CPU Execution Time Analysis based on RISC-V ISA Simulators: A Survey

Nicolai Iuga[1], Ionel Zagan[1,2], Vasile Gheorghita Gaitan[1,2]
[1] Faculty of Electrical Engineering and Computer Science, Stefan cel Mare University, Suceava, Romania
[2] Integrated Center for Research, Development and Innovation in Advanced Materials, Nanotechnologies, and Distributed Systems for Fabrication and Control (MANSiD), Stefan cel Mare University, Suceava, Romania
nicolaiiuga@gmail.com

***Abstract*—Multiple components affect the behavior of CPU execution tasks, mainly mutual exclusion during access to the shared resources and synchronous transmissions between tasks that require specific priorities. Referring to real-time task scheduling, the most important issues addressed are monitoring the deadline of the tasks, the probability of non-compliance with the deadline and ensuring that they are completed within the deadline imposed by the controlled system. This leads to the fact that a tasks set is feasible if the system has adequate resources to execute the tasks without losing any deadlines. To approach this challenge, this article addresses the feasibility analysis of runtime for real-time tasks to be performed in a virtual space before applying to a real embedded device. These processes take place offline, even before a system executes a set of tasks. In order to perform the feasibility analysis of the system, efforts are being made to provide virtual environments and simulators, such as the RISC-V ISA simulator and the WCET (Worst Case Execution Time) measurement, subsequently. These simulators have application-specific requirements, with their own advantages.**

Keywords— Computer architecture; Pipeline processing; Performance analysis; Scheduling.

I. INTRODUCTION

The proposal of different ISAs (Instruction Set Architectures), CPU hardware implementations and the use of central processing units in systems embedded in everyday life is regularly growing. An obvious example is robots, distributed data acquisition systems, and modern machines. The wiring is replaced with bus-type systems, smart switches take the place of classic switches and motor controllers integrate powerful processors. This allows for easy integration of additional sensors and more efficient error analysis in the event of a fault.

Very often, time-constrained control applications are completed by creating substantial pieces of code in assembly language [1], setting task priorities, using inter-task communication mechanisms, and using interrupts, respectively. The main repercussion of this concept is that management software written by empirical procedures can be deeply uncertain. If it cannot verify all crucial time constraints a priori and the operating system does not incorporate explicit procedures for carrying out tasks in real time, the system may perform fine for a time, but may not meet deadlines established in individual unique but cases. The repercussions of collapse can occasionally be disastrous and can harm humans or lead to significant destruction to the environment.

Running real-time embedded systems in the design of security systems comes with quite a challenge in terms of CPU runtime analysis. Recent researches has led to the improvement of static program evaluation mechanisms that definitely establish the upper limits for the WCET of code fragments expressed as procedures in execution [2]. We name systems in real-time, when their correct behavior revolves around not particularly on the logically correct output but further on the point when these tasks are performed [3]. A real-time process must perform within strict time limits, applied to the actions in its environment and the process it leads. This means that the correct performance of a system in real-time revolves around not simply on the result of the prediction but likewise on when the result is generated. Therefore, knowledge of the execution attributes of a program is crucial to the outstanding design and development of a real-time system [1]. The problem of calculating the stringent limits of the execution of a program is a dynamic field of analysis, with software and tools that use both static methods and dynamic evaluation. The evaluation of static techniques calculates the upper limits of the execution time from an analytical model of the target architecture. Dynamic evaluation programs determine the execution time of assessments executed on real hardware. Hybrid techniques merge runtime data obtained from assessments with static analytical data, such as control flow charts, to increase the security and accuracy of results. Probabilistic techniques seek to determine statistical models from assessments to calculate the upper limits of the execution time [4].

We structured this article as follows. After a concise introduction in chapter I, chapter II presents the current state of research, considering the methods of estimating WCET and the importance of this analysis in the design of critical systems. Chapter III describes the analysis of the most unfavorable execution time based on the RISC-V ISA simulators, and the last chapter presents the conclusions and future research directions.

