widespread application of this technology in the industry. In recent years, numerous researchers have proposed various optimization methods to reduce deployment overhead. These optimization techniques are generally classified into two categories: optimization of tracking logic and instrumentation areas.

Optimization of the Instrumentation Area: If static analysis can identify hardware regions that will never be involved in information flow propagation, instrumentation in those areas can be omitted. Previous work on optimizing IFT, conducted by Nicholas et al. [18], reduced deployment overhead by only tracking data from safety-critical modules. However, this granularity is too coarse, and the optimization remains insufficient. Chen et al. [6] reduced the deployment overhead of IFT in SelectiveTaint by dividing the must-not-taint directive. Zhang et al. [29] proposed the HardTaint optimization technique, which employs selective instrumentation and static analysis to achieve efficient hardware information flow tracing at the instruction architecture abstraction layer. The aforementioned methods utilize static analysis at various fine-grained levels to optimize information flow tracing within the instrumentation region. For large hardware designs, optimizing the instrumentation area can significantly reduce the deployment overhead associated with IFT.

Optimization of Trace Logic: Set lightweight information flow trace logic to reduce the complexity of the tracking model. Hu et al. [12, 24, 25] argue that the GLIFT logic for tracking information flow propagation contains redundancy under certain conditions. In the case of setting don't care input, replacing the complex GLIFT logic with simple propagation logic will not affect the precision. However, if the inputs influence the propagation of information flow as a don't care condition, the inaccurate propagation resulting from the substitution will introduce new false positives. Chen et al. [7] successfully simplified the tracking logic of multi-input cells from exponential complexity to linear complexity by introducing a "-" logic, which converts sum-of-products logic into product-of-sums logic. However, this method is not applicable to standard logic cells. Table 1 summarizes and compares the current work on IFT optimization.

3 Selective IFT

This section presents the deterministic constraint-driven IFT logic (Section 3.1) and the generation of tag-based selection information flow models (Section 3.2).

Table 1. Current Work on Information Flow Tracking Optimization

Method	Abstraction Level	Category	Precision	Overhead
[6]	Binary level	*IA*	Accurate	Complex
[29]	Binary level	*IA*	Accurate	Complex
[18]	Gate level	*IA*	Ignore information flows from other modules	Lightweight
[24, 25]	Gate level	*TL*	Introduce false positives	Lightweight
[12]	Gate level	*TL*	Imprecision (overtainting)	Complex
[7]	Gate level	*TL*	Imprecision (overtainting); Applicable in certain scenarios	Lightweight
SHIFT	RTL level	*TL and IA*	Accurate	Lightweight

TL, optimization of tracking logic; IA, optimization of instrumentation area.

3.1 IFT Logic

3.1.1 Precision of Trace Logic. By deploying information flow trace logic within the macrocell of hardware design, an information flow model is constructed to verify the system's confidentiality and integrity. This model assesses whether input signals influence output signals, and the precision of the trace logic affects the detection precision of the model. The imprecise information flow model employs a conservative least upper bound operator to determine the information flow, which can be represented as Equation (1):

$$L(O) = L(I_1)|L(I_2)|...|L(I_n). \tag{1}$$

The security labels of the outputs L(O) represent the OR operation of all input security labels, thereby implementing a lightweight yet imprecise information flow trace logic. Once the hardware design is synthesized into a netlist, the corresponding information flow trace logic is integrated. In the trace logic described in Equation (1), each netlist element incurs the overhead of one OR gate, resulting in a total overhead of n cells for a hardware design with n cells after synthesis.

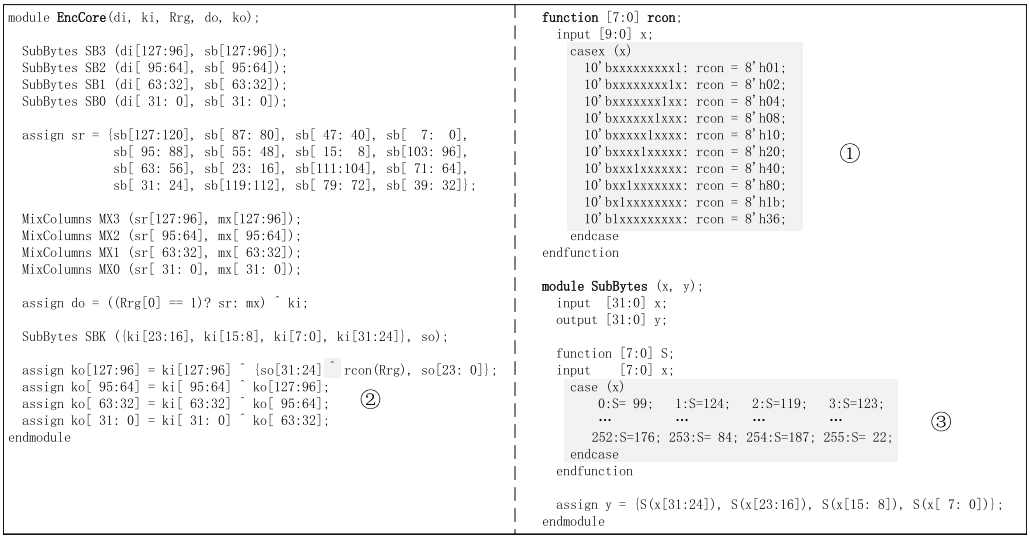
The information flow trace logic presented in Equation (1) facilitates the rapid verification of potential security vulnerabilities with minimal instrumentation overhead. However, conservative verification results can lead to false positives, and troubleshooting these issues adds extra work for security analysts.

Precise information flow trace logic not only considers the security label associated with the input values but also incorporates the signal values I_n of the input into the propagation logic. In this trace logic, the security label L(O) is influenced by both the input value and the input security label. The precise information flow trace logic can be expressed as Equation (2):

$$L(O) = f\{I_1, I_2, ...I_n, L(I_1), L(I_2), ..., L(I_n)\}. \tag{2}$$

Trace logic can accurately derive the information flow propagation based on different input values and security labels, but it introduces additional instrumentation overhead and verification time. The hardware design, after synthesizing into a netlist of n cells, has a total overhead of n^*x for the information flow model for the tracking logic of Equations (2) and (3), with an average cell-instrument overhead of x.

Unlike the OR operation, operation in Equations (1) and (2) introduces an information flow propagation function f based on the propagation characteristics of logic cells, enabling accurate but more complex IFT logic. However, this method only inserts different trace logic according to the cell type, providing generalized precision in IFT, which still results in significant redundancy overhead during actual deployment.



```
module EncCore(di, ki, Rrg, do, ko);

  SubBytes SB3 (di[127:96], sb[127:96]);
  SubBytes SB2 (di[ 95:64], sb[ 95:64]);
  SubBytes SB1 (di[ 63:32], sb[ 63:32]);
  SubBytes SB0 (di[ 31: 0], sb[ 31: 0]);

  assign sr = {sb[127:120], sb[ 87: 80], sb[ 47: 40], sb[  7:  0],
               sb[ 95: 88], sb[ 55: 48], sb[ 15:  8], sb[103: 96],
               sb[ 63: 56], sb[ 23: 16], sb[111:104], sb[ 71: 64],
               sb[ 31: 24], sb[119:112], sb[ 79: 72], sb[ 39: 32]};

  MixColumns MX3 (sr[127:96], mx[127:96]);
  MixColumns MX2 (sr[ 95:64], mx[ 95:64]);
  MixColumns MX1 (sr[ 63:32], mx[ 63:32]);
  MixColumns MX0 (sr[ 31: 0], mx[ 31: 0]);

  assign do = ((Rrg[0] == 1)? sr: mx) ^ ki;

  SubBytes SBK ({ki[23:16], ki[15:8], ki[7:0], ki[31:24]}, so);

  assign ko[127:96] = ki[127:96] ^ {so[31:24] ^ rcon(Rrg), so[23: 0]};
  assign ko[ 95:64] = ki[ 95:64] ^ ko[127:96];
  assign ko[ 63:32] = ki[ 63:32] ^ ko[ 95:64];
  assign ko[ 31: 0] = ki[ 31: 0] ^ ko[ 63:32];
endmodule
```

②

```
function [7:0] rcon;
  input [9:0] x;
    casex (x)
      10'bxxxxxxxxx1: rcon = 8'h01;
      10'bxxxxxxxx1x: rcon = 8'h02;
      10'bxxxxxxx1xx: rcon = 8'h04;
      10'bxxxxxx1xxx: rcon = 8'h08;
      10'bxxxxx1xxxx: rcon = 8'h10;
      10'bxxxx1xxxxx: rcon = 8'h20;
      10'bxxx1xxxxxx: rcon = 8'h40;
      10'bxx1xxxxxxx: rcon = 8'h80;
      10'bx1xxxxxxxx: rcon = 8'h1b;
      10'b1xxxxxxxxx: rcon = 8'h36;
    endcase
endfunction

module SubBytes (x, y);
  input  [31:0] x;
  output [31:0] y;

  function [7:0] S;
  input    [7:0] x;
    case (x)
        0:S= 99;   1:S=124;   2:S=119;   3:S=123;
        ...        ...        ...        ...
      252:S=176; 253:S= 84; 254:S=187; 255:S= 22;
    endcase
  endfunction

  assign y = {S(x[31:24]), S(x[23:16]), S(x[15: 8]), S(x[ 7: 0])};
endmodule
```

①

③

Fig. 1. Verilog code of AES cryptographic core.

3.1.2 Tracking Logic Based on Constraints. Equation (2), which represents the precise propagation of information flow, is utilized to calculate the propagation results under the uncertainty of all independent variable elements. However, when deterministic constraints are present for the independent variables, the complex trace logic can be substituted with a simpler equivalent. This observation motivates us to optimize IFT techniques: equivalent tracking can be accomplished using simplified trace logic under specific conditions. By introducing constraints on the inputs, the tracking logic can be streamlined. The exact IFT logic based on these constraints can be abstracted as Equation (3):

$$L(O) = f\{I_1, I_2, ...I_n, L(I_1), L(I_2), ..., L(I_n)\} \ \& \ I_i | L(I_i) == const. \tag{3}$$

To illustrate the logic of the constraint-based IFT, we use part of the AES code in Figure 1.

When certain variables behave as constants or exhibit constrained relationships with one another during the IFT process, the corresponding product terms in Equation (3) can be simplified. For instance, consider the AES encryption core illustrated in Figure 1. Optimization techniques can be categorized into region-based selective deployment and cell-based selective instrumentation, based on the variations in deterministic constraints.

For example, in the corresponding logical structure *r*con (Rrg) of ①, when the hardware design reads the memory data from an uncontaminated address, since the data stored in the memory are constants, i.e., the security labels of the corresponding logical cell inputs are LOW, indicating that no sensitive information will be output. In this case, there is no need to deploy the IFT model for the region, and the optimization falls under region-based selective deployment. Similarly, in areas ② and ③, the optimization of logic cells and logic structure of high abstraction layer are presented, respectively. Consider an OR cell where the security label of input port is constantly LOW. The information flow trace logic for the corresponding cell can be simplified by applying the constraints, making this optimization fall under the cell-based selective instrumentation.

After the hardware design is synthesized into a netlist of n cells, with p cells located in the uncontaminated region (similar to ①) and q cells in the logic-optimized region (similar to ② and ③), the total overhead of the information flow model, given an average cell-instrument overhead of x and an optimization coefficient of k, is $(n - p - q) * x + qkx$. Compared to Equation (1), this method

```
sig-const    ::= high, low
taint-const  ::= taint, untaint
t-var        ::= sig_i ∈ sig-const | taint_i ∈ taint-const
binary-opr   ::= '|' | '&'
unary-opr    ::= '~'
t-opr        ::= binary-opr | unary-opr
opt-tag      ::= Constraint information for the static analysis phase
t-expr       ::= [t-var | t-expr{t-var t-opr} | '(' t-expr ')' | t-opr t-expr]opt
assign-opr   ::= '='
t-statement  ::= t-var assign-opr t-expr
```

Fig. 2. Formal representation of information flow model generation.

ensures the precision of the tracking logic; in contrast to Equation (2), this method ensures the efficiency of the tracking logic. By leveraging the collected deterministic constraints, the tracking logic for the current operation can be simplified, thereby reducing the deployment overhead of the information flow model while maintaining propagation precision.

3.2 Model Generation Based on Optimization Tag

The selectivity of SHIFT is manifested during the instrumentation phase, where the optimization information collected in the static phase determines whether the cell should be deployed with generic or optimized trace logic. We store the constraint information as an optimization tag to facilitate the generation of the information flow model. SHIFT instruments each macrocell netlist, and the formal representation of the information flow model is illustrated in Figure 2.

$sig\text{-}const$ and $taint-const$ represent the set of values of signals and corresponding security labels. $t-var$ represents a certain number of bits of signals or security labels, which together form the variables in the information flow trace logic $t-expr$, $assign-opr$, $binary-opr$, and $unary-opr$ are the operators in the information flow propagation logic, which are used to compose variable operation relations. The information flow tracing modeling process for hardware designs, although the information flow trace logic $t-expr$ is different for different types of logic cells, all of them consist of the variable $t-var$ and the operator $t-opr$ with parentheses used to prioritize the operators. The information in $opt-tag$ is derived from static analysis, and constraint information is applied in $t-expr$ to simplify the trace logic. $t-expr$ can also be represented in nested representations to implement complex information flow trace logics. The statement $t-statement$ semantically completes the process of modeling the information flow tracing of a logic cell, i.e., assigning the results of the propagation logic to the security labels corresponding to the output signals of the logic cell.

Consider the AND logic cell as an example. When signal A is a sensitive signal, it can propagate sensitive information to the output signal Y after passing through the AND cell. The selective information flow model of the AND logic cell, based on optimized tags, is illustrated in Figure 3. In the absence of constraints, the IFT is denoted as in the left figure: $Y_{taint} = A_{sig}\ \&\ B_{taint} \mid A_{taint}\ \&\ B_{sig} \mid A_{taint}\ \&\ B_{taint}$. However, when variable constraints are applied, the IFT logic of the AND cell is simplified.

The optimization tag in the above figure considers the constraints of A_{taint}. When A_{taint} behaves as a constant zero, the original information flow trace logic under this constraint simplified to the one in the right figure: $Y_{taint} = A_{sig}\ \&\ B_{taint}$. During instrumentation, the corresponding trace logic is deployed through tag, to optimize the tracking model of the AND logic cell. Note that the schematic only considers the case of A_{taint}, but in the actual deployment, all the input constraints are considered and the trace model optimization is performed.

