Toward Secure SoCs with Dynamic Information Flow Tracking

Ali Ait Hassou, William Pensec, Florent Bruguier and Pascal Benoit
LIRMM, Univ. Montpellier, Montpellier, France

***Abstract*—The increasing number of IoT devices worldwide raises a major security concern, especially when these systems process sensitive data and interact with untrusted user input. In this work, we propose a Dynamic Information Flow Tracking (DIFT) mechanism integrated at a System On Chip (SoC) level to protect against memory corruption attacks such as buffer overflow and format string exploits. Our approach extends an in-house RISC-V SoC by modifying the core microarchitecture, adding memory tagging support, transporting tags alongside the data across the system and integrating a hardware auto-tagging feature. We evaluate the proposed design in terms of hardware overhead and report FPGA-based experiments using representative IoT application benchmarks.**

***Index Terms*—Dynamic Information Flow Tracking, RISC-V, SoC Security, Memory Tagging, Hardware Auto Tagging, FPGA**

I. Introduction

IoT devices are deployed across diverse domains, from home automation to healthcare and industrial manufacturing, making them attractive for attackers. The wider the input surface of these systems, the more exposed they are to threats, primarily memory corruption vulnerabilities, which can lead to sensitive data leakage, data corruption, or control flow hijacking. In this context, protecting systems against such threats at run time is a priority.

Dynamic Information Flow Tracking (DIFT) is a promising approach to validate at run time that a program executes without security violations. The core idea is to classify data sources: system-generated data is labelled *trusted*, while data originating from external inputs such as network interfaces, sensors and user commands is labelled *untrusted*. The mechanism tags data accordingly and propagates these tags through the system following a programmer-defined propagation policy. In security-critical operations, such as loading/storing data into memory, function returns, or executing branch conditions, a checking policy verifies the tags and raises an exception upon violation, preventing the attack from succeeding.

In this paper, we propose a full hardware DIFT integration into the ADAM RISC-V SoC [1] including core-level tag propagation and checking, memory tagging, tag propagation across the system interconnect, and a hardware auto-tagging unit that automatically tags data arriving from the peripheral interface. We evaluate the area overhead on FPGA and validate the design against representative IoT benchmarks, including buffer overflow and format string exploits.

II. Related Works

Hardware-based DIFT has been proposed at various levels of abstraction.

Tiwari et al. [2] introduced GLIFT at gate level, achieving complete tracking at the cost of prohibitive area overhead (up to 70%).

At the microarchitecture level, Suh et al. [3] extended a SPARC processor with a 1-bit tag per byte achieving 1.6% performance overhead, while Dalton et al. [4] introduced Raksha with 4-bit tags per word and configurable security policies via dedicated ISA extensions.

On RISC-V, Palmiero et al. [5] extended the Rocket core reporting roughly 1% performance and 12.5% memory overhead.

At the SoC level, Porquet et al. [6] extended the system bus to carry tags, while Piccolboni [7] implemented DIFT within IP wrappers for loosely coupled accelerators.

However, none of these combines in core integration, SoC wide tag propagation, and automatic peripheral tagging at the RTL level.

Our work will fill this gap by integrating DIFT directly into the CV32E40P RISC-V core [8] pipeline within the ADAM SoC, propagating tags across the system interconnect and automatically tagging data from peripheral interfaces.

III. Architecture

The DIFT mechanism is integrated at three levels of abstraction within the ADAM SoC: the processor core, memory, and system interconnect. Figure 2 illustrates the overall architecture.

A. In core DIFT extension

We extend the CV32E40P pipeline (D-CV32E40P) to propagate and check a 1-bit tag in parallel with data. The propagation and checking occur in the ID and EX stages in parallel without any pipeline stalls. The policies of propagation and checking are defined by two control/status registers (CSR):

- **Tag Propagation Register (TPR)**: defines the output operand tag as a function of the input operand tags and the instruction class provided by the decoder. For example, an ALU operation with at least one tagged operand produces a tagged result.
- **Tag Checking Register (TCR)**: defines which operand triggers the security check, so a tagged operand used in a check operation will trigger a DIFT exception. For

example, a tagged PC will trigger an exception to prevent control flow deviation.

B. Memory Tagging

As illustrated in Figure 1, we extend the memory words by an extra four bits to adopt a 1-bit tag per byte granularity. The load store unit of the CV32E40P issues tag reads and writes in parallel with data accesses to keep them concurrent and avoid the complexity of synchronisation.

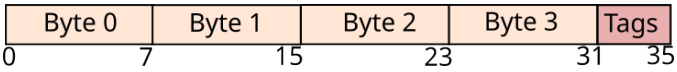

Fig. 1. 4-bit tags per memory word organization

C. SoC level Integration and Hardware Auto Tagging

In order to have tags synchronised with main data transfers and avoid bus width extension, we are transporting them across the AXI Interconnect using the `aruser ruser` and `awuser wuser` sideband signals. All fabric crossbar paths and bridges are extended to forward these signals transparently.