978-1-6654-8162-5/22/$31.00 ©2022 IEEE

II. RELATED WORK

CPU execution time and WCET are the most important aspects of real-time critical systems. An executable program exhibits a specific fluctuation regarding the execution times, determined by internal events and environmental interference. Of all the execution times of a program, the exact maximum is the highest execution time a program can ever reach [5]. This time is named WCET. There are two classes of WCET estimation methods, namely static and those based on practical measurements. A secure mechanism for timing evaluation is a static analysis by an abstract interpretation that produces proved upper limits for WCET tasks. WCET static analyzers are feasible for complex processors and support both single-core and multi-core processors. Thus, suitable models of processor / System-on-Chip (SoC) architecture can be established. However, there are current high-performance SoCs that have uncertain or undocumented items that affect synchronization performance. WCET hybrid analysis merges static value and path evaluation with assessments to capture task synchronization behavior. Compared to end-to-end measurements, the improvement of hybrid techniques consists of brief code fragments can be selected, which then cover the full program under evaluation. Based on these assessments, the most unfavorable path can be determined [4]. The most representative tools for determining WCET are: pWCET (partially WCET) [6], Heptane [7], OTAWA [8], W-SEPT [9], SWEET [10], Bound-T [11] and T- CREST (Time-predictable multi-core architecture for embedded systems) [12].

Wilhelm Reinhard et al. [13] summarized existing WCET estimation methods and tools. Static techniques are not based on actual hardware execution. They consider the code itself, merge the control flow chart with a model of the hardware architecture, and provide an upper bound for this mix. Measurement-based techniques implement code on physical hardware or a simulator for specific inputs. Next, based on the measured times, the margins execution times are determined. Therefore, static techniques are highly secure and guarantee that the execution time will not be higher than the prediction obtained.

In 2015, Jacobs et al. [6] examined an individual case of pWCET. Calculating pWCET is useful in many contexts, such as debugging performance and energy efficiency. Detection of code fragments with large WCET is highly problematic, as the worst case may appear in unusual circumstances and may not be simple to unmask, so common profiling styles do not handle. pWCET is helpful for developers to determine possible bottlenecks (at worst) in their applications and to concentrate their efforts on proper code snippets. pWCETs are useful for splitting a task into smaller chunks that do not overlap. Runtime is also important for security: some software attacks involve inserting other code into a target application. A shelter may consist in calculating the WCET of a task and checking that the current execution time does not exceed the calculated value [14].

Damien Hardy et al. [7] introduced Heptane, a free software program that predicts upper execution time limits for MIPS (Microprocessor without Interlocked Pipeline Stages) and ARM v7 architectures, to analyze with late WCET evaluation routines. The proposed software architecture designed for Heptane is modular and extensible to simplify the assimilation of new methods. Heptane calculates WCETs using static binary code analysis. It has a static analysis of micro-architectural parts such as caches. Linked to alternative static open source WCET analysis tools, Heptane focuses especially on cache analysis (cache hierarchy analysis, support for various replacement schemes), but only supports two target processors.

Rabab Bouziane et al. [14] showed how Heptane, a static WCET estimating tool, can calculate pWCET based on ILP (Instruction Level parallelism). Unlike Heptane, OTAWA [8] supports multiple processor architectures but implements less advanced cache analysis. The OTAWA simulator was adapted by Maiza Claire et al. [9] for the W-SEPT project. The W-SEPT project was developed to examine and use the influence of program semantics on WCET evaluation. Semantic issues and the elimination of low-feasibility paths are the focus of the project. As much as available, the goal is to increase and readjust the classic WCET estimation flow.

SWEET (SWEdish Execution Time tool) [10] focuses on program flow study and does not cover any hardware analysis. The purpose of SWEET flow analysis is to automatically calculate instruction flow information. SWEET provides a strong loop-related analysis, as the upper limits of the number of loop iterations must be known to obtain WCET estimates. SWEET flow analysis can also determine more accurately lines that are executable depending on the structure of the control flow graph, but are not workable when examining the semantics of the program and the values of the inputs.

Bound-T [11] is a software tool that uses static analysis to calculate WCETs and the use of the embedded stack. This tool supports various processor architectures, but does not provide cache analysis. Another issue with this topic in Bound-T is memory access. If the analyzed program accesses a variable, sometimes as a whole (say a 32-bit word), and sometimes in chunks (such as four bytes), Bound-T analysis of the values of the variable and thus the entire control flow, may be wrong.

T-CREST (Time-predictable multi-core architecture for embedded systems) is a static WCET estimation tool applied to the study of Patmos architecture. The toolkit enhances, for example, a hardware model, processes and translates input format information for *AbsInt aiT* timer analysis annotations, which operate this data in enhancement to the ELF binary to calculate strict WCET limits [12].