SHIFT traverses the microcell netlist of the entire hardware design, identifies the type of each logical cell, and instruments information flow trace logic based on the optimization tag. The IFT

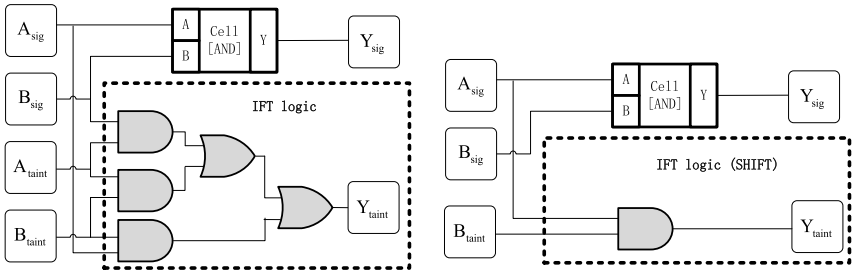


Fig. 3. Information flow tracking model for AND.

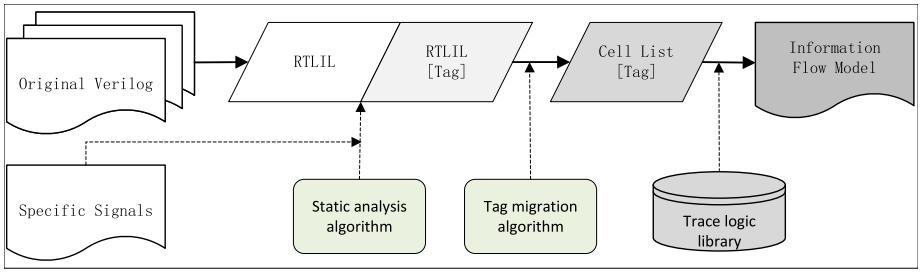


Fig. 4. The framework flow of SHIFT.

model of the entire hardware design is constructed from the bottom up, enabling information flow traceability across the netlist.

4 SHIFT Implementation

The framework flow of the SHIFT optimization technique is divided into three parts, as illustrated in Figure 4. First, the tag generation algorithm operates on the coarse-grained **Register-Transfer-Level Intermediate Language (RTLIL)** abstraction layer. Next, the tag migration algorithm is employed during the synthesis process from RTLIL to the Cell List. Finally, a selective instrumentation algorithm is applied to the logical netlist. The core concept of the framework is to optimize the propagation logic at the lower abstraction layer by utilizing the constraint information collected from the higher abstraction layer.

4.1 Tag Generation Algorithm for RTLIL

To address the complexities associated with HDL, which can be challenging to analyze and process, we employ Yosys' intermediate representation, RTLIL, to perform automated and efficient static analysis. The RTLIL::Design structure is utilized to represent the hardware design, which consists of several RTLIL::Module structures. These modules incorporate RTLIL::Cell, RTLIL::Process, and RTLIL::Memory structures as high-level logical components to depict the hardware design. The Label Generation Algorithm 4.1 is implemented to statically analyze and assign corresponding optimization tags to the high-level logical structures.

The algorithm begins by preprocessing the high-level structures, such as Process and Memory, of the top-level module while collecting its input and output information. It then determines the initial value of the safety label for the input signal and identifies the "tainted region" based on the propagation of tainted signals, as well as the "non-tainted region" based on the propagation of non-tainted signals. Additionally, it optimizes the non-tainted region starting from the top-level module.

Algorithm 4.1: Algorithm for Generating Tag

```
Input: The RTLIL represent of Hardware Design
Input: source_taint_signal, source_untaint_signal
Output: Yosys RTLIL that carries Optimized Tag
Ⅰ: IR_inoutput_init()
 1:  Process_init_process(proc, proc_input, proc_output)
 2:  Memory_init_process(mem, mem_input, mem_output)
Ⅱ: MOD_flag_process()
 1:  source_taint_sig → taint_sig_set
 2:  source_untaint_sig → untaint_sig_set
 3:  set taint_sig_num, untaint_sig_num = 0;
 4:  set area_opt_flag, port_opt_flag = false;
 5:  while (taint_sig_set.size >= taint_sig_num) do
 6:    taint_sig_set.size → taint_sig_num
 7:    if (area_opt_flag == true) untaint_sig_set.size → untaint_sig_num
 8:    for [proc_it][mem_it] ∈ current_module<process>< memory> do
 9:      if ([proc_it)[mem_it]->get_attribute(taint_area) == true) continue
10:     else if (area_opt_flag == false)
11:       for input_it ∈ [proc_input][mem_input] do
12:         if (taint_sig_set.count(it_input) ≠ null)
13:           for output_it ∈ [proc_output][mem_output] do
14:             output_it → taint_signal_set
15:           [proc_it][mem_it]->set_attribute(taint_area, true)
16:     else if (area_opt & ! [proc_it][mem_it].get_attribute(taint_area))
17:       for output_it ∈ [proc_output][mem_output] do
18:         output_it → untaint_sig_set
19:       [proc_it][mem_it]->set_attribute(untaint_area, true)
20:   for [cell_it] ∈ current_module<cell> do
21:     if ([cell_it]->get_attribute(taint_area) & ! area_opt_flag) contuine
22:       for sig_it ∈ cell_it<connections> do
23:         if (sig_it ∈ cell_it.output)  sig_it.wire->name → out_name
24:         if (! area_opt_flag & ! taint_signal_set.count(sig_it.wire->name))
25:           cell_it->set_attribute(taint_area, true)
26:           out_name → taint_signal_set
27:         if (area_opt_flag & ! untaint_signal_set.count(sig_it.wire->name))
28:           if (cell_it->get_atttribute(taint_area) != true)
29:             cell_it->set_attribute(untaint_area, true)
30:             out_name → untaint_signal_set
31:         else cell_it->set_attribute(port_opt, sig_it.first)
32:   if (taint_signal_num == taint_signal_set.size())
33:     set area_opt_flag = true;
34:     if (untaint_signal_num == untaint_sig){
35:       process_internal_mod();
36: end
```

This process is repeated for any modules that require further analysis. The specific implementation details of the tag generation algorithm for RTLIL are shown in Algorithm 4.1.

The algorithm receives the RTLIL representation of the hardware design along with the initial values of the security labels provided by the security testers. It first analyzes and processes the top-level modules. The input and output information of the high-level structure is stored in the "IR_inputput_init()" function. Taint labels are stored, and the associated values are initialized in

lines 1–3 of the "Mod_flag_process()" function. Starting from line 4, the processing logic for tag generation is initiated, traversing the various structures in the RTLIL representation sequentially. Lines 7–14 of the algorithm traverse the Process and Memory structures according to the taint set, while lines 19–25 traverse the Modular Cell and Logical Cell structures based on the same taint set. When the input signals of a structure contain tainted signals, the output signals are added to the tainted set, and tag information is assigned to the corresponding structure. This operation is repeated until the number of taint sets no longer increases, indicating that the "non-tainted area" has been maximized to cover the RTLIL structure. At this point, line 32 of the algorithm sets the "area_opt_flag" to high.

Lines 15–18 of the algorithm traverse the Process and Memory structures based on the non-tainted set, while lines 26–30 traverse the Modular Cell and Logical Cell structures according to the same non-tainted set. When the inputs of the structures contain non-tainted signals, the output signals are added to the non-tainted set, and the attribute information is updated in the corresponding structures. This process is repeated until no further additions can be made to the non-tainted set, indicating that the "non-tainted area" has been maximized to encompass the RTLIL structure.

The function "process_internal_mod()" on line 34 of the algorithm recursively processes the aforementioned algorithm for modular Cell that requires further analysis. Optimization tags for RTLIL structures in the hardware design are generated from the preceding static analysis. The high-level language constructs of Modular Cell, Process, and Memory offer an appropriate level of granularity for analysis, enabling efficient and lightweight delineation of unreachable tainted regions and optimization of tainted areas.

For instance, after analyzing the hardware design using Yosys, there are N logical structures, which include N_{mem} Memory, N_{mod} module instances, N_{proc} Process, and N_{cell} macrocells. SHIFT traverses these structures sequentially based on the taint labels provided by security personnel. In the best-case scenario, the N logical structures are arranged in the order of taint propagation, resulting in a time complexity of $O(n)$. Conversely, in the worst-case scenario, the N logical structures are arranged in reverse order of taint propagation, leading to a time complexity of $O(n^2)$. The arrangement of logical structures is determined by the synthesis algorithm of Yosys, with an average time complexity of $O(n^2)$.

4.2 Migration of Optimization Tag

After the hardware design has been processed by the tag generation algorithm, only the RTLIL::Cell, RTLIL::Process, and RTLIL::Memory structures retain information about optimization tags. However, during Yosys' synthesis process, the Process and Memory structures are ultimately decomposed into RTLIL::Cell and RTLIL::Wire. This decomposition necessitates the transfer of optimization tags from the high-level structures to the cell structure. Figure 5 illustrates the migration of optimized labels.

The Process structure is processed by Pass::proc into a unit structure consisting of mux and triggers, and SHIFT passes the optimization tags carried by the Process to the synthesized cell structure by adding the proc code. For example, when the Process belongs to "non-tainted area," the corresponding logical cell after synthesis carries the corresponding optimization tag.

The Memory structure is processed by Pass::mem into mux and other cell structure, SHIFT passes the optimization tag of Memory structure to the synthesized cell structure. For instance, when Memory belongs to "tainted area," i.e., when its read/write or address signals contain tainted information, but since Memory reads memory data as constants, the corresponding logical cell after synthesis carries the corresponding optimization tag.

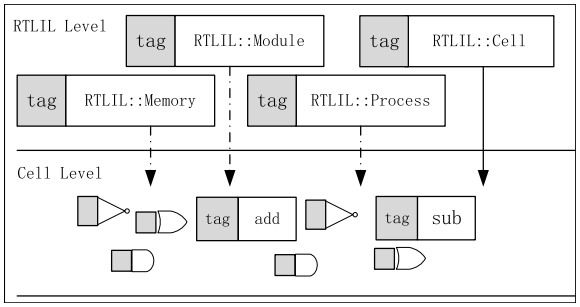


Fig. 5. Optimized label passing from high-level structure to cell.

Algorithm 4.2: Algorithm for Instrumenting Taint Logic

```
1: Input: The Yosys IR represent of Hardware Design
2: output: Information Flow Tracking Modle
3: Cellift_MUX(module, cell) begin
4:     ports[4] = {cell->get_port(A), cell->getPort(B), cell->getPort(S), cell->getPort(Y)};
5:     port_taints[i] = get_corresponding_label_signals(module, ports[i]);
6:     extend_s = RTLIL::SigSpec(RTLIL::SigBit(ports[S], data_size));
7:     if(cell->get_attribute(ID(clean)) == true){
8:         module->connect(port_taints[A], port_taints[Y]);}
9:     else if(cell->get_attribute(ID(mem_opt)) == true){
10:        module->addXor(NEW_ID, ports[A], ports[B], port_taints[Y]);}
11:    else{
12:        not_s = module->Not(NEW_ID, extended_s);
13:        not_s_or_s_taint = module->Or(NEW_ID, extended_s_taint, not_s);
14:        s_or_s_taint = module->Or(NEW_ID, extended_s_taint, extended_s);
15:        a_t_and_not_s_or_s_t = module->And(NEW_ID, extended_a_taint, not_s_or_s_taint);
16:        b_t_and_s_or_s_t = module->And(NEW_ID, extended_b_taint, s_or_s_taint);
17:        data_taint = module->Or(NEW_ID, a_t_and_not_s_or_s_t, b_t_and_s_or_s_t);
18:        a_xor_b = module->Xor(NEW_ID, ports[A], ports[B]);
19:        s_t_and_a_xor_b = module->And(NEW_ID, extended_s_taint, a_xor_b);
20:        module->addOr(NEW_ID, s_t_and_a_xor_b, data_taint, port_taints[Y]);}
30: end
```

The hardware design after the above synthesis consists only of the cell structure, i.e., the optimization tags are migrated from the high-level structure to the RTLIL::Cell structure.

4.3 Conditional Instrumentation of Cells

The optimization tags obtained at the high abstraction layer are passed to the logical netlist abstraction layer after synthesis, where SHIFT deploys taint propagation logic based on these optimization tags. Unlike previous deployments of the same taint trace logic based on the type of logical cell, SHIFT applies different levels of optimization to logical cell of the same type based on the optimization tag. This approach selects the logical expression with the lowest overhead while maintaining the required propagation precision. Algorithm 4.2 outlines the SHIFT instrumentation process for the MUX logic.

Lines 4–6 of the algorithm generate and process the tainted ports of the Mux cell. Starting from line 7, the corresponding taint propagation logic is deployed based on the optimization tags passed down from the high abstraction layer. For instance, line 8 generates and processes the taint

Fig. 6. SHIFT confidentiality security vulnerability detection.

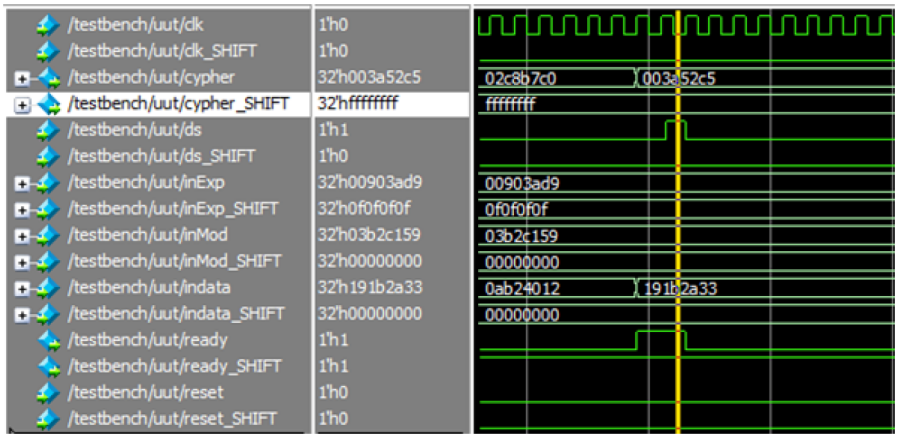


Fig. 7. SHIFT integrity security vulnerability detection.

propagation logic by determining whether the Mux logic cell possesses the "clean" optimization tag, if the tag exists, the Mux cell is located in the taint unreachable region, meaning there is no need to deploy taint propagation logic. In this case, the input security label is directly connected to the output security label. Line 9 deploys the corresponding accurate and lightweight taint propagation logic by determining the "mem_opt" optimization tag. If no optimization tag is present, the generic taint propagation logic of the Mux cell is shown in lines 11–20 code.