At the boundary between the system interconnect and the Advanced Peripheral Bus (APB), a hardware auto-tagging unit, as shown in Figure 2, is monitoring read responses from peripherals (UART, GPIO, SPI, TIMER). When enabled via a dedicated memory-mapped register, it forces `ruser = 4'b1111` on selected peripheral read responses, marking them as tagged. This hardware auto-tagging feature will:

- Eliminates software intervention and reduces the attack window by tagging peripheral data atomically upon arrival.
- Removes run-time tagging overhead and enables transparent protection of unmodified legacy binaries.

IV. Initial Results

The design was implemented on a Nexys Video board featuring a Xilinx Artix-7 XC7A200T FPGA, using Vivado 2024.2 as the synthesis and implementation toolchain. As shown in Table I, the DIFT integration introduces a lightweight overhead of 2.54% in LUTs and 3.73% in flip-flops on the CV32E40P core and 2.35% in LUTs and 1.98% in flip-flops at the full SoC level, confirming the feasibility of embedding it into real-world IoT devices.

TABLE I
FPGA Resource Overhead of DIFT

Design	LUTs	ΔLUTs	Regs	ΔRegs
CV32E40P (base)	10 037	2.54%	3 858	3.73%
CV32E40P (DIFT)	10 292		4 002	
ADAM SoC (base)	24 437	2.35%	13 388	1.98%
ADAM SoC (DIFT)	25 011		13 653	

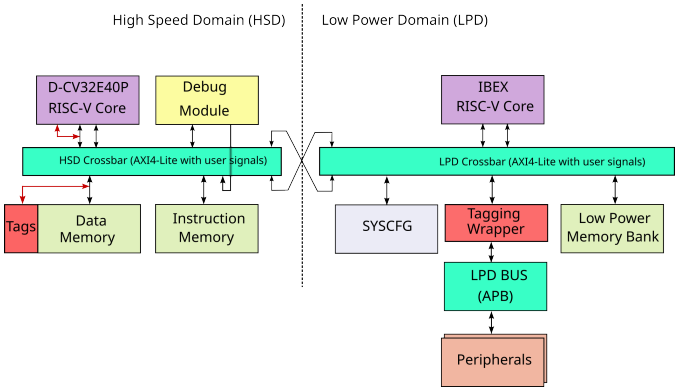

Fig. 2. Initial exploration architecture: an ADAM SoC instance with DIFT Integrated

V. Conclusion

Our work proposed an end-to-end integration of DIFT in Adam SoC; to our knowledge, this gap was not filled in prior works. The integration consists of modifying the micro architecture of the CV32E40P, extending the AXI4-Lite bus to cover side band user signals, extending memory for 4-bit tags per word, and implementing an auto-tagging unit that monitors and tags input data of our SoC. This implementation resulted in a negligible performance and area overhead at the full SoC level to prevent most known software attacks. As future work, we will be interested in a finer memory tag management field to reduce in advance more the overhead.

References

[1] Felipe Paiva Alencar et al. "ADAM: ADAptive Microcontroller Platform for Edge AI Systems". In: *Proceedings of the 36th International Workshop on Rapid System Prototyping*. RSP '25. Association for Computing Machinery, 2026, pp. 29–35. ISBN: 9798400722240. DOI: 10.1145/3768617.3770731. URL: https://doi.org/10.1145/3768617.3770731.

[2] Mohit Tiwari et al. "Complete Information Flow Tracking from the Gates Up". In: *Sigplan Notices - SIGPLAN* 44 (Mar. 2009), pp. 109–120. DOI: 10.1145/1508284.1508258.

[3] G. Suh et al. "Secure Program Execution Via Dynamic Information Flow Tracking". In: *ACM SIGPLAN Notices* 39 (Dec. 2004), pp. 85–96. DOI: 10.1145/1037947.1024404.

[4] Michael Dalton, Hari Kannan, and Christos Kozyrakis. "Raksha: a flexible information flow architecture for software security". In: *SIGARCH Comput. Archit. News* 35.2 (June 2007), pp. 482–493. ISSN: 0163-5964. DOI: 10.1145/1273440.1250722. URL: https://doi.org/10.1145/1273440.1250722.

[5] Christian Palmiero et al. "Design and Implementation of a Dynamic Information Flow Tracking Architecture to Secure a RISC-V Core for IoT Applications". In: *2018 IEEE High Performance extreme Computing Conference (HPEC)*. 2018, pp. 1–7. DOI: 10.1109/HPEC.2018.8547578.

[6] Joël Porquet and Simha Sethumadhavan. "WHISK: An uncore architecture for Dynamic Information Flow Tracking in heterogeneous embedded SoCs". In: *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 2013, pp. 1–9. DOI: 10.1109/CODES-ISSS.2013.6658991.

[7] Luca Piccolboni, Giuseppe Di Guglielmo, and Luca P. Carloni. "PAGURUS: Low-Overhead Dynamic Information Flow Tracking on Loosely Coupled Accelerators". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018), pp. 2685–2696. DOI: 10.1109/TCAD.2018.2857321.

[8] OpenHW Group. *CORE-V CV32E40P: 4-stage 32-bit RISC-V core*. Version cv32e40p_v1.8.3, accessed 2026-04-15. 2024. URL: https://github.com/openhwgroup/cv32e40p.