III. WCET ANALYSIS BASED ON RISC-V ISA SIMULATORS

The performance of a real-time system depends on the fact that all scheduled tasks can be guaranteed to be completed before the deadlines. Best Case Execution Time (BCET) and WCET are used for real-time systems scheduling analysis. In recent years, there was considerable interest in using formal methods and, in particular, in the verification of automated models for calculating BCET and WCET, as they immediately provide precise answers to these questions immediately. There are presently several simulators developed for the RISC-V architecture, we listed these in Table I. Operating RISC-V (Reduced Instruction Set Computer) has led to significant

advances in the open-hardware community, with many new models of processors and accelerators. However, turning open hardware into tools for researching computer architecture can be a challenge. It hasn't been long since RISC-V released the initial stable announcement of the RISC-V vector development, and there are already several free and commercial products. Examples are Ara [15] from ETH Zurich and Xuantie-910 [16].

Incorporating open-source hardware and software components can be time-consuming and difficult to maintain, besides being difficult to simulate. In particular, when using open-hardware platforms, there is a simulation gap, with researchers addressing slow software simulators or fast FPGA (Field Programmable Gate Array) prototyping. There are commercial emulation tools, but they are far too expensive to use in the academic research activity [17]. Researchers in computer architecture still require a simulation framework that supports them to cooperate with their partners in industry and research. However, the licensing conditions of a simulator and the quality of the code may prevent this teamwork. Some open source software licenses may be extremely restrictive, principally in an industrial environment, as they request the release of any simulator improvements. In addition, weak code quality and lack of modularity may prevent new programmers from understanding and changing the code.

FireSim can simulate random hardware models created in Chisel (Constructing Hardware in a Scale Embedded Language) or projects that can be converted to FIRRTL (Flexible Internal Representation for RTL), including Verilog projects via the Yosys Verilog to the FIRRTL stream. FireSim can develop its own RTL (Register Transfer Level) and run at prototype FPGA speeds close to cloud FPGAs, while achieving cycle performance results. FireSim can integrate custom software templates for units that you do not want / need to create as RTL. FireSim was essentially designed to simulate data centers by fusing real-time open logic for RISC-V processors with a cycle-specific network simulation. FireSim gives all the RTLs and models needed for cycles, simulating precisely between one and thousands of multi-core computer nodes. It also offers a Linux distribution that is suitable with the RISC-V systems. One of the frequently operated computer system architecture analysis platforms that includes system-level architecture and processor micro-architecture is gem5 [18], which can analyze those new theories about vector architectures. The Gem5 simulator surmounts these limitations by bringing a malleable, interchangeable simulation system able of evaluating a wide area of systems and is accessible to all researchers [18].

TABLE I. SIMULATORS DEVELOPED FOR RISC-V ARCHITECTURE

Simulator Name	**FireSim [19]**	**gem5 [20]**	**Spike [21]**	**BRISC-V** Error! Reference source not found.	**WebRISC-V [23]**	**riscvOVPsim [24]**
Licence	BSD (Berkeley Software Distribution)	BSD-style	BSD 3-clause	Proprietary – open source	BSD 3-clause	Proprietary (core simulation platform), Apache License

For multimedia extensions, gem5 supports Intel MMX and SSE (64-bit and 128-bit extensions), which are achieved as part of the root micro-architecture. However, support for extra ongoing developments, such as AVX2 and AVX-512, is missing. In terms of vector architecture, there is complete support for ARM SVE. Gem5 is primarily created in C ++ and Python, and most segments are supplied under a BSD-style license. It can simulate a complete system with operating system devices or only in user spaces, where system functions are produced directly by the simulator in syscall emulation mode (SE mode). It can build flexibly a memory system from caches. Newly, the Ruby simulator has been incorporated with gem5 to bring a highly malleable memory system model [25].

