RVEBS: Event-Based Sampling on RISC-V

Tiago Rocha, Nuno Neves, Nuno Roma, Pedro Tomás, Leonel Sousa
INESC-ID, Instituto Superior Técnico, University of Lisbon, Portugal
{tiagolopesrocha,nuno.neves,nuno.roma,pedro.tomas,las}@inesc-id.pt

***Abstract*—As RISC-V ISA continues to gain traction for both embedded and high-performance computing, the demand for advanced monitoring tools has become critical to fine-tuning the applications' performance. Current RISC-V hardware performance monitors already provide basic event counting but lack sophisticated features like event-based sampling, which are available in more established architectures such as x86 and ARM. This paper presents the first RISC-V Event-Based Sampling (RVEBS) system for comprehensive performance monitoring and application profiling. The proposed system builds upon existing RISC-V specifications, incorporating necessary modifications to enable the desired functionality. It also presents an OpenSBI extension to provide privileged software access to newly implemented control status registers that manage the sampling process. An implementation use case based on an OpenPiton processor featuring a CVA6 core on 28nm CMOS technology was presented. The results indicate that the proposed scheme is lightweight, highly accurate, and does not impact the processor's critical path while maintaining minimal impact on overall application performance.**

***Index Terms*—RISC-V, Event-Based Sampling, Hardware Performance Monitors, Performance Monitoring**

I. Introduction

Despite the high complexity of general-purpose processors, the high computational demands presented by many application domains often require the full exploitation of the architecture capabilities to satisfy the performance requisites. This often results in an added programming effort and a significant barrier to software developers, as it is generally not obvious what are the sources of performance degradation. To mitigate this burden, most Central Processing Units (CPUs) are now equipped with Performance Monitoring Units (PMUs) to support the profiling of software via the sampling of hardware events [1], such as the number of instructions retired, or cache misses.

As the RISC-V Instruction Set Architecture (ISA) continues to gain traction not only in the micro-controller and embedded market [2] but also in High-Performance Computing (HPC) applications [3], better and more complete support for software profiling tools and methods is needed. RISC-V ISA already provides the specification of a PMU to count hardware events through both fixed and programmable counters, and some efforts have already been made to port widely used software profiling tools to RISC-V machines [4]. However, the currently available methods and tools still lag behind the x86 and ARM CPU architectures that, on top of supporting direct counting of hardware events, also allow operating their Hardware Performance Monitors (HPMs) in sampling mode [5], [6], [7], where each counter collects information about the CPU state and can be parameterized with a specific sampling period.

To that extent, Event-Based Sampling (EBS) is a powerful profiling technique supported by modern processors through HPMs. It samples hardware events to identify a vast set of performance issues such as inter-thread coherence traffic [8], false sharing [9], long latency remote memory accesses in NUMA multicore systems [10], data locality problems [11], bandwidth consumption [12], and conflict cache misses [13]. This method also allows the analyses of instruction pointers and effective memory addresses, to pinpoint the exact lines of code (or the data objects) causing bottlenecks with minimal overhead compared to cycle-accurate hardware simulators.

To achieve such lower overhead, EBS triggers the data collection step only when certain events occur a predefined amount of times, reducing traffic and capturing a broader range of performance insights, particularly in complex and long-running applications. This approach is less intrusive and offers more granular and representative data without significantly impacting the system's performance.

Major industry players like Intel, AMD, IBM, and ARM already support EBS through specific implementations like Intel *PEBS* [5], AMD *IBS* [6], IBM *MRK* [14], and ARM *SPE* [7]. These implementations are supported on specialized hardware on the HPM to enable EBS. However, a detailed RISC-V implementation is still pending, with only brief mentions of its potential use [15]. As a consequence, RISC-V still cannot perform EBS, as no specification for the hardware nor Supervisor- and User-level software to support it exists.

In this paper, we propose a novel extension to the RISC-V HPM that introduces EBS capabilities, enabling fine-grained performance analysis and profiling. Its main contributions are:

1) **RISC-V Performance Counter Extension**: we propose a new set of Control Status Registers (CSRs) that provides support for EBS without requiring processor interruption or stalling. This allows the processor to trigger samples of specific performance event counts (e.g., cache misses, branch mispredictions) as well as from General-Purpose Registers (GPRs), offering improved insight into system behaviour without the overhead of continuous monitoring.
2) **OpenSBI Extension**: to provide supervisor-level access to the new EBS feature, we extend OpenSBI with new API

