# Towards Full Time Protection of an Open-Source, Out-of-Order RISC-V Core

Nils Wistoff[1], Gernot Heiser[2] and Luca Benini[1,3*]

[1]Integrated Systems Laboratory, ETH Zürich
[2]UNSW Sydney
[3]Department of Electrical, Electronic and Information Engineering, University of Bologna

**Abstract**

*Microarchitectural timing channels enable information transfer between security domains that are supposed to be isolated, bypassing the operating system's security boundaries. They result from shared microarchitectural state that depends on execution in one security domain and impacts timing in another. Since modern ISAs do not specify timing behaviour, they are insufficient to address these channels. The temporal fence instruction was recently proposed as a RISC-V extension that clears the processor's microarchitectural state and thus removes any timing dependence on execution history. It has been demonstrated to be extremely effective at low hardware overhead for in-order RV processors, such as CVA6 [1]. In this work-in-progress, we provide initial insight into the effectiveness and cost of the temporal fence on the open-source, 12-stage, out-of-order RV64GC core OpenC910. We highlight challenges that arise from the out-of-order microarchitecture of OpenC910 and propose an approach that leverages the custom Xthead extension of OpenC910 and minimises the required hardware modifications to enable time protection.*

## Introduction

Security schemes for application-class processors largely rely on memory protection, restricting the memory view of applications using abstractions such as address space virtualisation. However, as Ge et al. [2] argue, and the Spectre attacks have prominently demonstrated [3], this abstraction is insufficient for a principled isolation of applications. There exist timing dependencies between concurrently running applications through shared microarchitectural state, where one application's execution may impact another application's execution time. This timing dependency can be leveraged to bypass the system's security boundaries and covertly transfer information. We refer to such a communication channel as a *timing channel*.

In [4], the authors complement the existing notion of memory protection with *time protection*, a concept assumed at preventing timing channels. As they show, current instruction set architectures (ISAs) do not provide the means to enforce time protection. Hence, they propose to extend the existing ISA abstractions by adding control over shared microarchitectural state.

To address this need, Wistoff et al. [1] propose the temporal fence instruction (`fence.t`), which clears shared microarchitectural components that may constitute a timing channel. Furthermore, they propose a systematic approach for implementing `fence.t` by conservatively clearing all non-architectural state. They demonstrate this approach on CVA6, an in-order, 6-stage RV64GC core [5].

*Corresponding author: `Nils Wistoff`

In this work-in-progress, we prototype the `fence.t` instruction in OpenC910, an out-of-order (OoO) RISC-V core [6]. As we show, this core's high complexity and OoO architecture introduce new challenges that need to be addressed. We propose an approach that combines existing instructions in OpenC910 with an aggressive reset of microarchitectural state to enable time protection with minimal hardware modifications.

## fence.t in Large OoO Cores

### OpenC910

OpenC910 is an industrial, 64-bit, application-class, 12-stage, OoO RISC-V core by T-Head Semiconductor Co., Ltd., implementing the RV64GCXtheadc ISA. It was open-sourced in 2021 under the Apache License and features the custom Xtheadc extension [6].

In the following, we will introduce those elements of the extension that we leverage in this work.

**sync.i Instruction** The Instruction Stream Synchronisation (`sync.i`) instruction serves as a barrier in the instruction stream. It ensures that the instructions preceding it retire before those that follow it.

**dcache.call Instruction** The Data Cache Clear All (`dcache.call`) instruction clears the L1 data cache, writing back all dirty cache lines.

**mrvbr CSR** The Machine Mode Reset Vector Base Address Register (`mrvbr`) holds the address from which the core starts execution after coming out of reset.

## The Problem of Mixed State

OoO execution can introduce new challenges for temporal partitioning of shared microarchitectural state. On a simple in-order core, it is trivial to differentiate between architectural components that must be preserved on a temporal fence (the register files) and the non-architectural state that needs to be cleared (everything else). However, advanced microarchitectural optimisations, such as register renaming, obscure the boundary between architectural and non-architectural state. While the logical registers define the processor's architectural state, their allocation to physical registers constitutes non-architectural state with possible timing implications, harbouring the potential for timing channels. This raises the question of how to preserve the logical registers while bringing the re-namer and physical register file to a deterministic, history-independent state.