Spike, the ISA RISC-V simulator, achieves a practical model of one or more RISC-V versions. Projects are primarily versioned to show when the Application Programming Interface (API) has been enlarged or become inappropriate. In this vision, Spike intends to track the SemVer (Semantic Versioning Specification) modeling scheme, in which later versions are incremented when adjusting compatible APIs. WebRISC-V is a web-based RISC-V simulation environment that aims to facilitate student learning and the teaching experience of instructors. One of the major improvements of WebRISC-V is its direct availability in the web browser based on its innovative implementation. WebRISC-V has the back end created in PHP and the front end in HTML and JavaScript. However, if the user prefers a local installation, it can be done on a Linux or Windows server. WebRISC-V integrates the five stages of the RISC-V architecture, incorporating the ability to investigate the behavior of hazard detection and redirection units [26]. This simulator incorporates the majority of the instructions of the basic 64-bit RISC-V ISA module and its extension. To avoid slowing down or blocking the server, the execution of each uploaded program is restricted to 1000 clock cycles, and the data memory is restricted to 5 KB. So, in state of possible programming faults, such as infinite loops, the execution can end in a limited time. When loading assembly instructions, the analyzer tests for unaccepted / wrong instructions or wrong labels. If there is a failure, the simulation pauses and the corresponding line number is exposed. Execution of the WebRISC-V assembly code can be done in two ways, executing all the code simultaneously or step by step. The pipeline registers for each stage have a certain color for Fetch, Decode, Execute, Memory, and Write-Back (Fig. 1). Simultaneous execution of all code lines is usually used to verify the rightness of the assembly code, but could also check the final status of registers and data memory or entire clock cycles. By performing in simple steps, the user can observe in the left panel the progress of the instructions in each step and can analyze the values of the registers and the contents of the data memory.

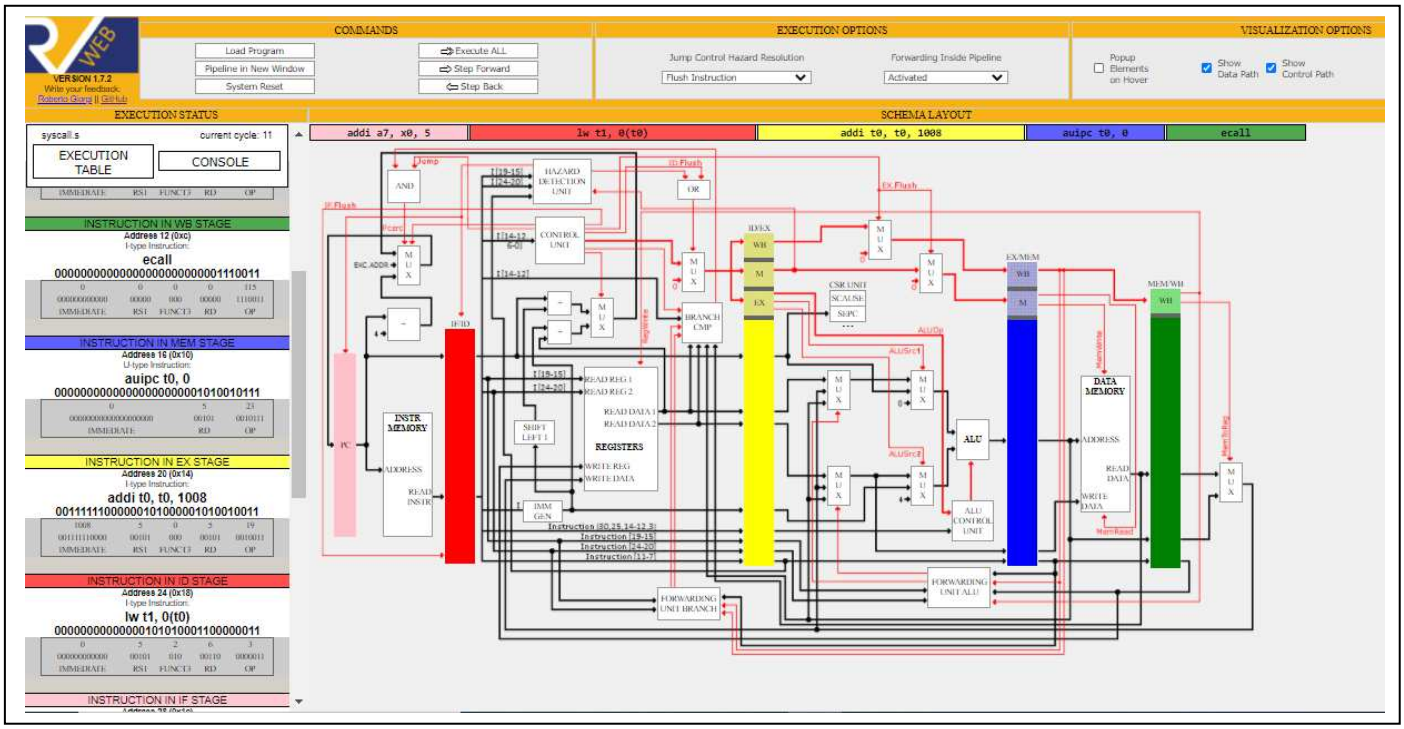

Fig. 1. WebRISC-V highlighting the steps of processing instructions.[27]

WebRISC-V clearly specified the binary code fields for each statement, along with the numeric value, the type of statement, and the value of each field (Fig. 2). By picking the hazard detection and redirection units, the user can examine the analogous input and output signals (Fig. 3) and thus can track any errors during execution.

