Hardware/Software Co-Analysis for Worst Case Execution Time Bounds

Can Joshua Lehmann*, Lars Bauer, Hassan Nassar*, Heba Khdr*, Jörg Henkel*

* *Karlsruhe Institute of Technology (KIT), Chair for Embedded Systems (CES)*
can.l@posteo.de, lars.bauer42@gmx.de, {nassar, heba.khdr, henkel}@kit.edu

***Abstract*—Ensuring that safety-critical systems meet timing constraints is crucial to avoid disastrous failures. To verify that timing requirements are met, a worst-case execution time (WCET) bound is computed. However, traditional WCET tools require a predefined timing model for each target processor, which is not available when using custom instruction set extensions. We introduce a novel approach based on hardware-software co-analysis that employs an instrumented hardware description of the target processor, removing the requirement for a separate timing model. We demonstrate this approach by extending the FemtoRV32 Individua RISC-V processor with a custom instruction set extension and show that it accurately models the timing behavior of the resulting system.**

***Index Terms*—WCET, ISA, Hardware/Software Co-Design**

I. Introduction

Safety-Critical applications in industries such as industrial control, automotive or aerospace often have to operate within predefined timing requirements [1]. Analyzing the Worst-Case Execution Time (WCET) of applications is therefore pivotal for verifying these systems' timing constraints and ensuring their reliability. Finding WCET bounds is difficult due to the complexity of modern hardware features. Traditional WCET tools such as AbsInt's aiT [2] and OTAWA [3] are able to automatically find upper bounds on the WCET for a given application [4]. To accomplish this, they analyze the application binary using a timing model of the target processor.

In recent years, we have seen the wide spread adoption of open and extensible ISAs such as RISC-V. Vendors often use custom instruction set extensions to implement application specific instructions. However, performing timing analysis in such dynamic environments with traditional WCET tools is difficult, since they require a timing model of the target processor. Usually this timing model is created at tool construction time [4]. Since custom instruction set extensions change the timing behavior of the target processor, the timing model needs to be modified to reflect the timing behavior of custom instructions. This is both time intensive and error prone.

We propose a novel method that finds WCET bounds by using HW/SW co-analysis [5], [6] without a separate timing model. Instead, timing behavior is derived directly from an instrumented hardware description of a processor. This enables the use of our method in hardware/software co-design environments where a processor is extended with custom instructions. Since the timing behavior is derived directly from the hardware description, we eliminate the timing model as a source of errors.

II. Background

In traditional WCET tools [7], [4], [2], [3], users provide an application binary and select the target processor's timing model. If the tool does not reuse control flow information from the compiler [7], a control flow graph has to be reconstructed from the application binary. The timing model of the target processor is then used to create an abstract execution graph. Compared to the control flow graph, it includes the microarchitectural states of the processor and can thus be used to model the timing behavior of the processor. Some tools only implicitly use the abstract execution graph to compute basic block timings. Loop bounds, derived from the control flow graph or manually set by the user, are encoded in an ILP with IPET [8]. An ILP solver then determines the WCET bound.

Hardware-software co-analysis analyzes a processor and an application binary as a single system [5]. Previous work has implemented a co-analysis tool for Verilog by extending a Verilog simulator [6]. The given processor and a memory containing the application binary are instantiated into a combined top-level module. First, a single simulation is started beginning at the reset state of the processor. When an unknown value (`X`) reaches the program counter (PC), a branch has been encountered and two simulations for each option are spawned. Simulation states with equivalent PC value are merged. When two bits with different values are merged, they are replaced with an unknown value (`X`).

III. Computing the WCET using Co-Analysis

By using hardware-software co-analysis [5], [6], our method directly obtains the abstract execution graph without the need for control flow reconstruction or a timing model. This enables it to automatically handle custom instruction set extensions. First, the instrumented processor is instantiated into a top-level design. The top-level design connects the processor to a memory containing the application binary. We then use hardware-software co-analysis to analyze the combined top-level design as a single system. The analysis reconstructs a state machine from the top-level design. Since it is constructed based on the processor's hardware description, it already contains microarchitectural states and is thus analogous to the abstract

978-3-9826741-0-0/DATE25/© 2025 EDAA

execution graph in traditional WCET tools. Its large scale structure is similar to that of the control flow graph. We use the implicit path enumeration technique [8] to encode the WCET problem in the reconstructed state machine as an integer linear program (ILP). Loop bounds are added as additional flow-constraints. Finally, we use an ILP solver to find the WCET bound by maximizing the number of cycles before termination.

IV. Evaluation

We evaluate the use of hardware-software co-analysis for WCET analysis using the hardware description of the FemtoRV32 Individua processor [9]. This core is part of the FemtoRV32 family, a family of lightweight multi-cycle RISC-V processors designed to run on FPGA platforms. The Individua variant supports a 32-bit data path and implements the RV32IMAC instruction set. We use benchmark applications from the Mälardalen benchmark set [10], as it is one of the most widely used benchmark sets in WCET analysis. We use the clang compiler to compile the benchmarks into standalone applications and optimize them for size.