In the following, we propose an approach that is minimally intrusive to the hardware: software saves the architectural registers on the stack before `fence.t` conservatively resets the entire register file. Immediately afterwards, software restores the registers by loading them from the stack, resulting in a deterministic physical register layout.

## Hardware Modifications

The key hardware modification is the addition of the `fence.t` instruction, which, on this system, triggers a reset of the entire core except for the control and status registers (CSRs).

## Invocation of fence.t

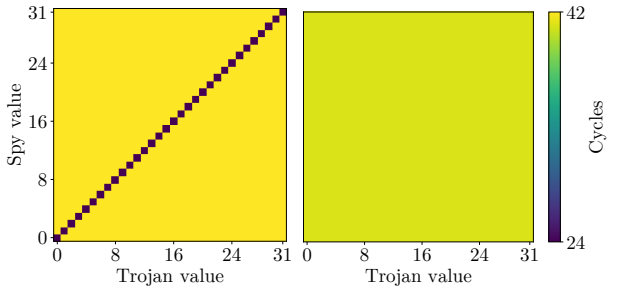
We aim to minimise the required hardware modifications for `fence.t` and reuse existing instructions as much as possible. As a result, a sequence of steps before and after `fence.t` is necessary to preserve computational and architectural correctness.

**Step 1: Save context.** Save the stack pointer (`sp`) at a known location that is not affected by `fence.t`, e.g. the `mscratch` CSR. Write the architectural registers onto the stack.

**Step 2: Define reset vector.** Write the address of the instruction following `fence.t` into the `mrvbr` CSR. Execution after `fence.t` will resume from here.

**Step 3: Clear L1 data cache.** Execute `dcache.call` to write back dirty cache lines.

**Step 4: Execute fence.t.** This resets the entire core except for the CSR files. We guard the `fence.t` instruction by `sync.i` instructions to ensure that all previous steps have been completed.




*a: Unmitigated.* *b: Using `fence.t`.*

**Figure 1:** *The L1 data cache channel on OpenC910.*

**Step 5: Restore context.** Restore the stack pointer and load the architectural registers from the stack.

# Preliminary Results

For initial validation of the proposed mechanism, we port the bare-metal timesec-bench [7] to OpenC910 and run it in RTL simulation. The resulting channel matrix in Figure 1 shows the time for the spy to access different cache lines given a particular value signalled by the Trojan, where any variation along a horizontal cut indicates a channel. Such a channel is clearly present in the unmitigated case, as the diagonal line in Figure 1a evinces. Conversely, invoking `fence.t` when switching between the Trojan and the spy removes all correlations, as shown in Figure 1b.

In future work, we will evaluate this approach's security and performance implications in the context of an operating system.

# References


[1] Nils Wistoff et al. "Systematic Prevention of On-Core Timing Channels by Full Temporal Partitioning". In: *IEEE Trans. Comput.* 72.5 (2023), pp. 1420–1430. DOI: 10.1109/TC.2022.3212636.

[2] Qian Ge, Yuval Yarom, and Gernot Heiser. "No Security Without Time Protection: We Need a New Hardware-Software Contract". In: *APSys'18*. ACM, 2018, 1:1–1:9. ISBN: 978-1-4503-6006-7. DOI: 10.1145/3265723.3265724.

[3] Paul Kocher et al. "Spectre Attacks: Exploiting Speculative Execution". In: *S&P'19*. May 2019, pp. 1–19. DOI: 10.1109/SP.2019.00002.

[4] Qian Ge et al. "Time Protection: The Missing OS Abstraction". In: *EuroSys'19*. ACM, 2019, 1:1–1:17. ISBN: 978-1-4503-6281-8. DOI: 10.1145/3302424.3303976.

[5] F. Zaruba and L. Benini. "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology". In: *IEEE Trans. on VLSI Systems* 27.11 (2019), pp. 2629–2640. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2019.2926114.

[6] T-Head Semiconductor Co., Ltd. *OpenC910 Core*. 2021. URL: https://github.com/T-head-Semi/openc910.

[7] Mathieu Escouteloup et al. "Under the dome: preventing hardware timing information leakage". In: *CARDIS'21*. Nov. 2021, pp. 1–20. URL: https://hal.archives-ouvertes.fr/hal-03351957.