Work supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under project 2022.06780.PTDC (DOI: 10.54499/2022.06780.PTDC). We also acknowledge the contributions from projects UIDB/50021/2020 (DOI: 10.54499/UIDB/50021/2020), and the European High Performance Computing Joint Undertaking (JU) under Framework Partnership Agreement No 800928, Specific Grant Agreement No 101036168 (EPI SGA2) and Grant Agreement No 101143931 (POP3). The JU receives support from the European Union's Horizon 2020 research and innovation programme and from Croatia, Czech Republic, France, Germany, Greece, Italy, Netherlands, Portugal, Spain, Sweden, and Switzerland.

functions. These interfaces allow the operating system and hypervisors to control, configure, and retrieve performance data from the new CSRs performance counters securely.

3) **Validation**: we validate the proposed extension on a CVA6 core, embedded on an OpenPiton manycore framework, ensuring that the EBS system operates as intended and integrates smoothly in the existing RISC-V architecture.

The source code for the modified OpenPiton, CVA6, OpenSBI and tests are available at https://github.com/hpc-ulisboa/RVEBS.

II. Background

Although EBS has seen significant development across major architectures, its implementation varies in sophistication. Intel's PEBS, AMD's IBS, and ARM's SPE all provide detailed performance data, enabling precise software optimizations [16]. Meanwhile, RISC-V still lacks native support for EBS.

A. Event-Based Sampling: Intel, AMD and ARM

Precise Event-Based Sampling (PEBS) [5], available in Intel processors provides detailed performance data by capturing specific hardware events, with minimal overhead. Unlike traditional performance monitoring, PEBS directly associates performance events with specific instructions in the instruction pipeline. This is achieved by recording the architectural state (e.g., register contents and program counter) when an event of interest occurs, thereby providing more fine-grained and accurate performance data. PEBS also uses dedicated hardware buffers to store the samples, which can be later post-processed by insightful tools to allow the tuning of real-world workloads. Intel PEBS has been widely adopted in tools like Intel VTune Profiler [17] for profiling high-performance applications.

AMD, on the other hand, provides an Instruction-Based Sampling (IBS) mechanism [6] that supports two sampling types: instruction fetch sampling (IBS fetch) and micro-operation sampling (IBS op). Control registers are programmed depending on the sampling type, with one counter monitoring instruction fetches and another tracking micro-operations. When an event is sampled, information about the event is stored in Model-Specific Registers. However, while IBS offers rich insights into micro-operations, it lacks the granularity of sampling individual hardware events, like load or branch instructions. As a result, and despite providing valuable data, IBS is less accurate compared to Intel's PEBS [18]–[20].

ARM's precise event sampling mechanism is implemented through the Statistical Profiling Extension (SPE) [7]. Like AMD's IBS, SPE counts dispatched micro-operations, allowing it to monitor and sample data from those events.

B. RISC-V Performance Monitoring

As RISC-V gets growing adoption, the ISA has kept evolving, receiving several revisions and new extensions, as well as more sophisticated toolchains for software development and performance analysis (e.g., [4]). In what concerns the PMU, a minimum interface was initially defined in version 1.7 of the privileged specification [21], but has grown until version 1.11 [22]. Presently, RISC-V allows monitoring of both fixed (cycles and retired instructions) and up to 29 programmable counters, with the latter being implementation-dependent. Additionally, the specification allows inhibiting individual counters and controlling supervisor access to event counters.

The evolution of the RISC-V HPM specification align it more closely with other ISAs, showcasing a growing sophistication in performance analysis tools. However, advanced features such as EBS are yet to be specified on the RISC-V ecosystem, creating a critical gap in RISC-V's performance analysis infrastructure.

III. RVEBS

Traditional performance monitoring based on continuous sampling often imposes a non-negligible overhead, which may compromise the gathered system information. Accordingly, the proposed EBS extension significantly reduces this monitoring overhead while providing more contextually relevant performance data. It does so by implementing an event-triggered sampling system that captures the system state information only when specific events occur a predetermined amount of times.

The EBS system for RISC-V proposed in this paper is more closely related to the functionality of Intel's PEBS than the other implementations presented due to its low overhead, the high accuracy it achieves and its low sensitivity to sampling rate. However, while PEBS is highly effective, it is tightly integrated with Intel's proprietary architecture and uses interrupts to stop normal code execution to run microcode that captures system information. In contrast, our proposed RISC-V performance counter extension samples system information in parallel with normal code execution, never stalling the processor. By leveraging a similar sampling mechanism, our extension aims to deliver high-precision event recording and low-overhead performance monitoring, filling a critical gap in RISC-V's performance analysis tools.