SHIFT traverses the logical netlist of the entire hardware design, identifies the type of each logical cell, and applies IFT logic based on optimization tags. The tracking model of the entire hardware design is constructed from the bottom up, thus realizing the ability of the hardware design to track tainted information on the logical netlist. Then tag the key assets in the design and observe the security labels output from the release point to analyze whether the hardware design satisfies the specified security policy, thus realizing the verification of the security attributes of the hardware design.

5 SHIFT Experiments and Analysis

This section uses hardware design from Trust-Hub [2] and OpenCores [1] to evaluate the precision and efficiency of SHIFT. The precision primarily verifies whether SHIFT matches the detection capability of CellIFT in terms of confidentiality and integrity, as detailed in Section 5.1. Section 5.2 compares SHIFT and CellIFT in terms of the number of synthesized cells, synthesis time, synthesized area, and the verification time, further demonstrating the efficiency of SHIFT.

5.1 Precision of SHIFT

The confidentiality of the hardware design requires that sensitive information not flow to publicly visible output ports. In the Trust-Hub test benchmark AES_T100, the Trojan uses a pseudo-random number generator to create CDMA sequences and keys for dissimilarity operations, and the

Benchmark	Trojan behavior	security property	Detect result	
			CellIFT	SHIFT
AES-T100	Leaks the key through covert CDMA channel	Confidentiality	✓	✓
AES-T200	Leaks the key through covert CDMA channel	Confidentiality	✓	✓
AES-T400	Leaks the key through antenna channel	Confidentiality	✓	✓
AES_T900	Leaks the key through covert CDMA channel	Confidentiality	✓	✓
AES_T1200	Leaks the key through covert CDMA channel	Confidentiality	✓	✓
AES_T1600	Leaks the key through antenna channel	Confidentiality	✓	✓
BasicRSA_T100	Leaks the key through ciphertext	Confidentiality	✓	✓
BasicRSA_T200	Replaces the key to disable encryption	Intergrity	✓	✓
BasicRSA_T300	Leaks the key through ciphertext	Confidentiality	✓	✓
BasicRSA_T400	Replaces the key to leak plaintext	Confidentiality; Intergrity	✓	✓

Fig. 8. Detection precision of SHIFT under common cipher benchmarks.

generated sequences are forwarded to the output ports of the leaked circuits, thus compromising the data confidentiality of the hardware design. By setting the bits of the corresponding security label of the key to HIGH, the flow of the key information in the hardware design can be effectively observed. As shown in Figure 6, when the key leaks to the output port Capacitance of the covert channel through the above Trojan horse, the tainted label of the corresponding signal Capacitance_SHIFT is shown to be HIGH, thus capturing the confidentiality violation of the hardware design.

In the same way as the above detection principle, SHIFT can also be used to detect data integrity, where the integrity of the hardware design requires that data from untrustworthy domains cannot affect the system. In the Trust-Hub test benchmark BasicRSA_T200, the Trojan uses constants to replace the inExp parameter, and the breakdown of data integrity prevents the output ciphertext from being decrypted properly. As shown in Figure 7, a common detection method for related Trojans is to set HIGH to the security label of external inputs or important assets and observe whether the security label cypher_SHIFT of the ciphertext output cypher is HIGH or not, thus determining the integrity violation of confidential information.

Benchmark	Cell Number ($unit$)		Synthesis Time (s)		Cell Area (um^2)			Verification Time (ns)		
	SHIFT	CellIFT	SHIFT	CellIFT	Org	SHIFT	CellIFT	Org	SHIFT	CellIFT
AES benchmarks from Trust-Hub										
AES_T100	358301	447752	1.03	1.23	241136.7	525747.2	814406.8	3.74	5.74	6.44
AES_T200	358322	447762	1.06	1.23	241137.1	525890.3	820118.9	3.74	5.74	6.44
AES_T400	358410	447850	1.08	1.20	242253.3	528144.8	822800.8	3.74	5.77	6.43
AES_T700	358362	447802	1.08	1.21	242261.9	525932.1	820166.8	3.74	5.74	6.44
AES_T900	358384	447824	1.09	1.23	241357.4	529208.0	823864.1	3.74	5.78	6.43
AES_T1200	358370	447810	1.08	1.22	241874.3	529316.9	823972.9	3.74	5.78	6.43
AES_T1600	358461	447901	1.10	1.24	242253.9	527606.9	822263.0	3.74	5.76	6.44
Average	19.9% ↓		12.1% ↓		281.1% ↑	-	35.7% ↓	153.9% ↑	-	10.5% ↓
Other cryptographic core benchmarks										
RSA_T100	1108	1114	0.30	0.28	6305.8	16850.0	16855.2	1.94	3.25	3.25
RSA_T200	1115	1127	0.29	0.28	6191.5	17105.2	17119.8	1.94	3.25	3.25
RSA_T300	1178	1182	0.32	0.30	6676.8	17275.7	17280.2	1.94	3.25	3.25
RSA_T400	1185	1198	0.32	0.31	6629.8	17321.4	17330.3	1.94	3.25	3.26
Average	0.7% ↓		5.2% ↑		265.8% ↑	-	0.5% ↓	167.5% ↑	-	0.1% ↓
AES_core	31250	51821	0.91	1.02	12551.6	27598.6	47294.3	0.72	1.17	1.39
DES_core	3376	5386	0.49	0.62	1971	2401	4194.2	0.33	0.41	0.51
Benchmakrs for other cores										
SHA512	7214	7798	2.14	2.15	39530.8	115750.2	117692.5	3.72	4.34	4.45
SPI	1978	2073	0.35	0.32	1161.1	2346.7	3041.3	0.39	0.43	0.48
OpenMSP 430	5022	5873	0.77	0.81	8194.4	19879.9	22793.7	0.72	1.31	1.33
Trng	10999	12204	3.39	3.42	83902.2	218302.8	233499.2	0.28	0.57	0.61
Uart	9137	9214	1.05	1.06	7893.2	18549.3	18721.3	0.12	0.17	0.17

Fig. 9. Detected lightness of SHIFT under multiple test benchmarks.

SHIFT uses the determined constraint information to optimize the IFT logic during the synthesis process, ensuring that only redundant tracking logic is eliminated without introducing false alarm and omission problems. As shown in Figure 8, we perform Trojan detection on 10 representative benchmark programs and observe the output results using different test cases, and the experimental results show that SHIFT maintains the same detection precision as CellIFT.

Conservative Optimizations for SHIFT: During the static analysis phase, SHIFT first divides the coarse-grained functions (modules) into regions based on the taint signals. It only cancels the tracking logic for a region when no taint signals are detected. Simultaneously, SHIFT simplifies the tracking logic using specific constants. This process does not introduce any uncertainty and preserves the original tracking precision. In summary, SHIFT implements optimization labels through conservative static processing, ensuring that it does not create false alarm issues due to optimization efforts under any conditions.

5.2 Lightweight of SHIFT

SHIFT is used as an optimization technique for hardware information flow tracing, and its lightweight nature is verified by comparing it with the information flow models generated by CellIFT on multiple metrics. The experimental environment is Yosys and Design Compiler, the synthesis process uses 40 nm cell library and measures the number of synthesized cells, information flow model generation time, synthesized area, and verification time of various mainstream hardware cryptographic designs and interface communication designs.

SHIFT optimizes the deployed tracking logic by statically analyzing the collected deterministic constraints, and the number of redundant logics reduced by SHIFT is visualized in the Cell Number and Cell Area metrics. The Systhesis Time metrics reflect the generation time of the information flow model, which includes the static analysis time and the time to interpolate the tracking logic

in SHIFT. The Systhesis Time metric reflects the generation time of the information flow model, including the static analysis time of SHIFT and the time of inserting staking tracking logic. The Verification Time metric reflects the verification time of the security attribute of the information flow model.

As shown in Figure 9, on the AES test set from Trust-Hub, compared with CellIFT, SHIFT reduces the deployment time of trace logic by 12.1%, the generated information flow model reduces the number of cells by 19.9% and the synthesis area by 35.7%, and reduces the security analysis time of the information flow model by 10.5%. The experimental results show that SHIFT can effectively alleviate the problem of excessive overhead in deploying IFT techniques on large and complex hardware designs.

6 Summarize

This article presents a constraint-driven optimization technique for SHIFT. In terms of the capability to trace information flow, the trace logic based on macrocells offers superior precision and performance compared to other abstraction layers. In addition, to reduce the instrumentation overhead, SHIFT adapts optimization techniques from the field of software information flow tracing to mitigate redundant logic in hardware information flow tracing models. Specifically, coarse-grained static analysis can efficiently extract constraints and selectively deploy trace logic when generating trace models. In comparison to previous optimization techniques, SHIFT demonstrates advancements in both optimization efficiency and scalability. Experimental results across various hardware designs indicate that SHIFT can reduce the overhead associated with deploying tracking logic without increasing false positives, thereby achieving an accurate and lightweight HIFT technology.

However, this work necessitates that security personnel provide the initial taint value, and the coarse-grained static analysis may still introduce some redundancy in the IFT model. To address the former issue, we propose the automated deployment of taint initial values by leveraging large language models in the future. For the latter issue, we can consider incorporating additional logic into the instrumentation process to capture dynamic runtime constraint information, which will result in a more efficient IFT model.

References

[1] OpenCores. 2025. Open Source IP-Cores. Retrieved from https://opencores.org/projects?expanded=Crypto%20core

[2] Trust-Hub. 2025. Trust-Hub.org. Retrieved from https://trust-hub.org/#/benchmarks/chip-level-trojan

[3] Armaiti Ardeshiricham, Wei Hu, and Ryan Kastner. 2017. Clepsydra: Modeling timing flows in hardware designs. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 147–154.

[4] Armaiti Ardeshiricham, Wei Hu, Joshua Marxen, and Ryan Kastner. 2017. Register transfer level information flow tracking for provably secure hardware design. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1691–1696.

[5] Armaiti Ardeshiricham, Yoshiki Takashima, Sicun Gao, and Ryan Kastner. 2019. VeriSketch: Synthesizing secure hardware designs with timing-sensitive information flow properties. In *2019 ACM SIGSAC Conference on Computer and Communications Security*, 1623–1638.

[6] Sanchuan Chen, Zhiqiang Lin, and Yinqian Zhang. 2021. {SelectiveTaint}: Efficient data flow tracking with static binary rewriting. In *30th USENIX Security Symposium (USENIX Security '21)*, 1665–1682.

[7] Yongliang Chen, Xiaole Cui, Xiaoxin Cui, and Xing Zhang. 2024. The area-efficient gate level information flow tracking schemes of digital circuit with multi-level security lattice. *Microelectronics Journal* 144 (2024), 106088.

[8] Shuwen Deng, Doğuhan Gümüşoğlu, Wenjie Xiong, Y. Serhan Gener, Onur Demir, and Jakub Szefer. 2017. Secchisel: Language and tool for practical and scalable security verification of security-aware hardware architectures. Cryptology ePrint Archive, Report 2017/193.

[9] Dorothy E. Denning. 1976. A lattice model of secure information flow. *Communications of the ACM* 19, 5 (May 1976), 236–243. DOI: https://doi.org/10.1145/360051.360056

[10] Mehran Goli and Rolf Drechsler. 2021. ATLaS: Automatic detection of timing-based information leakage flows for SystemC HLS designs. In *26th Asia and South Pacific Design Automation Conference*, 67–72.

[11] Wei Hu, Armaiti Ardeshiricham, and Ryan Kastner. 2021. Hardware information flow tracking. *ACM Computing Surveys* 54, 4, Article 83 (May 2021), 39 pages. DOI: https://doi.org/10.1145/3447867

[12] Wei Hu, Andrew Becker, Armita Ardeshiricham, Yu Tai, Paolo Ienne, Dejun Mu, and Ryan Kastner. 2016. Imprecise security: Quality and complexity tradeoffs for hardware information flow tracking. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[13] Wei Hu, Baolei Mao, Jason Oberg, and Ryan Kastner. 2016. Detecting hardware trojans with gate-level information-flow tracking. *Computer* 49, 8 (2016), 44–52.

[14] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. 2020. Spectre attacks: Exploiting speculative execution. *Communications of the ACM* 63, 7 (Jun. 2020), 93–101. DOI: https://doi.org/10.1145/3399742

[15] Xun Li, Vineeth Kashyap, Jason K. Oberg, Mohit Tiwari, Vasanth Ram Rajarathinam, Ryan Kastner, Timothy Sherwood, Ben Hardekopf, and Frederic T. Chong. 2014. Sapper: A language for hardware-level security policy enforcement. In *19th International Conference on Architectural Support for Programming Languages and Operating Systems*, 97–112.

[16] Xun Li, Mohit Tiwari, Jason K. Oberg, Vineeth Kashyap, Frederic T. Chong, Timothy Sherwood, and Ben Hardekopf. 2011. Caisson: A hardware description language for secure information flow. *ACM SIGPLAN Notices* 46, 6 (2011), 109–120.

[17] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, et al. 2020. Meltdown: Reading kernel memory from user space. *Communications of the ACM* 63, 6 (May 2020), 46–56. DOI: https://doi.org/10.1145/3357033

[18] Geraldine Shirley Nicholas, Dhruvakumar Vikas Aklekar, Bhavin Thakar, and Fareena Saqib. 2023. Secure instruction and data-level information flow tracking model for risc-v. *Cryptography* 7, 4 (2023), 58.

[19] Jason Oberg, Wei Hu, Ali Irturk, Mohit Tiwari, Timothy Sherwood, and Ryan Kastner. 2010. Theoretical analysis of gate level information flow tracking. In *47th Design Automation Conference*, 244–247.

[20] Jason Oberg, Sarah Meiklejohn, Timothy Sherwood, and Ryan Kastner. 2014. Leveraging gate-level properties to identify hardware timing channels. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33, 9 (2014), 1288–1301.

[21] Maoyuan Qin, Xinmu Wang, Baolei Mao, Dejun Mu, and Wei Hu. 2020. A formal model for proving hardware timing properties and identifying timing channels. *Integration* 72 (2020), 123–133.