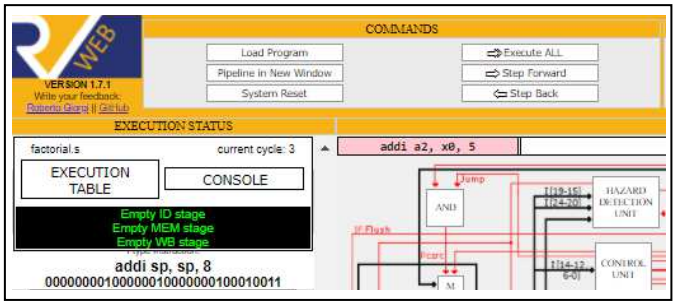

Fig. 2. WebRISC-V and the characteristics of binary code fields. [27]

The WebRISC-V simulator was developed based on the WebMIPS simulator [28] which came with several enhancements such as:

- A list of implemented instructions has been included to the “upload program” page, to provide users with a table with easy examples.
- The cycle number and program name are always visible to improve context awareness.
- The user can examine the values inside the threads by directly moving the cursor on a certain architectural component (a pop-up window can be opened for convenience).

In addition, an important revision to increase usage is the Step-Back feature, which grants the user to go back one step at a time during execution to accurately analyze changes and make it easier to understand. The user can go back and forth on each clock cycle to see the specific changes.

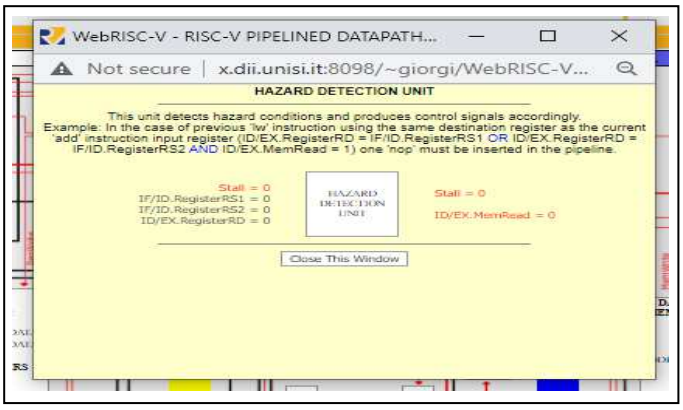

Fig. 3. WebRISC-V and the hazard detection unit. [27]

The BRISC-V(Boston RISC-V architecture design exploration suite) platform offers many opportunities for practical education in computer architecture [29]. The platform contains a RISC-V simulator used to test software independent of any hardware system, a suite of RISC-V tools for compiling and executing user code, a modular, parameterized, synthesizable multi-core RISC-V hardware system described in Verilog and a graphical user interface (GUI) used to generate and view hardware systems with one or more cores. BRISC-V

platform tools provide a wealth of RISC-V-based software and hardware understanding resources [31].

The riscvOVPsim simulator is simple to understand and efficient to operate. It is flexible, proper and quick, being developed by Imperas Software. There is no complex installation process or scripts for downloading and installing riscvOVPsim, thus just download and run the executable with the configuration options for RISC-V. riscvOVPsim is configurable to represent exactly the same deployment options that RISC-V processing operators choose, making it an excellent tool for developing RISC-V application software and verification and compliance testing. The simulator can join to GDB (GNU Project debugger) and Eclipse to debug source code and run in batch mode for regression analysis and use in repeated integration environments. It also has many tracking options to help you develop your program. riscvOVPsim has a built-in fixed platform comprising a processor instance of a variant of the RISC-V processor model and a memory subsystem.