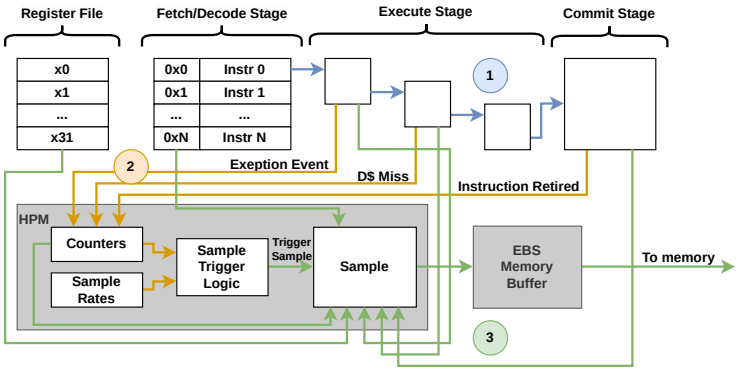
A. Proposed System Architecture

The proposed EBS system for RISC-V comprises three interconnected parts that work together to capture the performance data with minimal overhead, as it is shown in Figure 1a:

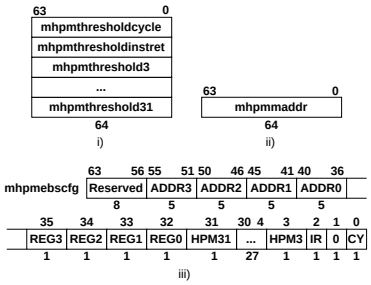
① the datapath of the processor under analysis, where instructions are fetched, executed, and committed; during program execution, events are monitored by the performance counters in the HPM module;

② the HPM module is responsible for monitoring and counting the hardware events; each time a specific event occurs in the datapath, a corresponding counter in the HPM is incremented; this process happens continuously as the program executes; however, event counting alone is not sufficient for detailed performance analysis, as it provides only cumulative data without context; to address this, the EBS system extends the HPM to trigger sampling when certain event thresholds are reached;

③ the sample and store logic is responsible for gathering data from multiple sources, including the HPM counters, the processor's datapath, and the register file; once the sampling is triggered, this logic captures a snapshot of the current system state, which includes key data like the Program Counter, performance counter values, and GPRs.



(a) Overview of the RVEBS system design: ① the processor's datapath; ② the event counting infrastructure; ③ the sampling and buffering system.



(b) Newly proposed CSRs to support the desired EBS: i) HPM threshold CSRs, `mhpmthresholdX`; ii) HPM memory address CSRs, `mhpmmaddr`; iii) EBS configuration register, `mhpmebscfg`.

Fig. 1: Proposed: (a) sampling system architecture; (b) new RISC-V HPM CSRs.

In the considered architecture, the sampled data is first stored in local registers before being transferred to a dedicated memory buffer. This buffer serves as an intermediate staging area so that the final memory transfer operation is only performed when the processor is not using the load/store system. Finally, the sampled data is committed to main memory, where it can be accessed by privileged software for analysis.

B. Current Missing Facilities

The existing RISC-V privileged specification version 20240411 already supports fixed and programmable event counting but lacks some important functionalities for event-based sampling. Specifically, it does not provide means to:

- set thresholds and compare them with counter values;
- trigger a sample when thresholds are reached;
- control the storage addresses of the samples;
- control which signals are sampled;
- request sample storage in the memory system.

To address these limitations, the proposed EBS system is designed to operate with minimal interference in the normal processing workflow. This system extends the RISC-V specification by adding the necessary hardware facilities and system-level support for effective EBS.

C. Proposed RISC-V Specification Improvement

To support EBS, the new CSRs presented in Figure 1b are introduced:

- **threshold management registers**: to manage the thresholds to be considered by each counter, allowing the system to compare register values against predefined limits and trigger the sampling when the thresholds are met;
- **memory address register**: to specify the physical address where sampled data is to be stored, ensuring a proper data management and retrieval;
- **configuration control register**: to select the signals to be sampled, including various counter values and GPRs.

These new CSRs are integrated within the machine-level address range to align with the RISC-V HPM specification.

D. OpenSBI EBS Extension

To allow secure and streamlined supervisor-level access to the proposed new facilities of the HPM, a new OpenSBI extension was specified. It provides a standardized interface for privileged software to interact with the performance monitoring hardware securely and efficiently. This ensures that only authorized supervisor-level software can configure the EBS system, preventing unintended access or manipulation of the performance counters and configuration CSRs.