[22] Fabian Schuiki, Andreas Kurth, Tobias Grosser, and Luca Benini. 2020. LLHD: A multi-level intermediate representation for hardware description languages. In *41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 258–271.

[23] Flavien Solt, Ben Gras, and Kaveh Razavi. 2022. {CellIFT}: Leveraging cells for scalable and precise dynamic information flow tracking in. {RTL}. In *31st USENIX Security Symposium (USENIX Security '22)*, 2549–2566.

[24] Yu Tai, Wei Hu, Dejun Mu, Baolei Mao, Lantian Guo, and Maoyuan Qin. 2018. A simplifying logic approach for gate level information flow tracking. In *12th International Conference on Communications and Networking*. Springer, 302–311.

[25] Yu Tai, Wei Hu, Hui-Xiang Zhang, De-Jun Mu, and Xing-Li Huang. 2016. Generating optimized gate level information flow tracking logic for enforcing multilevel security. *Automatic Control and Computer Sciences* 50 (2016), 361–368.

[26] Mohit Tiwari, Hassan, M. G. Wassel, Bita Mazloom, Shashidhar Mysore, Frederic T. Chong, and Timothy Sherwood. 2009. Complete information flow tracking from the gates up. In *14th International Conference on Architectural Support for Programming Languages and Operating Systems*, 109–120.

[27] Ofir Weisse, Jo Van Bulck, Marina Minkin, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Raoul Strackx, Thomas, F. Wenisch, and Yuval Yarom. 2018. *Foreshadow-NG: Breaking the Virtual Memory Abstraction with Transient out-of-Order Execution.* Technical report.

[28] Danfeng Zhang, Yao Wang, G. Edward Suh, and Andrew C. Myers. 2015. A hardware design language for timing-sensitive information-flow security. *ACM SIGPLAN Notices* 50, 4 (2015), 503–516.

[29] Yiyu Zhang, Tianyi Liu, Yueyang Wang, Yun Qi, Kai Ji, Jian Tang, Xiaoliang Wang, Xuandong Li, and Zhiqiang Zuo. 2024. HardTaint: Production-run dynamic taint analysis via selective hardware tracing. *Proceedings of the ACM on Programming Languages* 8, OOPSLA2 (2024), 1615–1640.

Received 18 January 2025; revised 5 June 2025; accepted 22 August 2025

Analog and Temporary On-chip Memory for ANN Training and Inference

SHREYAS DESHMUKH and RAGHAV SINGHAL, Indian Institute of Technology Bombay, Mumbai, India
SHRUTI LANDGE, Indian Institute of Technology Indore, Indore, India
VIVEK SARASWAT, ANMOL BISWAS, ABHISHEK KADAM, and AJAY K. SINGH, Indian Institute of Technology Bombay, Mumbai, India
SREENIVAS SUBRAMONEY, Intel Labs, Bengaluru, India
LAXMEESHA SOMAPPA, MARYAM SHOJAEI BAGHINI, and UDAYAN GANGULY, Indian Institute of Technology Bombay, Mumbai, India

On-chip training at the edge becomes a primary requisite for real-time and security-sensitive artificial neural network (ANN) applications. In-memory computation (IMC) techniques have been proposed to facilitate data-intensive computational operations in ANNs. IMC-based multiply-accumulate (MAC) accelerates ANN training but suffers from significant communication overhead between the MAC engine and the off-chip storage for the intermediate data. This article proposes an analog temporary on-chip memory (ATOM) to store this intermediate data during ANN training. The ANN training architecture with the proposed ATOM has two significant advantages. First, the energy required to store intermediate data is scaled down by ~40× due to the on-chip and analog nature of the memory. Second, the proposed architecture avoids power and area-consuming analog-to-digital converters (ADCs) between neural network stages. The ATOM cell measurements are carried out from 20 fabricated chips, and the impact of ATOM characteristics on ANN system performance accuracy is analyzed. This article shows significant latency improvement of ~9× and area savings of ~5× for intermediate data storage compared to the on-chip SRAM during ANN training's forward and backward pass operations. An improvement in the area and latency will be beneficial to instrument the area- and energy-efficient hardware system for on-chip ANN applications.

The Semiconductor Research Corporation (SRC), DST Nano Mission, MeitY, and the Government of India support this work in parts through the NNetRA project. GlobalFoundries supported silicon fabrication through the university program. Fraunhofer IIS, Erlangen supported the design submission. The Prime Minister's Research Fellowship supported Vivek Saraswat's work.

Authors' Contact Information: Shreyas Deshmukh (corresponding author), Indian Institute of Technology Bombay, Mumbai, India; e-mail: shreyasdesh@iitb.ac.in; Raghav Singhal, Indian Institute of Technology Bombay, Mumbai, India; e-mail: 10raghavsinghal@gmail.com; Shruti Landge, Indian Institute of Technology Indore, Indore, India; e-mail: shruti.parag.landge@gmail.com; Vivek Saraswat, Indian Institute of Technology Bombay, Mumbai, India; e-mail: vsaraswat009@gmail.com; Anmol Biswas, Indian Institute of Technology Bombay, Mumbai, India; e-mail: 194076019@iitb.ac.in; Abhishek Kadam, Indian Institute of Technology Bombay, Mumbai, India; e-mail: aakadam@iitb.ac.in; Ajay K. Singh, Indian Institute of Technology Bombay, Mumbai, India; e-mail: ajayksinghel@gmail.com; Sreenivas Subramoney, Intel Labs, Bengaluru, India; e-mail: sree.subramoney@gmail.com; Laxmeesha Somappa, Indian Institute of Technology Bombay, Mumbai, India; e-mail: laxmeesha@ee.iitb.ac.in; Maryam Shojaei Baghini, Indian Institute of Technology Bombay, Mumbai, India; e-mail: mshojaei@iitb.ac.in; Udayan Ganguly, Indian Institute of Technology Bombay, Mumbai, India; e-mail: udayan.ganguly@iitb.ac.in.

ACM 1550-4840/2025/10-ART14
https://doi.org/10.1145/3765899

CCS Concepts: • **Hardware** → **Dynamic memory**; **Memory and dense storage**; • **Computer systems organization** → *Distributed architectures*;

Additional Key Words and Phrases: Analog memory, Artificial Neural Network (ANN), In-Memory Computation (IMC), Matrix-Vector Multiplication (MVM), On-chip training

ACM Reference format:
Shreyas Deshmukh, Raghav Singhal, Shruti Landge, Vivek Saraswat, Anmol Biswas, Abhishek Kadam, Ajay K. Singh, Sreenivas Subramoney, Laxmeesha Somappa, Maryam Shojaei Baghini, and Udayan Ganguly. 2025. Analog and Temporary On-chip Memory for ANN Training and Inference. *ACM J. Emerg. Technol. Comput. Syst.* 21, 4, Article 14 (October 2025), 18 pages.
https://doi.org/10.1145/3765899

1 Introduction

Artificial neural networks (ANNs) have become a prominent candidate for **machine learning (ML)** applications like image classification [47], face recognition [2], and speech processing [30]. For these applications, hardware systems at the edge of the cloud or in autonomous (cloud-free) environments prefer the on-chip/*in-situ* ANN training [24, 26]. On-chip training at the edge has certain advantages: first, latency, energy, and data privacy improvements due to the avoidance of cloud communication or when the cloud is unavailable [24]. Second, on-chip training is variability-aware and suitable for dynamic environments because it can adapt to the operational setup (e.g., voltage supply variation), environment (e.g., temperature, noise, etc.), and manufacturing non-idealities (e.g., transistor mismatch, variability) [11, 24].

In-memory computation (IMC) is a popular choice to accelerate ANN training and inference as it has reduced area, power, and latency requirements compared to conventional Von-Neuman architecture [10, 21, 29, 37, 44]. The analog IMC with a bi-directional memory array has been demonstrated to accelerate the ANN training using a back-propagation algorithm [4, 17, 41]. Some analog IMC-based **multiply-accumulate (MAC)** operation unit accepts digital inputs and generate analog outputs. It requires **analog-to-digital converters (ADCs)** with a high area and power penalty to convert the MAC output to digital form for the following ANN layers [15, 20, 28]. Therefore, the challenge is to design ANN hardware with reduced ADC requirements.

In addition, extensive intermediate data is generated during the ANN training's **forward pass (FP)** and **backward pass (BP)** operations. This intermediate data, i.e., neuronal activation, error gradient, and weight gradient, is temporarily required during training and conventionally stored in off-chip digital memory [6, 17, 34]. However, off-chip communication cost (i.e., area, power, and latency) is remarkably high [12, 35]. Moreover, ADCs are required before off-chip communication of intermediate data because IMC outputs, which partially represent intermediate data, are analog. Therefore, analog on-chip intermediate data storage is another challenge.

In this article, we offer a solution to these challenges of reduced ADC requirements and on-chip intermediate data storage for ANN on-chip training hardware. We propose an ***a*nalog *t*emporary *o*n-chip *m*emory (ATOM)** to store intermediate data in the analog domain and surmount conventional on-chip training challenges with off-chip intermediate data storage. The proposed ATOM uses a field-effect transistor gate capacitor to store the intermediate data in an analog domain. We have presented the ATOM unit cell characteristics from 20 fabricated chips. In addition, we have analyzed the impact of analog memory storage on ANN system performance.

Table 1 qualitatively presents the status of the available and proposed intermediate data storage cells for ANN training and inference. Off-chip DRAM [17] and on-chip SRAM [34] have been commonly used to store intermediate data; however, ADCs are required to convert the analog

Table 1. Status of the Available Intermediate Data Storage Cells for ANN Training and Inference

	[17]	[34]	[31]	[4]	[15]	**Proposed**
Application	Training	Training	Inference	Training	Inference	Training
On-chip Memory	No	Yes	Yes	Yes	Yes	Yes
Analog Memory	No	No	No	Yes	Yes	Yes
Experimental Demonstration	No	Yes	No	No	Yes	Yes
Retention Analysis	NA	NA	Yes	No	No	Yes
Variability Analysis	NA	NA	NA	No	No	Yes

MAC outputs before storing them in digital memories. Recently, an innovative mixed CMOS cell memory design has been introduced to optimize performance, area, and energy efficiency for intermediate data storage by combining SRAM and eDRAM cells [31]. However, ADCs remain necessary during AI operations to convert intermediate data into a digital format for storage in the on-chip digital memory. Moreover, on-chip analog memories have been explored for training [4] and inference [15], but challenges such as memory decay, variability, and their impact on system performance need further consideration. Despite the benefits of analog memory and its processing, such as power and area efficiency and quantization-free processing [15, 16], non-idealities like variability and capacitor discharge-induced decay can hinder performance, unlike in digital domain implementations. Therefore, in this article, to address the challenge of area- and energy-hungry ADC usage during ANN training and inference, we propose ATOM for intermediate data storage and evaluate the robustness of on-chip training network performance against the effects of decay and variability in ATOM.

Since ATOM stores intermediate data in the analog domain, it is particularly advantageous when paired with the analog IMC unit, which accepts the activation inputs in the analog domain. The IMC architectures are categorized based on whether the accumulation operation occurs in the analog or digital domain as analog and digital IMC [18, 19, 29, 44]. The literature proposes and demonstrates the charge-based [8, 14, 25, 46] and current-based [13, 17, 41] analog IMC units. Specific analog IMC units (both current- and charge-based) accept analog input [13, 15, 25], while others require inputs in the digital form [14, 17, 46]. However, the output from all analog IMC units will always be in analog form. Consequently, for analog IMC, when inputs are accepted in the analog domain, the need for ADCs can be avoided by storing the MAC output in addition to other intermediate data (i.e., error gradient and weight gradient) in the analog domain using ATOM. However, for analog IMC architectures that accept digital input, ADCs are essential because the output from the previous layer (or previous IMC unit) will be in analog form. Therefore, the proposed ATOM can eliminate the need for ADCs during intermediate data storage for ANN architecture with an analog IMC unit (both current- and charge-based), which accepts the input in analog form.

In [40], we have presented another application of analog memory, i.e., a memory element, to store the weights in the IMC unit. While [40] highlights the importance of analog memory as a synapse (weight) within the context of IMC operations in ANN, this article demonstrates the role of analog memory as a neuron component, specifically for storing intermediate data generated during ANN inference and training. In [40], we showed that analog volatile memory for synapses enhances inference performance accuracy, as the natural decay of this memory acts as a form of regularization, helping to mitigate the challenge of overfitting during ANN training. On the other hand, analog memory demonstrated in this article for intermediate data storage addresses the challenges of off-chip data communication and the costly analog-to-digital conversion inherent in traditional ANN training architectures [17].

The ANN training with analog IMC using conventional memories like SRAM [29] or emerging memories like RRAM [1, 32] for weight storage can use the proposed ATOM for intermediate data storage. As this article presents the analog memories for intermediate data storage, the other applications of analog memories during ANN inference and training, i.e., weight storage [7, 40], bit-multiplication output storage (before accumulation) [45, 46] in the memory array during IMC are not in the scope of this article.

The main contributions of this article are:

—Design and experimental characterization of the ATOM prototype (from 20 different chips) to store intermediate data during ANN training. This analog on-chip storage results in minimized off-chip data communication for intermediate data storage.
—Evaluation of system-level performance with measured ATOM non-ideal characteristics, including memory retention constant and variability.

The rest of the article is organized as follows: Section 2 discusses the basic operations of ANN training and inference. Section 3 introduces the ANN training architecture along with the proposed ATOM schematic. Section 4 presents the proposed ATOM characterization. Section 5 analyzes the system performance with the ATOM characteristics from the previous section and compares the ANN architecture using ATOM with other conventional memory storage. Section 6 provides the conclusion.

2 ANN Training and Inference

Inspired by the biological neural system, the ANN, popularly used in ML applications, can be considered a network of neurons. The ANN training using a back-propagation algorithm can be regarded as a 4-step operation, namely (i) FP, (ii) BP for error calculation, (iii) calculation of gradient for the weight (CG), and (iv) weight calculation and update (CW). During the inference operation that follows training, the trained weights from the training steps are used to generate the inference result.

