MINOTAuR: A Timing-Predictable Open Source RISC-V Core Featuring Speculative Execution

Alban Gruin Thomas Carle Hugues Cassé Christine Rochange Pascal Sainrat

IRIT — Université Toulouse 3 — CNRS

Motivation

In real-time systems, precise Worst-Case Execution Time (WCET) analysis is required to schedule tasks in such a way that they meet their deadlines. In multicore processors, interference between cores (e.g. on a shared memory bus) impacts the WCET, and must be taken into account.
The usual solution is the compositional approach: latencies to shared resources are analysed separately, then combined to the WCET of tasks. However, so-called "timing anomalies" make this approach unsafe. Timing anomalies are situations in which a worst-case local situation (e.g. a cache miss) does not result in the worst global situation (i.e. the WCET).

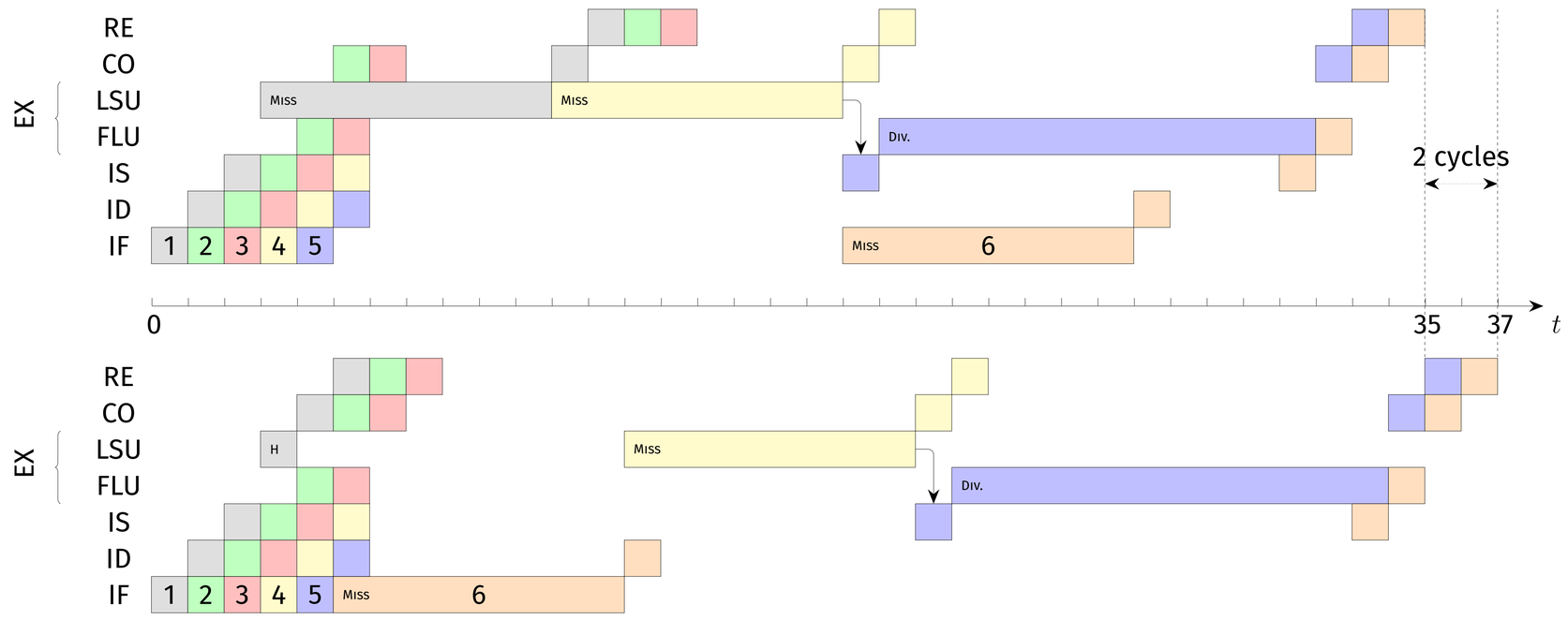
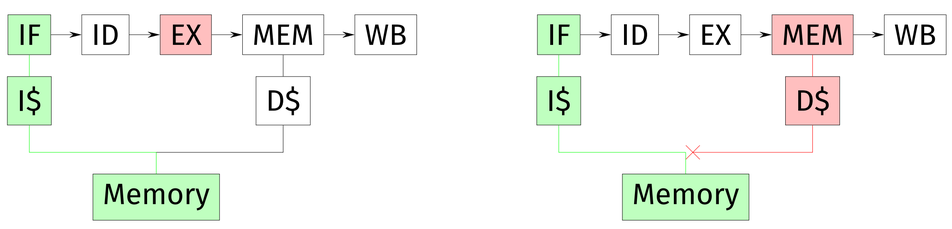