The extension introduces a set of Supervisor Binary Interface (SBI) calls tailored to control the EBS system. These include:

- **event configuration**: the supervisor can configure which hardware events will trigger sampling;
- **threshold management**: the supervisor can set thresholds for event counts in the HPM;
- **memory write location**: after allocating a memory region, the supervisor can configure the EBS system to store samples in that location;
- **sample configuration**: the supervisor can program the system to sample any of the information available in that implementation.

By incorporating these functionalities, the OpenSBI extension ensures that the EBS system can be seamlessly integrated into existing RISC-V platforms. It provides the necessary tools for supervisor-level software to interact with the new CSRs in a way that is both flexible and secure. Moreover, the use of standardized SBI calls ensures portability across different RISC-V implementations. The design of the OpenSBI extension emphasizes modularity and scalability, allowing future enhancements or additional performance monitoring features to be added without requiring major changes to the interface.

IV. Implementation

The proposed EBS system, along with the OpenSBI extension, was implemented in the OpenPiton processor [23] [24] framework integrated with a CVA6 core [25]. Naturally, other processor environments and architectures could equally be considered with minor differences in the system integration.

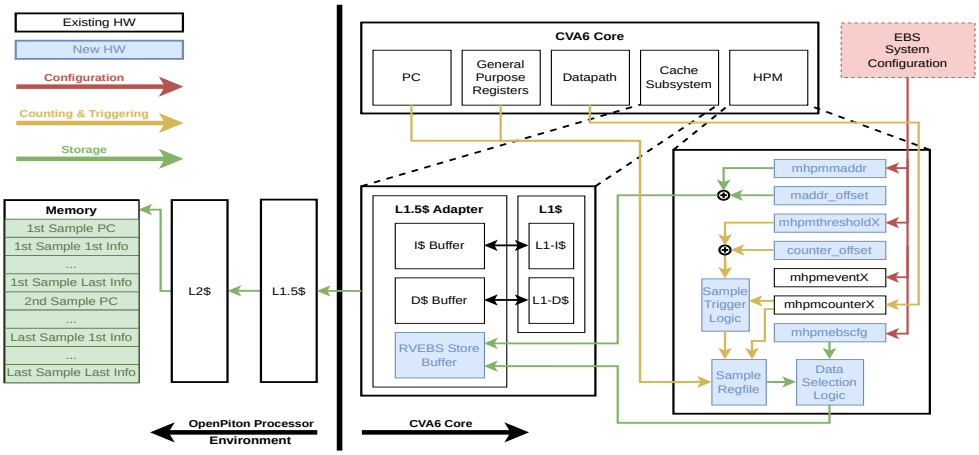


Fig. 2: Overview of the implemented EBS system.

The considered implementation required some modifications in the original system architecture, including changes in the HPM and in the core's cache subsystem to support new CSRs and efficient memory access for storing the sampled data. The CVA6 core already implements the `mcycle` and `minstret` fixed counters, together with 6 programmable counters. OpenPiton operates with three cache levels: L1 (divided into data and instruction caches), L1.5 (serving both as glue logic for the coherent mesh and as write buffer for the L1) and L2.

The complete RVEBS implementation, illustrated in Figure2, can be divided into three domains:

1) **system configuration**: to setup the event types, threshold values, and sampling behaviour;
2) **counting and sample triggering**: to monitor the events and trigger sampling when specific thresholds are met;
3) **data sampling and storing**: to handle the transfer of the sampled data into the main memory with minimal impact on the processor's performance;

Each of these components plays a crucial role in the system's ability to perform efficient and accurate event-based sampling.

A. System Configuration Domain

The configuration of the EBS system is a key element that defines the overall system operation. Besides determining which events to monitor through the `mhpmeventX` CSRs, this part allows the selection of:

- the considered sampling rates, through the `mhpmthresholdX` CSRs (see Figure 1b i));
- the memory location where samples are stored, through the `mhpmmaddr` CSR (see Figure 1b ii));
- definition of which information is stored in each sample, through the `mhpmebscfg` CSR (see Figure 1b iii)).