In the first step, i.e., FP, the **matrix-vector multiplication (MVM)** operation between the weight matrix (W_n) and the input vector (X_n) of a layer n is performed to generate the MAC output (Z_n). The activation function (f) is applied on the MAC output to produce the input for the next ANN layer (X_{n+1}), as shown in Equation (1):

$$X_{n+1} = f(Z_n) = f(X_n.W_n + b_n). \tag{1}$$

In Equation (1), b_n represents the biasing elements of layer n. The second step, which is a BP, calculates the error vector (δ_n) for layer n by performing the MVM operation between the transposed weight matrix (W_n^T) of layer n and the error vector (δ_{n+1}) of layer $n+1$. The derivative of the $f(Z_n)$ with respect to Z_n, i.e., $f'(Z_n)$, is also used in error calculation, as shown in Equation (2):

$$\delta_n = \delta_{n+1}.f'(Z_n).W_n^T. \tag{2}$$

The outer product ($*$) between the error vector from the BP and the activation from the FP generates the weight gradient matrix (ΔW_n) during the weight gradient calculation step, as shown in Equation (3). Finally, the weight matrix gets updated using the weight gradient matrix in the weight update step, as shown in Equation (4):

$$\Delta W_n = (\delta_{n+1}.f'(Z_n)) * X_n, \tag{3}$$

$$W_n(new) = W_n(old) - \Delta W_n. \tag{4}$$

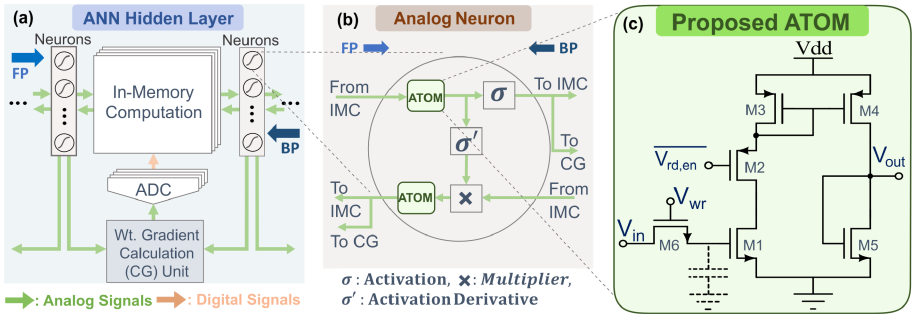


Fig. 1. ANN hidden layer and neuron with proposed ATOM: (a) ANN hidden/intermediate layer with neurons for on-chip training. (b) Proposed analog neuron structure with ATOM. (c) Proposed ATOM schematic.

This set of operations is performed for each data sample. The evaluation of data points from the learning data set once in the network is known as an epoch. The weight update can happen for every input or set of inputs, depending on the batch size.

With this basic understanding of ANN training operation, the following section discusses ANN training in an analog environment and proposes ATOM schematics to facilitate the ANN operation in the analog domain.

3 ANN Training Architecture and Analog Memory

A hidden or intermediate layer of the analog ANN training architecture is shown in Figure 1(a). In this architecture, the three stages of the training operation, i.e., FP, BP, and weight gradient calculation, are performed in the analog domain. Therefore, the ADCs are only needed during the weight update, i.e., before writing back the updated weights to the IMC array, resulting in minimal ADC usage compared to the ANN architecture discussed in [17, 41].

The weight matrix required to perform MAC operation using IMC during FP and BP is stored in the memory array. The exact weight matrix can be used in standard topology in FP and transposed topology in BP, which minimizes the hardware requirement for ANN training [17, 41]. As shown in Figure 1(a), the inputs required for the intermediate layers are in the analog domain; therefore, the ADCs are not needed during FP and BP.

3.1 On-chip Training with Analog Neuron

The neuron, shown in Figure 1(b), can be operated in the analog domain, including activation (σ) during FP, a derivative of activation (σ'), and analog multiplication ($\times$) between the activation derivative and error during BP. The activation and activation derivative circuits with the analog domain operations are discussed in [3, 5], and an analog multiplier can be used for multiplication operations in neurons. The neuron with a proposed analog memory module to store the intermediate data in the analog domain avoids the analog-to-digital conversion requirement during FP and BP. The CG unit, shown in Figure 1(a), can also be operated in the analog environment to perform the outer product between the error and activation vectors to generate the weight gradient matrix, further reducing ADC usage. This way, ADC usage's power and area penalty can be minimized for ANN training.

3.2 Proposed ATOM

The intermediate analog data generated in FP and BP are needed during later stages of training, i.e., for weight gradient calculation and weight update. However, after the training cycle for the

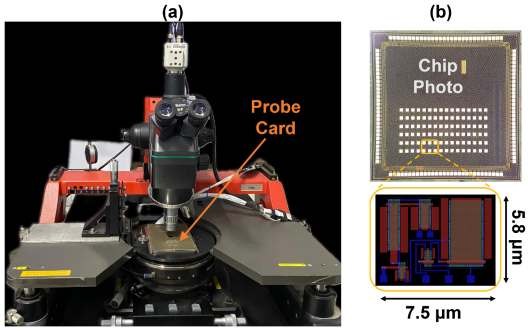


Fig. 2. Measurement setup using probe card and fabricated chip: (a) Fabricated chip with attached probe card setup for measurement. (b) Photograph of the fabricated chip with the ATOM layout.

given data samples is completed, the intermediate data associated with the given data samples is not required. Therefore, the analog memory, which can temporarily hold the data, i.e., for a few tens of clock cycles, can be used to store this intermediate data. We propose an ATOM for intermediate data storage. As ATOM is on-chip memory, the off-chip communication cost, i.e., area, energy, and latency, associated with intermediate data storage, gets eliminated to a considerable extent by storing a significant portion of intermediate data on-chip using ATOM. Moreover, the distributed ATOMs are available near the IMC unit, reducing the latency and energy needed for communication. The proposed analog temporary memory, shown in Figure 1(c), uses the gate capacitance of transistor M1 to hold or retain the input analog voltage (V_{in}) for the required time duration during ANN training. The detailed ATOM operation and prototype measurements from 20 fabricated chips are discussed in the following section.

4 ATOM Characterization

The ATOM prototype measurements are carried out using the 2×16 probe card, shown in Figure 2(a), with Keysight B1500A Semiconductor Device Analyzer. The ATOM circuit prototype is fabricated using the GLOBALFOUNDRIES (GF) 45 nm RF-SOI foundry technology. Figure 2(b) shows the photograph of the fabricated chip with the ATOM layout.

As shown in Figure 2(b), the ATOM layout requires an area of ~44 μm^2, which is ~5× higher than the equivalent conventional digital memory bit-cells (e.g., SRAM). However, it is much smaller than the equivalent ADC and memory bit-cells area by ~160×, which ATOM replaces. Moreover, as retention time depends on the bit-cell area, the ATOM area can be further reduced if the data needs to be retained for a shorter duration (as per requirement). A detailed analysis of retention time dependence on the bit-cell area is discussed in the following subsection.

Figure 3(a) shows the ATOM circuit schematic with input and output terminals for read/write operation [40]. During the write operation, the write pulse (V_{wr}) is applied to the gate terminal of the write transistor (M6) to store the input (V_{in}) at the M1 gate capacitor (V_{cap}). After the write operation is completed, the V_{in} and V_{wr} signals can be removed, and the M1 gate capacitance retains the input for a certain duration. The M1 gate capacitance is used to hold or retain the input, which will be read after a few operation cycles, i.e., a few tens of nanoseconds, during ANN training. The read circuit, which consists of the switch (M2), current mirror (M3–M4), and the load (M5), generates the output (V_{out}) almost equal (or proportional) to the V_{cap}. The switch (M2) needs to be turned ON during read operation, which is OFF otherwise, resulting in power saving. As V_{in} is stored at the gate terminal of M1 with the help of M6, the M1 gate capacitor, and M6 off-state leakage are the primary factors controlling the retention time of the ATOM cell. Therefore, M1

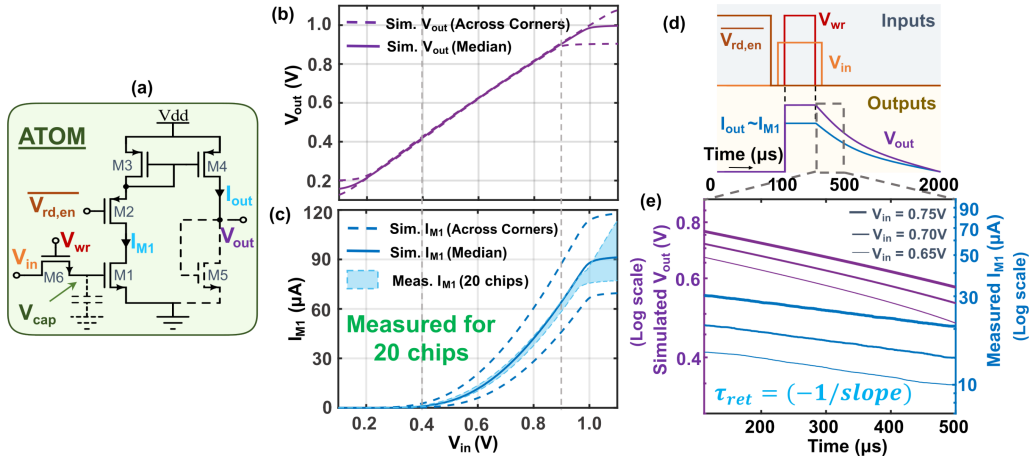


Fig. 3. Fabricated ATOM cell measurements and simulations: (a) ATOM schematic. (b) Simulated V_{out} vs. V_{in} DC characteristics (across process corners). (c) Measured (across 20 chips) and simulated (across process corners) I_{M1} vs. V_{in} DC characteristics. (d) Representative inputs and expected output behavior for ATOM cell measurements. (e) Decay measurement (slope) for I_{M1} and V_{out} with time for various V_{in} to calculate retention time constant (τ_{ret}).

and M6 dimensions need to be carefully chosen depending on the retention time requirement of the given system. With the available probe-card setup, we have measured the current through M1 (I_{M1}), which is equal to the I_{out} and proportional to the V_{out}.

Figure 3(b) and (c) presents ATOM's measured and simulated DC characteristics to determine the operating region of ATOM. Figure 3(b) shows simulated V_{out} vs. V_{in} DC characteristics across process corners. Figure 3(c) presents the I_{M1} vs. V_{in} DC characteristics measured across 20 chips (2 measurements each), shown by the shaded region and simulated across the process corners. It demonstrates the range of V_{in} for which I_{M1} follows the V_{in}, i.e., it implies the operating range of V_{in}. From Figure 3(c), it is observed that the simulated I_{M1} median (across process corners) follows the measured I_{M1}. It shows that the measured and simulated I_{M1} are well-matched with each other for the given V_{in} range (0.4 V to 1.0 V). Whereas it can be observed from Figure 3(b) that V_{out} follows the V_{in} from 0.2 V to 0.9 V. Therefore, from Figure 3(b) and (c), it can be concluded that for the range 0.4 V to 0.9 V, V_{out} and I_{M1} (i.e., outputs) follows V_{in} (i.e., input). Therefore, input should be kept between 0.4 V and 0.9 V for faithful reproduction of the input at the output terminal.

Figure 3(d) shows the ATOM operation's representative input and output signal behavior. The V_{in} should be stable while applying V_{wr} pulse to write or store the input V_{in} at V_{cap} terminal. After completing the write operation, the active low $\overline{V_{rd,en}}$ should be kept at logic zero during a read operation. For demonstrating continuous read operation of outputs, i.e., V_{out} and I_{M1}, the $\overline{V_{rd,en}}$ set to 0 V, as shown in Figure 3(d). After applying the V_{wr} pulse, the I_{M1} and V_{out} will take some time to settle to a steady state. And when V_{wr} and V_{in} signal gets removed, the V_{cap} starts decaying due to leakage from M6, i.e., V_{cap} terminal retains the charges for a certain duration after input signals get removed. This retention effect or decay is reflected in I_{M1} and V_{out}. With the available measurement setup limitation, we have measured I_{M1} and simulated V_{out} for various values of V_{in}, as shown in Figure 3(e). Figure 3(e) shows the decay with time in I_{M1} (measured) and V_{out} (simulated) for various V_{in}, after V_{wr} signal gets removed. This decay slope is used to calculate the retention time constant (τ_{ret}) of an ATOM unit cell ($\tau_{ret} = -1/decay\_slope$).

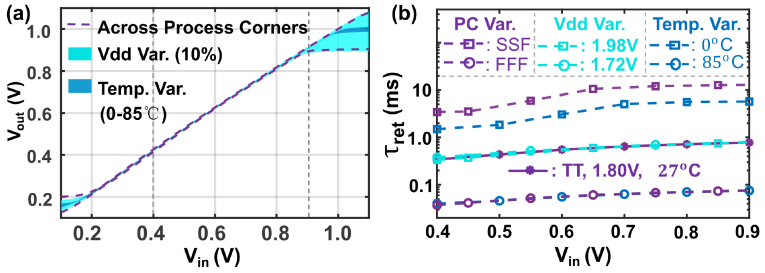


Fig. 4. Simulated ATOM characteristics with PVT variation: (a) Simulated V_{out} vs. V_{in} DC characteristics across PVT variation. (b) Simulated retention time constant, τ_{ret}, of V_{out} with PVT variation for various values of V_{in}. (Here, to observe the impact of individual element (P, V, or T) variation, the particular element is varied with the other two kept at a standard fixed level, i.e., during process corner (PC) variation, temperature is kept constant at 27°C with V_{dd} = 1.8 V. Whereas, during 10% V_{dd} variation, the TT corner is considered with a temperature of 27°C. Similarly, during temperature variation, temperature is varied from 0°C to 85°C with the TT corner and V_{dd} = 1.8 V.)

The impact of **process, voltage, and temperature (PVT)** and ATOM cell area variation on the retention time constant is discussed in the following subsection. The following section, Section 5, analyzes and presents whether this ATOM retention time is sufficient to retain the intermediate data during ANN training.

4.1 PVT and Area Impact on ATOM Characteristics

In ATOM, the charge or input is stored at the gate terminal of a transistor M1, i.e., at MOSCap, as shown in Figure 3(a), and MOSCap is sensitive to PVT as well as unit cell area variation. Therefore, this subsection analyzes and discusses the impact of PVT and unit cell area variation on ATOM characteristics.