Figure 1. Example of a timing anomaly on the Ariane processor.

In this work, we modified Ariane, an open-source RISC-V processor, to improve its predictability, without sacrificing its performance too much. We called the result MINOTAuR (Mostly IN-Order Timing predictAble processoR) [1].

Monotonicity

To allow compositionality, one needs to guarantee that the processor is free from timing anomalies. One solution is to design a "monotonous" pipeline, in which an instruction cannot prevent an "older" instruction (more advanced in the pipeline) from gaining access to a shared resource.



(a) Pipeline state at cycle x.

(b) Pipeline state at cycle $x+1$.

Figure 2. Example of monotonicity violation: the instruction in MEM misses in its cache and has to wait that a miss in the IF stage is resolved.

Issues with the Ariane processor

The Ariane processor is an open-source RISC-V core. It features parallel functional units, a scoreboard and a dynamic branch prediction system.

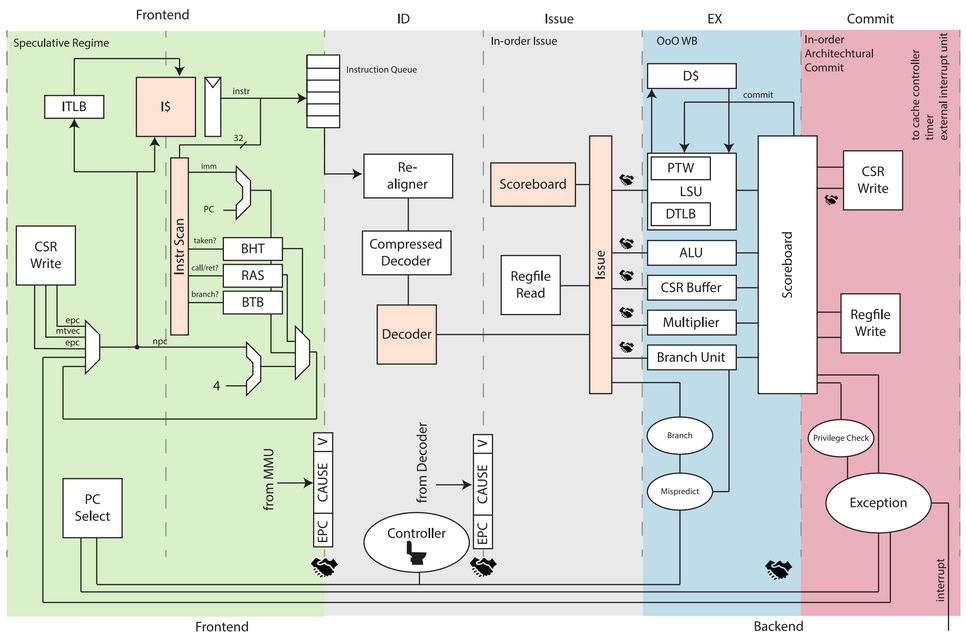


Figure 3. Overview of the Ariane processor. The parts highlighted in orange have been modified in MINOTAuR.