The implemented OpenSBI extension plays a critical role in this system configuration since the provided functions allow a simple and secure supervisor-level interface for setting up and managing the events, thresholds and sample options, making the system highly configurable for different performance monitoring scenarios. Its implementation includes several SBI calls responsible for interacting with the HPM CSRs: `ebs_get_event` and `ebs_set_event` functions manage the event configuration register; `ebs_get_threshold` and `ebs_set_threshold` interact with the threshold registers; `ebs_get_maddr` and `ebs_set_maddr` functions handle the memory address register; `ebs_get_ebscfg` and `ebs_set_ebscfg` deal with the EBS configuration register.

B. Counting and Sample Triggering Domain

This domain represents the main functionality of the proposed EBS system, determining when samples are captured.

At the occurrence of each event (previously programmed to be monitored), the corresponding counter is incremented. At this stage, it is important to note that the triggering procedure implemented in RVEBS is not controlled by counter overflows, as it is in other EBS implementations like Intel's PEBS [18]. Instead, each counter is paired with a `counter_offsetX` register, that stores the value of the corresponding counter at the last trigger occurrence. Hence, the trigger logic does not compare the counter values with the defined threshold but to the sum of the threshold with the count offset. This approach has a two-fold effect: i) it allows the counter to continuously increment, maintaining the existing HPM specification behaviour; ii) if a sample is delayed (a possibility discussed later in this Section), the following sampling intervals will not be affected.

Once a sample is triggered, the Sample Regfile (newly added to the HPM, see Figure 2) is updated. This register file is composed of 14 registers that hold all the information the `mhpmebscfg` CSR can select at the cycle the trigger occurs (i.e., the PC, all the counter values, and the GPRs indicated by the `ADDRX` fields of `mhpmebscfg`). Although this may require more hardware resources, it assures all the sample data is cycle-accurate. To access all 4 GPRs in one single cycle, the core's register file was modified to have 4 additional read ports.

C. Data Sampling and Storing Domain

Once a sample is triggered and the Sample Regfile is updated, the system must capture and store the relevant data without interfering with the ongoing execution of the processor.

Starting in the cycle after the trigger, the Data Selection Logic (see Figure 2) iterates over the Sample Regfile at 1 register per cycle and checks `mhpmebscfg` to determine if that value should be stored or not (except for the PC register, which

is always stored). If the information is configured to be stored, the register data along with the address it will be stored is sent to the RVEBS Store Buffer (newly added to the L1.5 Cache Adapter, see Figure 2). Once the buffer acknowledges the reception, the memory address offset register `maddr_offset` is incremented by 8 bytes, the length of each sampled information.

The RVEBS Store Buffer is a small FIFO implemented in CVA6 Cache Subsystem, at the L1.5 Cache Adapter module, where the request queues for the L1 data and instruction caches are already located. Following the principle of not interfering with the core's normal execution, the RVEBS Store Buffer only makes its requests to transfer the information into main memory when the L1 request queues are empty.

Naturally, in the event the L1 caches are very active, this might cause the RVEBS Store Buffer to fill-up and prevent the Data Selection Logic from sending more information to memory. In such a situation, the Data Selection Logic stops while the RVEBS Store Buffer is full, delaying new samples (if necessary) but never stalling the processor's execution path.

The store requested to the L1.5 cache are issued as non-cachable writes. This, along with the requests being injected to L1.5 through a dedicated buffer (with very little priority) mitigates the memory overhead of the EBS system.

V. Experimental Evaluation

To evaluate the effectiveness of the proposed EBS system for RISC-V, we mostly follow the methodology presented in [18], by gathering key metrics including instructions attribution, accuracy, sensitivity to sampling rate, and time overhead. We also measure the area overhead introduced by RVEBS.

Part of the presented results were obtained using a Verilator (version 5.008) simulation of the OpenPiton processor [24]. As a consequence of the long simulation time, the total number of instructions executed in each test had to be limited. Nevertheless, the ratios between each type of instruction have been maintained such as to guarantee accurate results.

A. Instruction Attribution

The first experiment aims to evaluate the potential bias introduced by the sampling mechanism and how accurate it attributes events to specific instructions that increment the same event count.

```
li a0, 100000
loop:
sb a1, 0(a2) # S1
sb a1, 0(a2) # S2
sb a1, 0(a2) # S3
sb a1, 0(a2) # S4
addi a0, a0, -1
bnez a0, loop
```

Listing 1: Store loop benchmark.

The conducted experiment consists of running a loop of store instructions a large number of times (see Listing 1), triggering samples with the store access event and using a sample interval indivisible by the number of store instructions as to prevent the sampling rate from always being reached by the same instruction. In this case, if the system presents no bias in the instruction attribution, each of the 4 store instructions should trigger 1/4 of the total samples.