Figure 4(a) and (b) shows the impact of PVT variation, i.e., process corners, 10% V_{dd}, and 0℃ to 85℃ temperature variation, on ATOM characteristics. Figure 4(a) presents the simulated V_{out} vs. V_{in} DC characteristics with PVT variation and shows that V_{out} follows the V_{in} for the input voltage between 0.2 V and 0.9 V. The fitted linear line has a maximum root mean square error of 3.4 mV across PVT. Therefore, it can be concluded that the ATOM DC characteristics are robust to PVT variation for the given operating range of 0.2 V to 0.9 V. Figure 4(b) presents the simulated V_{out} retention time constant (τ_{ret}) for various V_{in}. It shows the impact of PVT variation on τ_{ret} for the given operating range of 0.4 V to 0.9 V. There is a negligible impact of 10% V_{dd} variation on τ_{ret}. However, with increase in temperature (from 0°C to 85°C), there is ~1.7 order decrease in τ_{ret}. Similarly, process corner variation shows ~2 order change in τ_{ret}. From Figure 4(b), we can notice the worst-case τ_{ret} as ~0.05 ms. Now, whether the worst-case retention time affects the given system performance and the impact of variation in retention time on system performance are analyzed in the following section.

Figure 5 shows the impact of the ATOM cell area on the simulated V_{out} retention time constant. The solid line in Figure 5 shows the simulated retention time of the ATOM with fabricated unit cell dimensions. As expected, the retention time decreases with the decrease in area. It can be observed from Figure 5 that if the retention time requirement gets reduced by ~5×, we can reduce the ATOM cell area by ~2×. Therefore, the ATOM cell dimensions should be selected per the system's retention time requirement to optimize the overall system area.

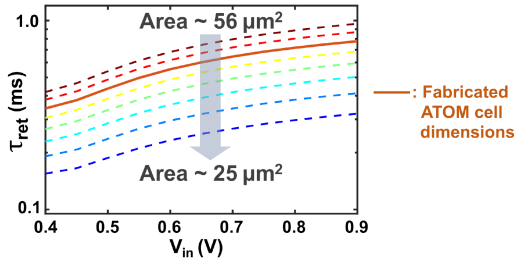


Fig. 5. Simulated ATOM retention time with area variation: Retention time constant, τ_{ret}, of V_{out} with ATOM unit cell area variation for various values of V_{in}.

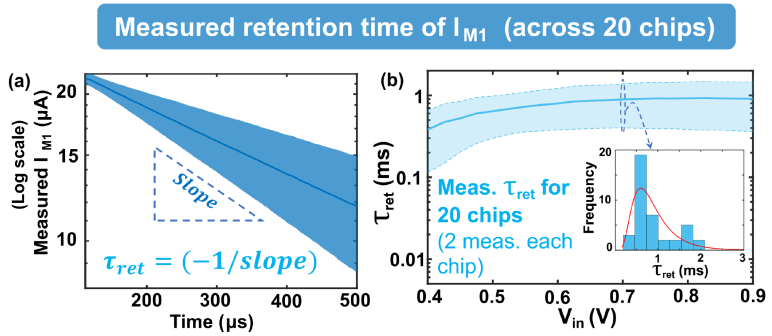


Fig. 6. Measured retention time of I_{M1}: (a) Experimental retention transient mean of I_{M1} with $\pm 1\sigma$ variation (shaded) for $V_{in} = 0.7$ V. (b) Measured retention time constant (τ_{ret}) mean of I_{M1} along with $\pm 1\sigma$ variation (shaded) for various values of V_{in} (inset) histogram of τ_{ret} for $V_{in} = 0.7$ V.

Table 2. Measured and Simulated ATOM Retention Time Constant (τ_{ret}) for I_{M1}

	Simulated	Measured
Mean Retention Time Constant at $V_{in} = 0.7$ V	0.44 ms	0.66 ms

4.2 ATOM Retention Time Measurements

The measured retention time of I_{M1} for 20 fabricated chips (2 measurements each) is shown in Figure 6. The retention time transient at $V_{in} = 0.7$ V with $\pm 1\sigma$ variation represented by the shaded region is shown in Figure 6(a). The I_{M1} decays over time, and the inverse of the decay slope, shown in Figure 6(a), represents the retention time constant (τ_{ret}). This τ_{ret} is used to evaluate the performance of the ANN system with ATOM.

The measured retention time constants of I_{M1} with various values of V_{in} are shown in Figure 6(b). The histogram of the measured τ_{ret} of I_{M1} for 20 chips (2 measurements each) with $V_{in} = 0.7$ V (inset) shows that τ_{ret} ranges from 0.3 ms to 2 ms. The simulated τ_{ret} for I_{M1} is also observed in the millisecond range (~ same range), shown in Table 2. This measured retention time constant in the milliseconds range is sufficient for ANN training with a few network levels to have performance accuracy with minimal or without degradation.

In summary, the ATOM characterization presented above provides the retention time constant, in milliseconds, for a given V_{in} range (0.4 V to 0.9 V). Moreover, the switch using M2 is used for the read operation, which results in no static power with the energy needed for a read operation as ~1.6 pJ.

Now, the adequacy of ATOM parameters for intermediate data storage during ANN training must be evaluated. Therefore, the impact of ATOM parameters on the performance of the ANN training system is discussed in the following section.

5 System Performance Evaluation

We simulated the system in PyTorch to analyze the impact of ATOM parameters on system performance during ANN training. The variations in ATOM retention time are applied to the memory, which is responsible for storing intermediate data. As we aim to analyze the impact of ATOM on system accuracy performance, all remaining circuits involved in ANN training are assumed to be immune to variations and non-idealities. The subsequent section describes the simulation model utilized to evaluate and analyze system performance with the ATOM retention effect.

5.1 Performance Evaluation

To evaluate system performance, we have used various network architectures, including AlexNet [23] and VGG [39], and different datasets like MNIST, CIFAR-10, and CIFAR-100 [22]. The models were trained using the SGD optimizer [27] with a momentum of 0.9, an initial learning rate of 0.05, and a batch size of 256. We train the model using a softmax cross-entropy loss and run it for 200 epochs. All our results are averaged over 5 different runs.

The training and inference are performed using a MAC with 8-bit quantized weights. For the evaluation model, we consider the SRAM memory array for IMC MAC with 7T SRAM bit-cells to store the weights and to perform bidirectional MAC [2, 17]. The analog shift-add scheme, discussed in [16], is considered in the periphery to obtain the weighted-bit precision addition. Our previous work [40] modeled ATOM's effects for storing the weights and biases. In this work, we store the intermediate memory components, i.e., activations, errors, and updates, in ATOM. The appropriate time to perform the compute or memory operations, i.e., operation time constant (τ_{op}), is taken to be 10 ns.

As shown in Figure 7(a), we investigated the impact of varying τ_{ret} across a broad range, from 10^3 ms to 10^{-6} ms, for various network architectures, including AlexNet [23] and VGG [39] with the CIFAR-10 dataset [22]. We observe minimal performance degradation over a significant span of τ_{ret} values, specifically when $\tau_{ret} \geq 10^{-4}$ ms. As can be seen in Figure 7(b), for $\tau_{ret} \geq 10^{-4}$ ms, the accuracy remains comparable to the non-ATOM scenario, i.e., for ideal analog memory with infinite τ_{ret} or no decay. Moreover, a mild regularization effect is observed in Figure 7(b), as the ATOM decay effectively serves as a weak regularizer within a suitable range [40]. For $\tau_{ret} < 10^{-4}$ ms, the training starts to become highly unstable due to extremely high regularization and fails to converge (i.e., validation accuracy drops to 10%), as can be observed in Figure 7(a).

As seen from Figure 7(a) and (c), with the depth of the network (from AlexNet to VGG-19), the required retention time to store intermediate data increases. As ATOM needs to store the intermediate data during ANN training (for the given input sample), the required retention time increases with the depth of the neural network (i.e., the number of layers). The deeper the network, the longer the ATOM retention time must be since the intermediate data from the first layer is needed until the FP (from the first to the last layer) and the BP (from the last to the first layer) are completed for the given input sample. The required retention time for the VGG-19 network with 19 layers is higher than the AlexNet network (with eight layers), as can be seen in Figure 7(c).

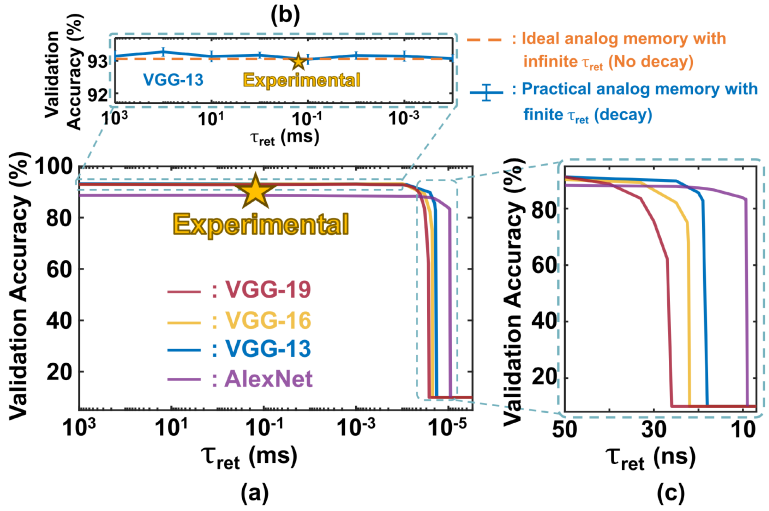


Fig. 7. Performance (accuracy) evaluation of various neural network architectures with ATOM retention constant and CIFAR-10 datasets: (a) Validation accuracy vs. retention time constant (τ_{ret}) for the CIFAR-10 dataset with VGG-19, VGG-16, VGG-13, and AlexNet network architectures and CIFAR-10 dataset. The τ_{ret} at which the system fails to converge, and validation accuracy drops to ~10%, increases with the depth of the network (from AlexNet to VGG-19). (b) Validation accuracy vs. retention time constant (τ_{ret}) for VGG-13 network architecture with CIFAR-10 datasets. No performance degradation is observed at the experimentally measured retention time constant of ~0.66 ms (highlighted by the yellow star) compared to the ideal analog memory with no decay or infinite τ_{ret} (shown by orange dotted line). (c) Validation accuracy vs. retention time constant (τ_{ret}) in normal scale to highlight the minimum τ_{ret} for the given neural network architectures at which the network fails to converge.

However, it is still very low compared to the experimentally measured retention time constant of ~0.66 ms of ATOM (highlighted by the yellow star), shown in Figure 7(a). Therefore, we can conclude that ATOM, with a given retention time and operation time, can be used for complex network architectures like VGG-19.

Moreover, we have evaluated the efficacy of ATOM (with a given retention time constant) for various datasets (such as MNIST, CIFAR-10, and CIFAR-100) with the VGG-13 neural network architecture, as presented in Table 3. Table 3 presents the evaluated system-level classification accuracy of the neural network with ATOM and other memory technologies presented in the literature. Various memory technologies for intermediate data storage have been presented in the literature [17, 31, 34, 38], and their performance is evaluated for various neural network architectures and datasets, as shown in Table 3. It can be observed from Table 3 that the VGG-13 network with ATOM and a measured retention time of 0.66 ms gives relatively the same classification performance as the baseline. It shows the feasibility of neural networks with ATOM for various datasets, including simpler datasets like MNIST (with 10 classes and grayscale images) and complex datasets like CIFAR-100 (with 100 classes and color images).

To gain a deeper understanding of the impact of the computational complexity of various datasets on system performance, we calculated the **floating point operations (FLOPs)** and MAC operations [42] for the VGG-13 network architecture across three datasets: MNIST, CIFAR-10, and CIFAR-100, as shown in Table 3. Due to the smaller grayscale image dimensions ($28 \times 28 \times 1$) of MNIST, it requires significantly fewer FLOPs and MACs compared to the color images ($32 \times 32 \times 3$) in CIFAR-10 and CIFAR-100, resulting in approximately 32% fewer computations. The identical

Table 3. System Level Performance Accuracy and Operational Complexity of Neural Network Architecture with ATOM and with Different Memory Technologies (Presented in Literature) for Intermediate Data Storage

		TComp'20 [17]	ISCA'16 [38]	JSSC'22 [34]	TVLSI'24 [31]	Proposed		
Memory Technology for Intermediate Data Storage		DRAM	eDRAM	SRAM	SRAM + eDRAM	ATOM		
Technology Node		7 nm	32 nm	40 nm	45 nm	45 nm		
Neural Network Architecture		ResNet-18	CNN/ DNN	ResNet-18	DNN, MLP, and GAN	VGG-13		
Dataset		ImageNet	--[a]	ImageNet	MNIST, CIFAR-10, CIFAR-100, ImageNet	MNIST	CIFAR-10	CIFAR-100
Classification Accuracy (%)	Baseline[b]	--[a]	--[a]	~75	Relative accuracy close to 1	99.34	93.81	73.81
	With Measured Data[c]					99.29	93.14	72.59
FLOPs		--[a]	--[a]	--[a]	--[a]	335 M	494 M	495 M
MACs Operations		--[a]	--[a]	--[a]	--[a]	167 M	247 M	248 M

[a]Not explicitly mentioned.
[b]System level performance accuracy with ideal analog intermediate data storage memory (without decay over time).
[c]System level performance accuracy with experimentally measured retention of ATOM that is 0.66 ms.

input image dimensions for CIFAR-10 and CIFAR-100 result in the overall FLOPs and MACs being nearly the same across both datasets. However, the slight rise in FLOPs and MACs, especially in the fully connected layers of VGG-13, for CIFAR-100 is due to an increased number of output classes (100 classes). As depicted in Table 3, the performance accuracy of the VGG-13 network decreases from MNIST to CIFAR-100, highlighting the impact of the growing complexity of the datasets on system accuracy performance.

Next, we analyze the impact of variability in the ATOM's retention time constant on the system performance with CIFAR-10 datasets and the VGG-13 neural network architecture. We use a fixed value of $\tau_{ret} = 1$ ms, which lies comfortably within the ideal range of τ_{ret} where we observe the slight regularization benefit, as can be seen in Figure 7(b). The leakage current variation of the write transistor M6 of ATOM unit cells dominates cell-to-cell variability in the retention time constant. The threshold voltage of M6, i.e., $V_{Th,M6}$, is Gaussian distributed, and τ_{ret} exponentially depends on $V_{Th,M6}$. Thus, τ_{ret} is sampled from a log-normal distribution:

$$\tau_{ret,0}.e^{\left(\frac{\Delta V_{Th,M6}}{\eta V_t}\right)},$$

where $\tau_{ret,0}$ is the nominal value of τ_{ret}, $\Delta V_{Th,M6}$ is a Gaussian random variable with zero mean and standard deviation $\sigma_{V_{Th,M6}}$. V_t is the thermal voltage (26 mV at room temperature), and η is an ideality factor (~1.5).