To make MINOTAuR monotonic, we prevented the instruction cache from making requests when a memory instruction is in the pipeline (from the fetch stage to the load/store unit), and when the processor is speculating to avoid changing the state of the cache, in case of a misprediction.

How RISC-V helps

Thanks to the open, royalty-free nature of RISC-V, it is easy to find high-quality, free and open-source processors design, like the Ariane. This allows us to focus on our research (no need to design bespoke processors), appropriate existing components (software, toolchains, operating systems), and make our changes open, so that the community can immediately benefit from them.

Formal model

Is an instruction ready?

$$
\begin{aligned}
&c.ready(i) := \\
&\quad (c.stg(i) \neq pre \wedge \neg c.pending(i, branch) \wedge pwrong(i)) \\
&\quad \vee (c.cnt(i) = 0 \wedge c.isnext(c.stg(i), i)) \\
&\wedge (c.stg(i) = \text{PC} \Rightarrow (ichit(i) \\
&\qquad \vee (\neg c.pending(i, branch) \wedge \neg c.pending(i, load) \wedge \neg c.pending(i, store) \\
&\qquad\quad \wedge \neg c.pending(i, atomic)))) \\
&\wedge (c.stg(i) = \text{IS} \Rightarrow \\
&\qquad ((opc(i) \in \{load, store, atomic\} \Rightarrow c.slot2(\text{LSU})) \\
&\qquad \wedge (opc(i) \notin \{load, store, atomic\} \Rightarrow \neg c.pending(i, csr)) \\
&\qquad \wedge (opc(i) \in \{mul, div\} \Rightarrow \neg c.pending(i, div)) \\
&\qquad \wedge (\forall j < i.dep(i, j), c.stg(j) \sqsupseteq_{\mathcal{S}} \text{CO}))) \\
&\wedge (c.stg(i) = \text{LSU} \Rightarrow (opc(i) \in \{store, atomic\} \wedge \neg c.pending(i, atomic)) \\
&\qquad\quad \vee (opc(i) = load \wedge (\neg c.pending(i, store) \wedge \neg c.pending(i, atomic))))
\end{aligned}
$$

Is a stage free?

$$
\begin{aligned}
&c.free(s) := \\
&\quad s \in \{\text{ALU}, \text{MUL}_1, \text{CSR}, \text{MUL}_2, \text{CO}, post\} \\
&\quad \vee (s \in \{\text{IF}, \text{IS}, \text{LSU}, \text{SU}\} \wedge c.slot(s)) \\
&\quad \vee (s \in \{\text{PC}, \text{ID}, \text{DIV}, \text{LU}, \text{ST}\} \wedge ((\neg \exists j \,.\, c.stg(j) = s) \\
&\qquad \vee (\exists j \,.\, c.stg(j) = s \wedge c.ready(j) \wedge c.free(c.nstg(j))))) \\
&\quad \vee (\exists i . c.stg(i) = s \wedge pwrong(i) \wedge \neg c.pending(i, branch))
\end{aligned}
$$

Results

We synthesized MINOTAuR on a Xilinx Zynq Z7-20 FPGA using Vivado, and compared its performance with Ariane on the TACLe benchmark suite and on CoreMark.

	Ariane	MINOTAuR
CoreMark score	110.77	105.72
Total cycles (TACLe)	599,720,712	604,672,290
Average		5.67 %
Weighted mean		0.83 %

Table 1. Performance comparison of Ariane and MINOTAuR.

We succesfully modified an existing processor to make it free from timing anomalies, while reconciling performance and timing-predictability: while the MINOTAuR core is slower on memory-intensive programs, its performance remains comparable to Ariane in other benchmarks.

References

Alban Gruin, Thomas Carle, Hugues Cassé, and Christine Rochange. Speculative execution and timing predictability in an open source RISC-V core. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 393–404. IEEE, 2021.

Université Toulouse III Paul Sabatier — Université Fédérale Toulouse Midi-Pyrénées

IRIT — Institut de Recherche en Informatique de Toulouse — CNRS - INP - UT3 - UT1 - UT2J