The obtained results are validated by checking the PC of the instruction that triggered each sample, and show that RVEBS presents no bias in instruction attribution, as each store instructions is responsible for 25% of the sample triggers.

B. Event Sampling Accuracy

To evaluate the accuracy of the presented EBS mechanism, it was conducted an experiment that evaluates the number of samples taken from a benchmark with a known number of events. Similarly to the tests performed on PEBS and IBS [18], the number of retired instructions and L1 data cache misses were monitored while executing a RISC-V version of the Accuracy-Bench benchmark [18] (see Listing 2). This benchmark was slightly modified to reduce the simulation time: reduction of loop1 iterations to 100; reduction of loop0 iterations to 1000; and replacement of the pseudo-randomized pointer-chasing memory access pattern, by a simpler code that increments the memory pointer by the width of the L1 data cache line (128 bits). The considered sample rate was 10000. Furthermore, the `addi` instruction is used as many times as it is needed to fulfil the tested ratios between the number of load instructions and the total instructions inside loop0 (the tested ratios were $\frac{1}{20}$, $\frac{1}{40}$, $\frac{1}{60}$, $\frac{1}{80}$ and $\frac{1}{100}$). The evaluated accuracy was defined as the deviation between the number of samples captured by the EBS system and the expected ground truth (i.e., the known number of monitored events).

```
li a4, 100          // continues
loop1:              ld a2, 0(a3)
li a1, 1000         addi a3, a3, 16
loop0:              addi a1, a1, -1
...                 bnez a1, loop0
addi t0, t0, 1      addi a4, a4, -1
...                 bnez a4, loop1
```

Listing 2: Simplified Accuracy-Bench benchmark for RISC-V

On the implemented RVEBS system, the monitoring of the two considered events (retired instructions and L1 data cache misses) resulted in 100% accuracy for both events, when considering all load instruction ratios. Comparing to the results of [18], RVEBS performs similarly to PEBS, which also achieved 100%, and better than IBS, which reached its maximum accuracy of 97.6%, while operating in fetch mode and measuring the instructions retired with a $\frac{1}{20}$ load instruction ratio. This is likely due to IBS's delay between the sample interrupt triggering and the execution of the interrupt handler to pause the counters. In contrast, the implemented RVEBS system only takes 14 cycles to store an entire sample in main memory (assuming that there is no other simultaneous L1 cache requests to the next cache level).

The reason RVEBS attains 100% in all cases with a sampling rate of 10000 is because the contention caused by the load instructions is never enough to stall the Sample RegFile. In other words, even in the least favourable scenario of $\frac{1}{20}$ of loads per total loop instruction, each sample has, at least, a 9500

cycles time-slot to be copied from the RVEBS Store Buffer to main memory.

To introduce an accuracy loss in the RVEBS system, at least one of two conditions must be met: i) the sample rate is so low that it would require the samples to be spaced less than 14 cycles apart to remain accurate, or; ii) the memory contention is so heavy that there are not enough cycles with no L1 cache requests to save each sample.

This assumption is also valid by running an extra test using the instructions retired as the monitored event with two modified conditions: i) the ratio between load instructions and total instructions in loop0 is increased to $\frac{1}{4}$; and the sampling rate was decreased to 16. As expected, the accuracy in this run dropped to 88.8%, which still stands as a good performance given the test conditions.

C. Sensitivity to Sampling Rate

Another analysis relates the variation of the accuracy with the decrease in the sampling rate. To characterize this aspect, the same simplified Accuracy-Bench benchmark was utilized, this time configured to have a $\frac{1}{20}$ ratio of load instructions in loop0. The monitored event in this test was the L1 data cache miss and the sampling rate varied between 10 and 100000.

The obtained results show that RVEBS achieved 100% accuracy with all the tested sampling rates, which is better than the results observed by [18] for PEBS and IBS. The prior reaching 100% in all sampling intervals except for 100, in which it got 99.3%. For IBS, data in [18] only starts at a sampling interval of 10000, where it achieved just 41.0%; with the sampling rate set at 100000, IBS achieved 94.3%. This better performance can be explained by RVEBS implementation that ensures that the processor is never stalled or interrupted. Unlike PEBS and IBS, the samples are taken in parallel with the normal processor execution, making this method much lighter in terms of the total cycles needed to take and store each sample. Furthermore, under those conditions there are at least 190 cycles between each sample where the L1 caches do not make any request to the next cache level, which is more than enough for a RVEBS sample to be stored in main memory.