We conducted simulations for various values of $\sigma_{V_{Th}}$ for M6 of ATOM, denoted as $\sigma_{V_{Th,M6}}$, and tracked the test accuracies, as shown in Table 4. Our observation reveals either no or minimal performance decline for $\sigma_{V_{Th,M6}}$ values up to 80 mV. This suggests performance robustness against the variability in ATOM's retention time. Consequently, we conclude that on-chip training using the proposed ATOM remains robust in the face of on-chip variations and charge loss over time within ATOM.

Therefore, based on the results illustrated in Figure 7 and Table 3, we can conclude that ATOM, when configured with an appropriate retention time constant, can be applied effectively across

Table 4. Accuracy Evaluation of VGG-13 Neural Network Architecture with ATOM and CIFAR-10 Datasets for Various $\sigma_{V_{Th,M6}}$ Values for τ_{ret} = 1 ms

$\sigma_{V_{Th,M6}}$ (mV)	Test Accuracy (%)
10	93.18 ± 0.16
20	93.10 ± 0.05
40	93.28 ± 0.05
80	93.23 ± 0.13

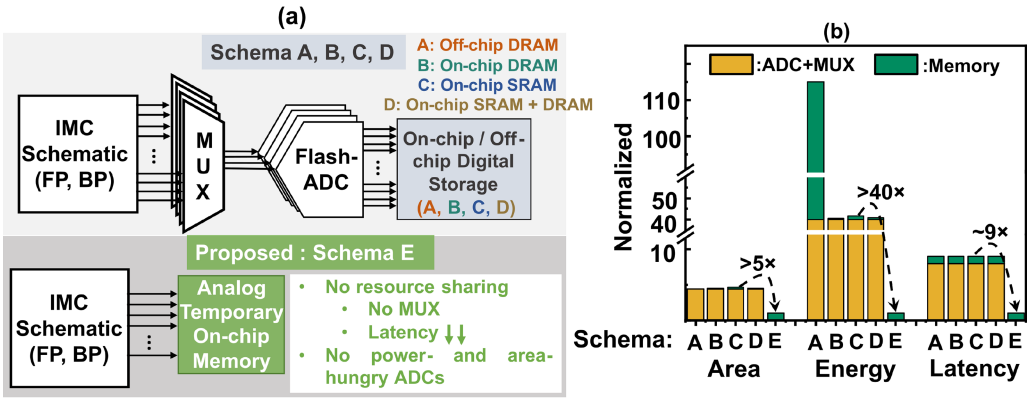


Fig. 8. Comparison between conventional and proposed schema for FP and BP operation of ANN training: (a) Conventional and proposed schema for intermediate data storage during FP and BP. (b) Comparison between various schemas for intermediate data storage during FP and BP.

various datasets and neural network architectures. Furthermore, as depicted in Figure 7, the required time constant for the presented neural network architecture can be minimized up to $\tau_{ret} \approx 10^{-4}$ ms without affecting system performance; it enables the reduction in the ATOM cell area (at least by 2×, as presented in Figure 5). This area reduction allows for more memory cells to be integrated within a given memory area. Moreover, as highlighted in Table 4, on-chip training using ATOM for intermediate data storage exhibits robustness against on-chip variations and charge loss over time within the ATOM.

5.2 Hardware Benefits

This subsection compares the area, energy consumption, and number of clock cycles (i.e., latency) required during FP and BP for intermediate data storage. The FP and BP operation in the schema with ATOM against various conventional schemas that utilize digital memory for intermediate data storage is shown in Figure 8(a). For a baseline, we considered different intermediate data storage memory technologies presented in the literature, including off-chip DRAM [17], on-chip DRAM (eDRAM) [38, 43, 48], SRAM [34], and SRAM+eDRAM [31], and labeled them as Schemas A, B, C, and D, respectively.

As shown in Figure 8(a), Schemas A, B, C, and D store the large volumes of intermediate data generated during ANN inference and training in digital memory. They rely on energy- and area-intensive ADCs to store intermediate data in digital memory because IMC output (which represents

Table 5. Estimated Resources for Intermediate Data Storage at Macro-level (128 × 128 Memory Array) by an ANN Hidden Layer with Various Schema during FP and BP Operation of Inference and Training (Estimated at 45 nm Technology Node)

Schema		A: DRAM [17]	B: eDRAM [38, 43, 48]	C: SRAM [34]	D: SRAM + eDRAM [31]	E: ATOM (Proposed)
Area (mm^2)	ADC + MUX	0.056	0.056	0.056	0.056	NA[a]
	Storage	−−[b]	0.001	0.003	0.002	0.012
	Total	−−[b]	0.057	0.059	0.058	0.012
Energy (nJ)	ADC + MUX	16.5	16.5	16.5	16.5	NA[a]
	Storage	30.7	0.18	0.64	0.24	0.41
	Total	46.7	16.6	17.1	16.7	0.41
Latency (No. of clock cycles)	ADC + MUX	8	8	8	8	NA[a]
	Storage	1	1	1	1	1
	Total	9	9	9	9	1
Throughput (GOPS)[c,d]		~102	128	128	128	~171
Energy Efficiency (TOPS/W)[c,d]		5.36	14.30	13.95	14.26	124.57

[a]Not Applicable: As ADCs and MUX are not required in the proposed schema.
[b]Off-chip DRAM area is not considered here (as a separate IC is required for DRAM).
[c]Resources required to perform MAC operation are taken into account in addition to intermediate storage.
[d]1 Operation = 1-bit Weight × analog Input.
GOPS represents Giga-operations Per Second, whereas TOPS/W represents Tera-operations Per Second per Watt.

a significant portion of intermediate data) is in the analog domain. To minimize the number of necessary ADCs, a multiplexer (MUX) is used to share one ADC between eight IMC analog outputs [17]. Although this reduces the area requirement, it increases the latency or the required number of clock cycles. Moreover, Schema A requires energy-hungry off-chip communication to store the intermediate data to off-chip digital memory. To reduce the area and power costs of ADCs, some studies suggest using analog memory for intermediate data storage [4, 15]. However, in [15], analog memory is demonstrated only for inference operations. Chang et al. [4] offer analog memory for inference and training, but its hardware has not been experimentally verified, and no detailed information about the analog memory retention requirement is available. The proposed ATOM for intermediate data storage (Schema E) does not require analog-to-digital conversion and minimizes off-chip communication for intermediate data storage, which are challenges associated with existing schemas. The benefits of this approach, in terms of area, energy, latency, throughput, and energy efficiency, are summarized in Table 5 for FP and BP operations of ANN training.

For comparison, presented in Table 5 and Figure 8(b), we have considered an ANN hidden layer with an IMC schematic using eight tiles of 128 × 128 memory crossbar array [17]. The parameters in Table 5 are estimated at the macro level for a 128 × 128 memory crossbar array macro, which serves as a foundational building block for ANNs. This macro efficiently performs MVM operations for inference and training of ANN using IMCs independent of the underlying network architecture or dataset [17, 33]. Therefore, the parameters in Table 5 are valid for all network architectures (e.g., AlexNet, VGG-13, VGG-16, and VGG-19) and datasets (e.g., MNIST, CIFAR-10, and CIFAR-100) discussed in this article and are not limited to just these examples.

Conventional memory technology, such as SRAM [17] or emerging memory technology, e.g., RRAM [9, 15, 36], can be used in this 128 × 128 memory crossbar array macro to perform the IMC operation. The resources required by the memory, ADCs, and MUX to store the intermediate data

generated by the ANN hidden layer for various schemas (shown in Figure 8(a)) are compared in Table 5. As the IMC schematic is common for all schemas and can use either SRAM or RRAM crossbar array, it is not considered in area, energy, and latency estimation presented in Table 5 and Figure 8(b) [31]. However, it is considered in the analysis of throughput and energy efficiency.

As shown in Figure 8(b), the proposed schema (Schema E) offers a significant improvement compared to on-chip SRAM (Schema C) for intermediate data storage, with a ∼5× reduction in area, a ∼40× reduction in energy, and a ∼9× decrease in the required number of clock operation for FP and BP. This is because the primary resource-consuming factor in a conventional schema is an ADC, with an estimated area of ∼0.007 mm^2 and energy of ∼0.065 nJ for 45 nm technology [16], which is entirely avoided in the proposed Schema E for FP and BP. Moreover, off-chip communication is a significant source of energy consumption in Schema A. However, in the proposed Schema E, this issue is significantly mitigated by utilizing ATOM to store a substantial volume of intermediate data, minimizing the need for off-chip communication. The off-chip DRAM communication is the costliest operation, which takes nearly 120 pJ/B [12, 26], compared to 0.7 pJ/B for eDRAM [26], 2.5 pJ/B for on-chip SRAM [12], 0.925 pJ/B for SRAM+eDRAM [31], and 1.6 pJ for the proposed ATOM. Further, the proposed Schema E achieves a latency improvement of ∼9× as multiplexers are not required at the periphery compared to other schemas. The throughput and energy efficiency (in TOPS/W) are estimated for all schemas, considering a 45 nm technology node and an operating frequency of 500 MHz. Schema A has lower throughput and energy efficiency than other schemas due to off-chip communication in addition to ADC usage. However, higher throughput and energy efficiency can be achieved with Schema E, as no ADCs and off-chip communication are required during FP and BP for intermediate data storage compared to other schemas.

Therefore, with its significant advantages in energy, latency, throughput, and overall performance, primarily resulting from the elimination of ADC usage during FP and BP, Schema E, which integrates the IMC unit with the proposed ATOM, presents a highly compelling solution for on-chip training. Schema E surpasses traditional digital memory solutions for intermediate data storage, utilizing analog data storage with ATOM, providing a more efficient and effective alternative for ANN applications.

6 Conclusion

We present a novel ATOM prototype for intermediate data storage during on-chip training and inference of ANNs in the analog environment. Our proposal's key concept is maintaining the intermediate data storage on-chip and in the analog domain, leading to ANN training with minimum ADCs. Furthermore, system hardware variability awareness in ANN, which improves performance accuracy, is achieved using on-chip training. Our proposed analog memory is designed to complement the operation in the analog domain. Our approach avoids needing area-intensive and power-hungry ADCs and memories compared to conventional intermediate data storage schemas. The proposed schema yields at least 5× improvement in the area, 40× in energy, and 9× in latency.

References

[1] Nihar Athreyas, Wenhao Song, Blair Perot, Qiangfei Xia, Abbie Mathew, Jai Gupta, Dev Gupta, and J. Joshua Yang. 2018. Memristor-CMOS analog coprocessor for acceleration of high-performance computing applications. *Journal on Emerging Technologies in Computing Systems* 14, 3 (Nov. 2018), Article 38, 30 pages. DOI: https://doi.org/10.1145/3269985

[2] Kyeongryeol Bong, Sungpill Choi, Changhyeon Kim, Sanghoon Kang, Youchang Kim, and Hoi-Jun Yoo. 2017. 14.6 A 0.62mW ultra-low-power convolutional-neural-network face-recognition processor and a CIS integrated with always-on Haar-like face detector. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 248–249. DOI: https://doi.org/10.1109/ISSCC.2017.7870354

[3] Lylia Thiziri Chabane, Dang-Kièn Germain Pham, Paul Chollet, and Patricia Desgreys. 2021. Design method of analog sigmoid function and its approximate derivative. In *2021 XXXVI Conference on Design of Circuits and Integrated Systems (DCIS)*, 1–5. DOI: https://doi.org/10.1109/DCIS53048.2021.9666181

[4] H.-Y. Chang, P. Narayanan, S. C. Lewis, N. C. P. Farinha, K. Hosokawa, C. Mackin, H. Tsai, S. Ambrogio, A. Chen, and G. W. Burr. 2019. AI hardware acceleration with analog memory: Microarchitectures for low energy at high speed. *IBM Journal of Research and Development* 63, 6 (2019), 8:1–8:14. DOI: https://doi.org/10.1147/JRD.2019.2934050

[5] Ethan Chen and Vanessa Chen. 2020. In-sensor time-domain classifiers using pseudo sigmoid activation functions. *Integration* 73 (2020), 43–49. DOI: https://doi.org/10.1016/j.vlsi.2020.03.002

[6] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138. DOI: https://doi.org/10.1109/JSSC.2016.2616357

[7] Zhengyu Chen, Xi Chen, and Jie Gu. 2021. 15.3 A 65nm 3T dynamic analog RAM-based computing-in-memory macro and CNN accelerator with retention enhancement, adaptive analog sparsity and 44TOPS/W system energy efficiency. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64, 240–242. DOI: https://doi.org/10.1109/ISSCC42613.2021.9366045

[8] Zhiyu Chen, Zhanghao Yu, Qing Jin, Yan He, Jingyu Wang, Sheng Lin, Dai Li, Yanzhi Wang, and Kaiyuan Yang. 2021. CAP-RAM: A charge-domain in-memory computing 6T-SRAM for accurate and precision-programmable CNN Inference. *IEEE Journal of Solid-State Circuits* 56, 6 (June 2021), 1924–1935. DOI: https://doi.org/10.1109/JSSC.2021.3056447

[9] Shreyas Deshmukh, Vivek Saraswat, Venkatesh Gopinath, Rajesh Nair, Laxmeesha Somappa, Maryam S. Baghini, and Udayan Ganguly. 2023. ANN Inference enabled by variability mitigation using 2T-1R bit cell-based design space analysis. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1–5. DOI: https://doi.org/10.1109/ISCAS46773.2023.10181912

[10] Anteneh Gebregiorgis, Hoang Anh Du Nguyen, Jintao Yu, Rajendra Bishnoi, Mottaqiallah Taouil, Francky Catthoor, and Said Hamdioui. 2022. A survey on memory-centric computer architectures. *Journal on Emerging Technologies in Computing Systems* 18, 4 (Oct. 2022), Article 79, 50 pages. DOI: https://doi.org/10.1145/3544974

[11] Sujan Kumar Gonugondla, Mingu Kang, and Naresh Shanbhag. 2018. A 42pJ/decision 3.12TOPS/W robust in-memory machine learning classifier with on-chip training. In *2018 IEEE International Solid-State Circuits Conference (ISSCC)*, 490–492. DOI: https://doi.org/10.1109/ISSCC.2018.8310398