Considering the nMPRA (multi pipeline register architecture, where *n* is the scale of multiplication) + nHSE (hardware scheduler engine for *n* threads) micro-architecture illustrated in Fig. 4, current research requires the development of a simulator for nMPRA architecture on RISC-V. For this, implementing the nMPRA specific resource remapping mechanism for RISC-V ISA is considered in the proposed simulator [32]. It is also important to highlight WCET on the proposed nMPRA + nHSE concept. The objectives of the innovative solutions of the high performance micro-architecture nMPRA and nHSE are represented by improving the time for changing the context of tasks and implementing specific RTOS (Real Time Operating System) functions in hardware, reducing kernel latency [29]. nMPRA is a characteristic architecture with various (*n*) pipeline registers. In this context, based on the multiplexing of multiplied pipeline resources for each processor instance (sCPUi), the nMPRA approach and the nHSE module produce an innovative solution with a kernel latency at one or two processor cycle events. These results represent a substantial improvement in RTOS software solutions or hybrid software / hardware implementations. Implementing RTOS in nMPRA, called HW_nMPRA_RTOS, also incorporates nHSE illustrated in [30]. HW_nMPRA_RTOS offers the essential functions of hardware-implemented RTOS with remarkable response times ranging from one machine cycle to less than three machine cycles (in occasional instances), depending on the case. Considering the RISC-V architecture, the CSR block is the design principle for HW_nMPRA_RTOS registers[33][34]. The objective of this project was to create, implement, experiment, and certify an ingenious approach called HW_nMPRA_RTOS. HW_nMPRA_RTOS is a merged acronym for nMPRA, nHSE and RTOS API, which operates and expands the notions of expanding data path capabilities. nHSE was designed as a scheduling module to enable the hardware deployment of an RTOS. The remarkableness of the HW_nMPRA_RTOS architecture is revealed in the name itself because the pipeline registers are expanded by sCPUi, which saves the hardware context of the thread.