To evaluate the RVEBS sampling rate limit, another run was conducted with the sampling rate being set again to 10 and the monitoring of retired instructions. This setup should saturate the sampling system by reaching consecutive sample thresholds less than 14 cycles apart. Such assumption was confirmed and the resulting accuracy dropped to 18.1%. However, it should be noted that these conditions are not regular use-cases of EBS systems, and were not, as such, considered in [18].

D. Time Overhead

To determine the time overhead introduced by the proposed RVEBS system, the modified Accuracy-Bench benchmark was run once more while monitoring the retired instructions event and with a $\frac{1}{20}$ load instruction ratio, with and without the EBS system being active. Given that RVEBS was designed to prevent any stalling of the processors normal execution, it is expected that the run time of the benchmark is not affected.

TABLE I: Added silicon area required to omplement CVA6 with RVEBS. The added area is decomposed into HPM, L1.5 Adapter and GP Regfile modules. All values in μm^2 except for the $\Delta/\Delta_{\text{CVA6}}$ column.

Module	RVEBS	Vanilla	Δ	$\Delta/\Delta_{\text{CVA6}}$ [%]
CVA6 Core	239110.99	216568.71	22542.28	**10.4**
HPM	15307.05	3181.75	12125.30	**53.8**
GP Regfile	15065.31	9693.79	5371.52	**23.8**
L1.5$ Adapter	2959.41	2214.80	744.61	**3.3**
Others	205779.22	201478.37	4300.85	**19.1**

The obtained results show a 0.01% variation. This small difference is mostly related to the spent time to setup the sampling system. Once again, this result compare favorably with the ones observed by [18]. PEBS and IBS were reported to introduce 12.55% and 1.44% time overhead in similar conditions, respectively,

E. ASIC Synthesis

To assess the amount of necessary resources to implement the proposed RVEBS, an ASIC synthesis (with and without modifications) was run for UMC's 28nm technology with a target clock frequency of 500MHz (maximum possible value), with the existing SRAM modules of the cache memories being black-boxed to avoid any interference with the synthesis results. Given that all introduced modifications are located inside the CVA6 core, that was the top module that was synthesized.

As presented in Table I, the addition of the RVEBS system increased the total area of the CVA6 core by approximately $22542\mu\text{m}^2$ (10.4%). The HPM represents 53.8% of this area increase, since it contains the new CSRs, a state machine to control the sample taking and memory transfer, and the Sample Regfile. The next most increased component was the GPRs Regfile, which represents 23.8% of the total increase. This is mainly due to the 4 new read ports added to the component, which increase its complexity. Having a smaller (but still noticeable) impact, the L1.5 cache adapter modification amounted 3.3% of the total area increase. This module now contains the RVEBS Store Buffer.

It is also important to note that the synthesis reported that none of the introduced or modified components are part of the critical path, which remains in the fused multiply-add facilities.

VI. Conclusions

In this paper, we have introduced the first RISC-V Event-Based Sampling (RVEBS) system designed for comprehensive performance monitoring and application profiling. Our system builds upon existing RISC-V specifications, incorporating necessary modifications to enable the desired functionality. We demonstrate an implementation use case on an OpenPiton processor featuring a CVA6 core on 28nm CMOS technology. The results indicate that the proposed scheme is lightweight, highly accurate, and does not impact the processor's critical path, while maintaining minimal impact on overall application performance. Moving forward, we plan to develop the RVEBS framework further and explore its application across different computational workloads.

REFERENCES

[1] B. Sprunt, "The basics of performance-monitoring hardware," *IEEE Micro*, vol. 22, no. 4, pp. 64–71, 2002.

[2] W. Krenn, A. Wilson, A. Suresh, and M. Freiberger, "The european chips act, the isolde project, and open-source hardware," in *2024 Argentine Conference on Electronics (CAE)*, pp. 136–141, 2024.

[3] F. Mantovani, P. Vizcaino, F. Banchelli, M. Garcia-Gasulla, R. Ferrer, G. Ieronymakis, N. Dimou, V. Papaefstathiou, and J. Labarta, "Software development vehicles to enable extended and early co-design: A risc-v and hpc case of study," in *High Performance Computing* (A. Bienz, M. Weiland, M. Baboulin, and C. Kruse, eds.), (Cham), pp. 526–537, Springer Nature Switzerland, 2023.

[4] J. M. Domingos, T. Rocha, N. Neves, N. Roma, P. Tomás, and L. Sousa, "Supporting risc-v performance counters through linux performance analysis tools," in *2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 94–101, IEEE, 2023.