[12] Mark Horowitz. 2014. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 10–14. DOI: https://doi.org/10.1109/ISSCC.2014.6757323

[13] Akhilesh Jaiswal, Indranil Chakraborty, Amogh Agrawal, and Kaushik Roy. 2019. 8T SRAM cell as a multibit dot-product engine for beyond von Neumann computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 11 (2019), 2556–2567. DOI: https://doi.org/10.1109/TVLSI.2019.2929245

[14] Hongyang Jia, Hossein Valavi, Yinqi Tang, Jintao Zhang, and Naveen Verma. 2020. A programmable heterogeneous microprocessor based on bit-scalable in-memory computing. *IEEE Journal of Solid-State Circuits* 55, 9 (2020), 2609–2621. DOI: https://doi.org/10.1109/JSSC.2020.2987714

[15] Hongwu Jiang, Shanshi Huang, Wantong Li, and Shimeng Yu. 2023. ENNA: An efficient neural network accelerator design based on ADC-free compute-in-memory subarrays. *IEEE Transactions on Circuits and Systems I: Regular Papers* 70, 1 (2023), 353–363. DOI: https://doi.org/10.1109/TCSI.2022.3208755

[16] Hongwu Jiang, Wantong Li, Shanshi Huang, Stefan Cosemans, Francky Catthoor, and Shimeng Yu. 2022. Analog-to-digital converter design exploration for compute-in-memory accelerators. *IEEE Design & Test* 39, 2 (2022), 48–55. DOI: https://doi.org/10.1109/MDAT.2021.3050715

[17] Hongwu Jiang, Xiaochen Peng, Shanshi Huang, and Shimeng Yu. 2020. CIMAT: A compute-in-memory architecture for on-chip training based on transpose SRAM arrays. *IEEE Transactions on Computers* 69, 7 (2020), 944–954. DOI: https://doi.org/10.1109/TC.2020.2980533

[18] Honggu Kim, Yerim An, Ryunyeong Kim, Sunyoung Kim, and Yong Shim. 2024. Union SRAM: PVT variation auto-compensated, bit precision configurable current mode 8T SRAM in memory MAC macro. *IEEE Access* 12 (2024), 162882–162893. DOI: https://doi.org/10.1109/ACCESS.2024.3487975

[19] Hyunjoon Kim, Taegeun Yoo, Tony Tae-Hyoung Kim, and Bongjin Kim. 2021. Colonnade: A reconfigurable SRAM-based digital bit-serial compute-in-memory macro for processing neural networks. *IEEE Journal of Solid-State Circuits* 56, 7 (2021), 2221–2233. DOI: https://doi.org/10.1109/JSSC.2021.3061508

[20] Yulhwa Kim, Hyungjun Kim, and Jae-Joon Kim. 2022. Extreme partial-sum quantization for analog computing-in-memory neural network accelerators. *Journal on Emerging Technologies in Computing Systems* 18, 4 (Oct. 2022), Article 75, 19 pages. DOI: https://doi.org/10.1145/3528104

[21] Maha Kooli, Antoine Heraud, Henri-Pierre Charles, Bastien Giraud, Roman Gauchi, Mona Ezzadeen, Kevin Mambu, Valentin Egloff, and Jean-Philippe Noel. 2022. Towards a truly integrated vector processing unit for memory-bound applications based on a cost-competitive computational SRAM design solution. *Journal on Emerging Technologies in Computing Systems* 18, 2 (April 2022), Article 40, 26 pages. DOI: https://doi.org/10.1145/3485823

[22] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning Multiple Layers of Features from Tiny Images*. Technical Report 0. University of Toronto, Toronto, Ontario. Retrieved from https://www.cs.toronto.edu/kriz/learning-features-2009-TR.pdf

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, Vol. 25. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[24] Navjot Kukreja, Alena Shilova, Olivier Beaumont, Jan Huckelheim, Nicola Ferrier, Paul Hovland, and Gerard Gorman. 2019. Training on the edge: The why and the how. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 899–903. DOI: https://doi.org/10.1109/IPDPSW.2019.00148

[25] Dinesh Kushwaha, Jaya Kumar Abotula, Rajat Kohli, Jwalant Mishra, Sudeb Dasgupta, and Anand Bulusu. 2024. Multi-bit compute-in memory architecture using a C-2C ladder network. *IEEE Transactions on Circuits and Systems II: Express Briefs* 71, 6 (2024), 3166–3170. DOI: https://doi.org/10.1109/TCSII.2023.3329261

[26] Haitong Li, Mudit Bhargava, Paul N. Whatmough, and H.-S. Philip Wong. 2019. On-chip memory technology design space explorations for mobile deep neural network accelerators. In *56th Annual Design Automation Conference 2019 (DAC '19)*. ACM, New York, NY, Article 131, 6 pages. DOI: https://doi.org/10.1145/3316781.3317874

[27] Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic gradient descent with warm restarts. arXiv:1608.03983. Retrieved from https://arxiv.org/abs/1608.03983

[28] Mahta Mayahinia, Abhairaj Singh, Christopher Bengel, Stefan Wiefels, Muath A. Lebdeh, Stephan Menzel, Dirk J. Wouters, Anteneh Gebregiorgis, Rajendra Bishnoi, Rajiv Joshi, et al. 2022. A voltage-controlled, oscillation-based ADC design for computation-in-memory architectures using emerging ReRAMs. *Journal on Emerging Technologies in Computing Systems* 18, 2 (March 2022), Article 32, 25 pages. DOI: https://doi.org/10.1145/3451212

[29] Sparsh Mittal, Gaurav Verma, Brajesh Kaushik, and Farooq A. Khanday. 2021. A survey of SRAM-based in-memory computing techniques and applications. *Journal of Systems Architecture* 119 (2021), 102276. DOI: https://doi.org/10.1016/j.sysarc.2021.102276

[30] Ali Bou Nassif, Ismail Shahin, Imtinan Attili, Mohammad Azzeh, and Khaled Shaalan. 2019. Speech recognition using deep neural networks: A systematic review. *IEEE Access* 7 (2019), 19143–19165. DOI: https://doi.org/10.1109/ACCESS.2019.2896880

[31] Duy-Thanh Nguyen, Abhiroop Bhattacharjee, Abhishek Moitra, and Priyadarshini Panda. 2024. MCAIMem: A mixed SRAM and eDRAM cell for area and energy-efficient on-chip AI memory. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 32, 11 (2024), 2023–2036. DOI: https://doi.org/10.1109/TVLSI.2024.3439231

[32] Leibin Ni, Hantao Huang, Zichuan Liu, Rajiv V. Joshi, and Hao Yu. 2017. Distributed in-memory computing on binary RRAM crossbar. *Journal on Emerging Technologies in Computing Systems* 13, 3, (March 2017), Article 36, 18 pages. DOI: https://doi.org/10.1145/2996192

[33] Xiaochen Peng, Shanshi Huang, Hongwu Jiang, Anni Lu, and Shimeng Yu. 2021. DNN+NeuroSim V2.0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 11 (2021), 2306–2319. DOI: https://doi.org/10.1109/TCAD.2020.3043731

[34] Kartik Prabhu, Albert Gural, Zainab F. Khan, Robert M. Radway, Massimo Giordano, Kalhan Koul, Rohan Doshi, John W. Kustin, Timothy Liu, Gregorio B. Lopes, et al. 2022. CHIMERA: A 0.92-TOPS, 2.2-TOPS/W edge AI accelerator with 2-MByte on-chip foundry resistive RAM for efficient training and inference. *IEEE Journal of Solid-State Circuits* 57, 4 (2022), 1013–1026. DOI: https://doi.org/10.1109/JSSC.2022.3140753

[35] Ximing Qiao, Xiong Cao, Huanrui Yang, Linghao Song, and Hai Li. 2018. Atomlayer: A universal reRAM-based CNN accelerator with atomic layer computation. In *55th Annual Design Automation Conference (DAC '18)*. ACM, New York, NY, Article 103, 6 pages. DOI: https://doi.org/10.1145/3195970.3195998

[36] Shubham Sahay, Mohammad Bavandpour, Mohammad Reza Mahmoodi, and Dmitri Strukov. 2020. A 2T-1R cell array with high dynamic range for mismatch-robust and efficient neurocomputing. In *2020 IEEE International Memory Workshop (IMW)*, 1–4. DOI: https://doi.org/10.1109/IMW48823.2020.9108142

[37] Abu Sebastian, Manuel Le Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou. 2020. Memory devices and applications for in-memory computing. *Nature Nanotechnology* 15, 7 (2020), 529–544. DOI: https://doi.org/10.1038/s41565-020-0655-z

[38] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 14–26. DOI: https://doi.org/10.1109/ISCA.2016.12

[39] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556. Retrieved from https://arxiv.org/abs/1409.1556

[40] Raghav Singhal, Vivek Saraswat, Shreyas Deshmukh, Sreenivas Subramoney, Laxmeesha Somappa, Maryam Shojaei Baghini, and Udayan Ganguly. 2023. Enhanced regularization for on-chip training using analog and temporary memory weights. *Neural Networks* 165 (2023), 1050–1057. DOI: https://doi.org/10.1016/j.neunet.2023.07.001

[41] Jian-Wei Su, Xin Si, Yen-Chi Chou, Ting-Wei Chang, Wei-Hsing Huang, Yung-Ning Tu, Ruhui Liu, Pei-Jung Lu, Ta-Wei Liu, Jing-Hong Wang, et al. 2022. Two-way transpose multibit 6T SRAM computing-in-memory macro for inference-training AI edge chips. *IEEE Journal of Solid-State Circuits* 57, 2 (2022), 609–624. DOI: https://doi.org/10.1109/JSSC.2021.3108344

[42] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2020. How to evaluate deep neural network processors: TOPS/W (alone) considered harmful. *IEEE Solid-State Circuits Magazine* 12, 3 (2020), 28–41. DOI: https://doi.org/10.1109/MSSC.2020.3002140

[43] Fengbin Tu, Weiwei Wu, Shouyi Yin, Leibo Liu, and Shaojun Wei. 2018. RANA: Towards efficient neural acceleration with refresh-optimized embedded DRAM. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 340–352. DOI: https://doi.org/10.1109/ISCA.2018.00037

[44] Naveen Verma, Hongyang Jia, Hossein Valavi, Yinqi Tang, Murat Ozatay, Lung-Yen Chen, Bonan Zhang, and Peter Deaville. 2019. In-memory computing: Advances and prospects. *IEEE Solid-State Circuits Magazine* 11, 3 (2019), 43–55. DOI: https://doi.org/10.1109/MSSC.2019.2922889

[45] Hechen Wang, Renzhi Liu, Richard Dorrance, Deepak Dasalukunte, Dan Lake, and Brent Carlton. 2023. A charge domain SRAM compute-in-memory macro with C-2C ladder-based 8-bit MAC unit in 22-nm FinFET process for edge inference. *IEEE Journal of Solid-State Circuits* 58, 4 (2023), 1037–1050. DOI: https://doi.org/10.1109/JSSC.2022.3232601

[46] Bo Zhang, Shihui Yin, Minkyu Kim, Jyotishman Saikia, Soonwan Kwon, Sungmeen Myung, Hyunsoo Kim, Sang Joon Kim, Jae-Sun Seo, and Mingoo Seok. 2023. PIMCA: A programmable in-memory computing accelerator for energy-efficient DNN inference. *IEEE Journal of Solid-State Circuits* 58, 5 (2023), 1436–1449. DOI: https://doi.org/10.1109/JSSC.2022.3211290

[47] Jintao Zhang, Zhuo Wang, and Naveen Verma. 2017. In-memory computation of a machine-learning classifier in a standard 6T SRAM array. *IEEE Journal of Solid-State Circuits* 52, 4 (2017), 915–924. DOI: https://doi.org/10.1109/JSSC.2016.2642198

[48] Sai Qian Zhang, Thierry Tambe, Nestor Cuevas, Gu-Yeon Wei, and David Brooks. 2024. CAMEL: Co-designing AI models and eDRAMs for efficient on-device learning. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 861–875. DOI: https://doi.org/10.1109/HPCA57654.2024.00071

Received 7 May 2024; revised 7 April 2025; accepted 22 August 2025

ACM Journal on Emerging Technologies in Computing Systems
https://jetc.acm.org/

Guide to Manuscript Submission
Submission to the *ACM Journal on Emerging Technologies in Computing Systems* is done electronically through https://mc.manuscriptcentral.com/jetc. Once you are at that site, you can create an account and password with which you can enter the ACM Manuscript Central manuscript review tracking system. Proceed to the Author Center to submit your manuscript and your accompanying files.

You will be asked to create an abstract that will be used throughout the system as a synopsis of your paper. You will also be asked to classify your submission using the ACM Computing Classification System through a link provided at the Author Center. For completeness, please select at least one primary-level classification followed by two secondary-level classifications. To make the process easier, you may cut and paste from the list. Remember, you, the author, know best which area and sub-areas are covered by your paper; in addition to clarifying the area where your paper belongs, classification often helps in quickly identifying suitable reviewers for your paper. So it is important that you provide as thorough a classification of your paper as possible.

The ACM Production Department prefers that your manuscript be prepared in either LaTeX or MS Word format. Style files for manuscript preparation can be obtained at the following location: https://www.acm.org/publications/authors/submissions. For editorial review, the manuscript should be submitted as a PDF or Postscript file. Accompanying material can be in any number of text or image formats, as well as software/documentation bundles in zip or tar-gzipped formats.

Questions regarding editorial review process should be directed to the Editor-in-Chief. Questions regarding the post-acceptance production process should be addressed to the Associate Editor, Stacey Schick, at schick@hq.acm.org.

Subscription and Membership Information.

Send orders to:

ACM Member Services Dept.
General Post Office
PO Box 30777
New York, NY 10087-0777

For information, contact:

Mail: ACM Member Services Dept.
1601 Broadway, 10th Floor
New York, NY 10019-7434
Phone: +1-212-626-0500
Fax: +1-212-944-1318
Email: acmhelp@acm.org
Catalog: https://www.acm.org/publications/alacarte

About ACM. ACM is the world's largest educational and scientific computing society, uniting educators, researchers and professionals to inspire dialogue, share resources and address the field's challenges. ACM strengthens the computing profession's collective voice through strong leadership, promotion of the highest standards, and recognition of technical excellence. ACM supports the professional growth of its members by providing opportunities for life-long learning, career development, and professional networking.

Visit ACM's Website. https://www.acm.org.