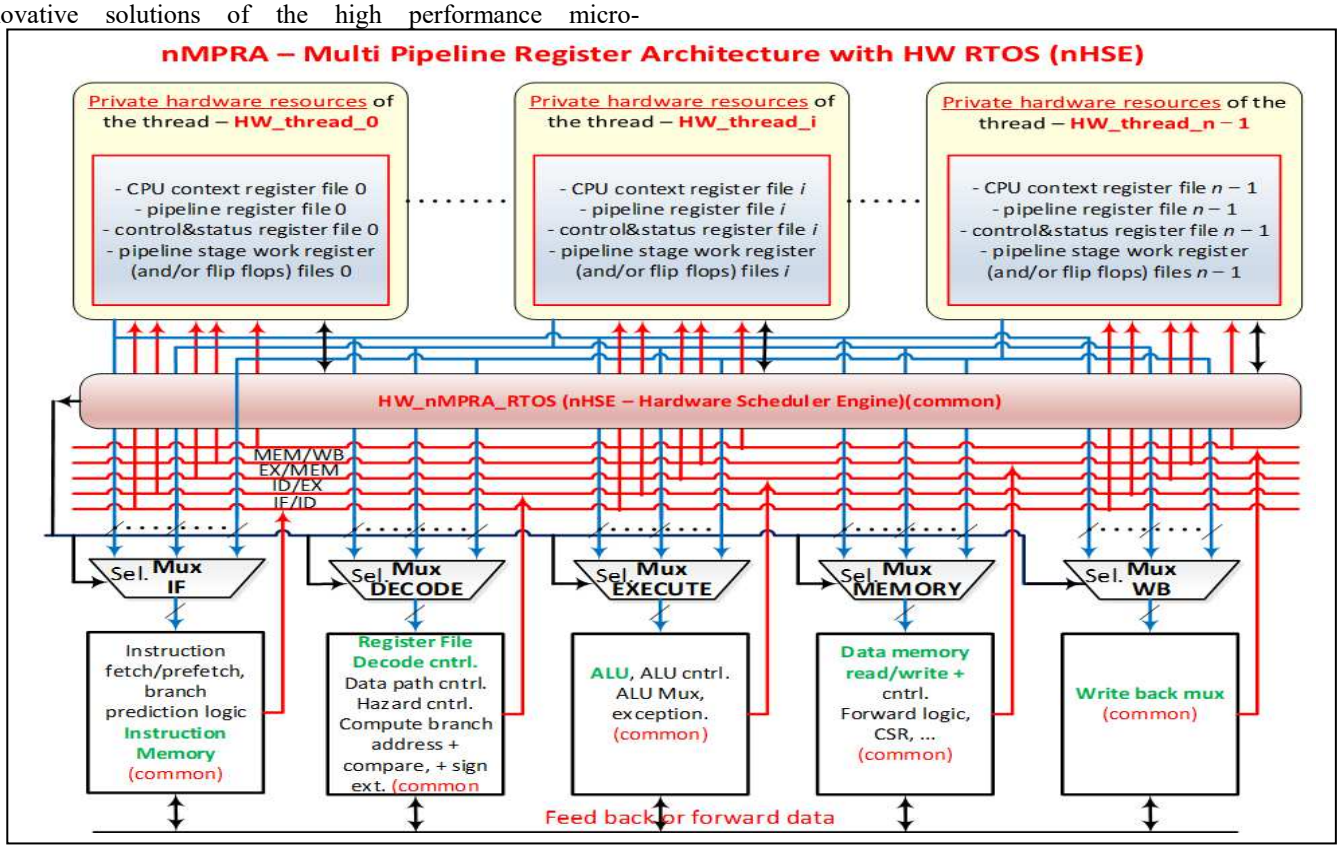

Fig. 4. Block diagram for nMPRA + nHSE microarchitecture [30]

This makes it easier to stop a pipeline instance and modify the context in an individual clock cycle. The multiplication of hardware resources and the registers of the nHSE scheduler are (partially) illustrated in Fig. 4.

The simulator for the designed architecture is based on predictive execution utilizing the nMPRA approach to accomplish the time constraints demanded in real-time applications without exceeding the deadlines demanded for sCPUi and the imposed power consumption limit. The simulation, synthesis and implementation of this project in an FPGA allows the development and debugging of applications. The research conducted for this paper was complemented by a set of practical tests specific to implementing HW_nMPRA_RTOS in FPGA [30], and the scientific results were certified based on well-chosen experiments.

IV. Conclusions and Future Work

All current security standards require reliable limits on the WCET of real-time tasks to be determined. The importance of synchronization analysis is proved by the recent funding of several projects, engaging leading industries in aerospace, rail and automotive, among others, with a focus on various RISC-V ISAs, multi-pipeline deployments with multiple issues and the resulting hardware increasingly complex. WCET estimates must be as strict as possible to optimize the capacity, power demands, and cost of the systems got.

From the analysis of the simulators available for RISC-V ISA, we conclude that we can use the open source resources available in WebRISC-V and BRISC-V projects, to develop a simulator that will be used to test nMPRA + nHSE micro-architecture as RISC-V ISA. The proposed simulator will be web-based so that it can be used independently of the user's operating system (similar to the implementation used by WebRISC-V and BRISC-V).

Future research considers simulator tests for the nMPRA + nHSE micro-architecture as RISC-V ISA for WCET estimation. This is required for verification and validation, essential steps in the design of critical systems. Critical safety systems must be guaranteed and the maximum execution time must be limited and established, so that response times can be ensured when critical cases involve a timely response. Utilizing a faster processor is not a solution to get a predictable response.

Acknowledgment

This work is supported by the project ANTREPRENORDOC, in the framework of Human Resources Development Operational Programme 2014-2020, financed from the European Social Fund under the contract number 36355/23.05.2019 HRD OP /380/6/13 – SMIS Code: 123847.

References

[1] BUTTAZZO, Giorgio C. Hard real-time computing systems: predictable scheduling algorithms and applications. Springer Science & Business Media, 2011.

[2] HECKMANN, Reinhold; FERDINAND, Christian. Worst-case execution time prediction by static program analysis. In: In 18th International Parallel and Distributed Processing Symposium (IPDPS 2004, pages 26–30. IEEE Computer Society. 2004.

[3] AHMAD, Shabir; MALIK, Sehrish; KIM, Do-Hyeun. Comparative analysis of simulation tools with visualization based on real-time task scheduling algorithms for IoT embedded applications. Int. J. Grid Distrib. Comput, 2018, 11.2: 1-10.

[4] KÄSTNER, Daniel, et al. Timeweaver: A¨ tool for hybrid worst-case execution time analysis. In: 19th International Workshop on Worst-Case Execution Time Analysis (WCET 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[5] FALK, Heiko; LOKUCIEJEWSKI, Paul. A compiler framework for the reduction of worst-case execution times. Real-Time Systems, 2010, 46.2: 251-300.

[6] JACOBS, Michael; HAHN, Sebastian; HACK, Sebastian. WCET analysis for multi-core processors with shared buses and event-driven bus arbitration. In: Proceedings of the 23rd International Conference on Real Time and Networks Systems. 2015. p. 193-202..