[5] Intel, "Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 3B," tech. rep., Intel, 2016.

[6] A. M. Devices, *AMD64 architecture programmer's manual volume 2: System programming*, 2006.

[7] ARM, *Arm Architecture Reference Manual for A-profile architecture, Version DDI 0487K.a, Chapter D16*.

[8] M. A. Sasongko, M. Chabbi, P. Akhtar, and D. Unat, "Comdetective: A lightweight communication detection tool for threads," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, (New York, NY, USA), Association for Computing Machinery, 2019.

[9] L. Luo, A. Sriraman, B. Fugate, S. Hu, G. Pokam, C. J. Newburn, and J. Devietti, "Laser: Light, accurate sharing detection and repair," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 261–273, 2016.

[10] X. Liu and J. Mellor-Crummey, "A tool to analyze the performance of multithreaded programs on numa architectures," *SIGPLAN Not.*, vol. 49, p. 259–272, feb 2014.

[11] M. A. Sasongko, M. Chabbi, M. B. Marzijarani, and D. Unat, "Reuse-tracker: Fast yet accurate multicore reuse distance analyzer," *ACM Trans. Archit. Code Optim.*, vol. 19, dec 2021.

[12] C. Helm and K. Taura, "Perfmemplus: A tool for automatic discovery of memory performance problems," in *High Performance Computing* (M. Weiland, G. Juckeland, C. Trinitis, and P. Sadayappan, eds.), (Cham), pp. 209–226, Springer International Publishing, 2019.

[13] P. Roy, S. L. Song, S. Krishnamoorthy, and X. Liu, "Lightweight detection of cache conflicts," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, CGO 2018, (New York, NY, USA), p. 200–213, Association for Computing Machinery, 2018.

[14] M. Srinivas, B. Sinharoy, R. J. Eickemeyer, R. Raghavan, S. Kunkel, T. Chen, W. Maron, D. Flemming, A. Blanchard, P. Seshadri, J. W. Kellington, A. Mericas, A. E. Petruski, V. R. Indukuru, and S. Reyes, "Ibm power7 performance modeling, verification, and evaluation," *IBM Journal of Research and Development*, vol. 55, no. 3, pp. 4:1–4:19, 2011.

[15] H. Cui, S. Liang, Y. Cui, W. Zhang, H. Zhan, C. Yang, X. Liu, and X. Cheng, "A hardware-software cooperative interval-replaying for fpga-based architecture evaluation," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–2, 2023.

[16] V. M. Weaver *et al.*, "Advanced hardware profiling and sampling (pebs, ibs, etc.): creating a new papi sampling interface," *Technical Report UMAINE-VMWTR-PEBS-IBS-SAMPLING-2016-08. University of Maine, Tech. Rep.*, 2016.

[17] Intel, *Intel® VTune™ Profiler User Guide Version 2024.2*.

[18] M. A. Sasongko, M. Chabbi, P. H. J. Kelly, and D. Unat, "Precise event sampling on amd vs intel: Quantitative and qualitative comparison," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–15, 2023.

[19] P. J. Drongowski, "Instruction-based sampling: A new performance analysis technique for amd family 10h processors," *Advanced Micro Devices*, vol. 1, no. 3, p. 11, 2007.

[20] J. Dean, J. E. Hicks, C. A. Waldspurger, W. E. Weihl, and G. Chrysos, "Profileme: Hardware support for instruction-level profiling on out-of-order processors," in *Proceedings of 30th Annual International Symposium on Microarchitecture*, pp. 292–302, IEEE, 1997.

[21] A. Waterman, Y. Lee, R. Avizienis, D. A. Patterson, and K. Asanovic, "The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 1.7," *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2015-49*, vol. 49, 2015.

[22] A. Waterman and K. Asanović, "The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 20190608-Priv-MSU-Ratified," *RISC-V Foundation*, June 2019.

[23] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff, "Openpiton: An open source manycore research framework," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, (New York, NY, USA), p. 217–232, Association for Computing Machinery, 2016.

[24] J. Balkind, K. Lim, F. Gao, J. Tu, D. Wentzlaff, M. Schaffner, F. Zaruba, and L. Benini, "Openpiton+ ariane: The first open-source, smp linux-booting risc-v system scaling from one to many cores," in *Workshop on Computer Architecture Research with RISC-V (CARRV)*, pp. 1–6, 2019.

[25] 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," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, pp. 2629–2640, Nov. 2019.