To demonstrate that our method can handle custom instructions, we extend the FemtoRV32 Individua processor [9] with the `cv.mac` integer multiply accumulate instruction as specified in the CORE-V extension [11]. We modify a subset of functions from the Mälardalen benchmark set [10] to use the custom instruction. We choose benchmarks that perform multiply accumulate operations and which can thus benefit from a fused multiply accumulate instruction. The WCET bounds of the modified applications are compared to the unmodified versions using our method. We manually add loop bound flow-constraints based on observed loop bounds during program execution. Since the FemtoRV32 Individua processor's instruction bus only supports 4 byte aligned loads, the use of compressed instructions can lead to differing timing behavior if an instruction is not aligned. To ensure comparable execution time between different versions of the functions, the benchmarks are compiled without compressed instructions. Additionally to computing the WCET bounds, we also measure actual execution times of the benchmarks. For this purpose we simulate the top-level design also used for hardware-software co-analysis.

Figure 1 compares the WCET bounds of the original benchmarks to the modified versions that use the custom multiply accumulate instruction. Our method is able to automatically model the timing behavior of the custom instruction and provide a reduced WCET bound. Comparing the computed WCET bounds to the maximum observed execution times shows that our method is able to accurately model the timing behavior of the custom instruction.

V. Conclusion

We demonstrate the feasibility of using hardware-software co-analysis finding WCET bounds. Hardware-software co-analysis enables WCET analysis in the presence of custom instruction set extensions. We demonstrate this by extending the FemtoRV32 Individua processor with the `cv.mac` multiply accumulate instruction. We show that hardware-software co-analysis can accurately model the timing behavior of this custom instruction on applications from the Mälardalen benchmark suite.

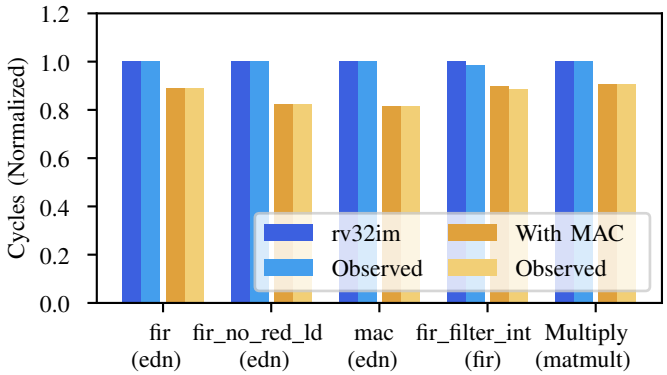

Fig. 1. Comparison of WCET bounds for each benchmark with and without the custom `cv.mac` instruction. Each WCET bound (left bar) is shown next to the maximum observed execution time of the function (right bar). The changes are displayed separately for each function where `cv.mac` was used. The number of cycles is normalized with respect to the WCET bound of the original benchmark.

Acknowledgment

This work was partially funded by the Federal Ministry of Education and Research, BMBF, as part of the MANNHEIM-CeCaS project (grant: 16ME0817) and by the German Research Foundation (DFG) as part of SPP 2377 project ARTS-NVM (grant: 502308721).

References

[1] L. Bauer *et al.*, "Analyses and architectures for mixed-critical systems: industry trends and research perspective", in *International Conference on Embedded Software Companion (EMSOFT)*, 2019.

[2] C. Ferdinand, "Worst case execution time prediction by static program analysis", in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2004, pp. 125–.

[3] C. Ballabriga *et al.*, "Otawa: an open toolbox for adaptive wcet analysis", in *International Conference on Software Technologies for Embedded and Ubiquitous Systems (SEUS)*, 2010, p. 35–46.

[4] R. Wilhelm *et al.*, "The worst-case execution-time problem—overview of methods and survey of tools", *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, 5 2008.

[5] H. Cherupalli *et al.*, "Bespoke processors for applications with ultra-low area and power constraints", in *International Symposium on Computer Architecture (ISCA)*, 2017, pp. 41–54.

[6] S. Sethumurugan *et al.*, "A scalable symbolic simulation tool for low power embedded systems", in *Design Automation Conference (DAC)*, 2022, p. 175–180.

[7] S. Hahn *et al.*, "LLVMTA: An LLVM-Based WCET Analysis Tool", in *International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2022, pp. 2:1–2:17.

[8] Y.-T. S. Li *et al.*, "Performance analysis of embedded software using implicit path enumeration", *SIGPLAN Not.*, vol. 30, no. 11, p. 88–98, 11 1995.

[9] B. Levy et al., "Femtorv32". [Online]. Available: https://github.com/BrunoLevy/learn-fpga/tree/master/FemtoRV

[10] J. Gustafsson *et al.*, "The Mälardalen WCET Benchmarks: Past, Present And Future", in *Workshop on Worst-Case Execution Time Analysis (WCET)*, 2010, pp. 136–146.

[11] "Core-v instruction set custom extension (v 1.3.1)", OpenHW Group. [Online]. Available: https://github.com/openhwgroup/cv32e40p/blob/master/docs/source/instruction_set_extensions.rst