[7] HARDY, Damien; ROUXEL, Benjamin; PUAUT, Isabelle. The heptane static worst-case execution time estimation tool. In: 17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[8] http://www.otawa.fr/

[9] MAIZA, Claire, et al. The W-SEPT project: Towards semantic-aware WCET estimation. In: 17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[10] http://www.mrtc.mdh.se/projects/wcet/sweet/.

[11] http://www.bound-t.com/.

[12] SCHOEBERL, Martin, et al. T-CREST: Time-predictable multi-core architecture for embedded systems. Journal of Systems Architecture, 2015, 61.9: 449-471.

[13] WILHELM, Reinhard, et al. The worst-case execution-time problem—overview of methods and survey of tools. ACM Transactions on Embedded Computing Systems (TECS), 2008, 7.3: 1-53.

[14] BOUZIANE, Rabab; ROHOU, Erven; GAMATIÉ, Abdoulaye. Partial worst-case execution time analysis. In: ComPAS: Conférence en Parallélisme, Architecture et Système. 2018. p. 1-8.

[15] CAVALCANTE, Matheus, et al. Ara: A 1-GHz+ scalable and energy-efficient RISC-V vector processor with multiprecision floating-point support in 22-nm FD-SOI. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2019, 28.2: 530-543.

[16] CHEN, Chen, et al. Xuantie-910: A commercial multi-core 12-stage pipeline out-of-order 64-bit high performance RISC-V processor with vector extension: Industrial product. In: 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2020. p. 52-64.

[17] KARANDIKAR, Sagar, et al. Using FireSim to Enable Agile End-to-End RISC-V Computer Architecture Research. 2019.

[18] BINKERT, Nathan, et al. The gem5 simulator. ACM SIGARCH computer architecture news, 2011, 39.2: 1-7.

[19] https://fires.im/.

[20] https://gem5.googlesource.com/public/gem5/.

[21] https://github.com/riscv/riscv-isa-sim.

[22] AGRAWAL, Rashmi, et al. The BRISC-V platform: A practical teaching approach for computer architecture. In: Proceedings of the Workshop on Computer Architecture Education. 2019. p. 1-8.

[23] https://github.com/Mariotti94/WebRISC-V

[24] https://www.ovpworld.org/riscvOVPsimPlus/

[25] http://www.gem5.org/Introduction.

[26] GIORGI, Roberto; MARIOTTI, Gianfranco. Webrisc-v: A web-based education-oriented risc-v pipeline simulation environment. In: Proceedings of the Workshop on Computer Architecture Education. 2019. p. 1-6.

[27] https://webriscv.dii.unisi.it/

[28] BRANOVIC, Irina; GIORGI, Roberto; MARTINELLI, Enrico. WebMIPS: a new web-based MIPS simulation environment for computer architecture education. In: Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture. 2004. p. 19-es.

[29] GAITAN, Nicoleta Cristina; GAITAN, Vasile Gheorghita; MOISUC, Elena-Eugenia Ciobanu. Improving interrupt handling in the nMPRA. In: 2014 International Conference on Development and Application Systems (DAS). IEEE, 2014. p. 11-15.

[30] GĂITAN, Vasile Gheorghiță; ZAGAN, Ionel. An Overview of the nMPRA and nHSE Microarchitectures for Real-Time Applications. Sensors, 2021, 21.13: 4500.

[31] Sahan Bandara, Alan Ehret, Donato Kava, and Michel Kinsy. 2019. BRISC-V: An Open-Source Architecture Design Space Exploration Toolbox. In Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '19). ACM, New York, NY, USA, 306–306. https://doi.org/10.1145/3289602.3293991

[32] N. C. Gaitan and L. Andries, "Using Dual Priority scheduling to improve the resource utilization in the nMPRA microcontrollers," 2014 International Conference on Development and Application Systems (DAS), 2014, pp. 73-78, doi: 10.1109/DAAS.2014.6842431.

[33] GAITAN, Nicoleta Cristina. Enhanced interrupt response time in the nMPRA based on embedded real time microcontrollers. Advances in Electrical and Computer Engineering, 2017, 17.3: 77-84.

[34] GAITAN, N. C. Real-time Acquisition of the Distributed Data by using an Intelligent System. Elektronika ir Elektrotechnika, 2010, 104.8: 13-18.