High-Integrity Performance Monitoring Units in Automotive Chips for Reliable Timing V&V

Enrico Mezzetti*, Leonidas Kosmidis*, Jaume Abella*, Francisco J. Cazorla*†

* Barcelona Supercomputing Center (BSC), Spain

† IIIA-CSIC, Spain

Abstract

As software continues to control more system-critical functions in cars, its timing is becoming an integral element in functional safety. Timing validation and verification (V&V) assesses software's end-to-end timing measurements against given budgets. The advent of multicore processors with massive resource sharing reduces the significance of end-to-end execution times for timing V&V and requires reasoning on (worst-case) access delays on contention-prone hardware resources. While Performance Monitoring
Units (PMU) support this finer-grained reasoning, their design has never been a prime consideration in high-performance processors – where automotive-chips PMU implementations descend from – since PMU does not directly affect performance or reliability. To meet PMUs instrumental importance for timing V&V, we advocate for PMUs in automotive chips that explicitly track activities related to worst-case (rather than average) software's behavior, are recognized as an ISO-26262 mandatory high-integrity hardware service, and are accompanied with detailed documentation that enables their effective use to derive reliable timing estimates.

Index Terms – Multicore Automotive Chips, Performance Analysis, Performance Monitors

I. Introduction

The number of mechanical automotive subsystems being enhanced or completely replaced by electrical/electronic (E/E) components is on the rise. Due to their safety-critical nature, it is mandatory to bring evidence that E/E systems behave correctly. ISO-26262 [1], which is becoming the prevalent safety standard for road vehicles, explicitly addresses the identification of functional and non-functional software safety requirements. For software timing, which falls into the latter category, ISO-26262 associates a *time budget* to each software unit. Budgets are determined on the basis of estimates or bounds to the software Worst-Case Execution Time (WCET) behavior. WCET estimates and budgets are therefore the basic elements for determining feasible task schedules, and for assessing the timing behavior of the overall system.

Timing V&V heavily relies on extensive testing aiming at spotting timing failures, i.e. a software component overrunning its assigned time budget (functional failures, which also have a prominent role in ISO-26262, are not covered in this work). The absence of timing failures during the testing phase serves as an argument to sustain the correctness of the software timing behavior. The effectiveness of the consolidated approach to timing V&V, which has historically focused on (just) assessing end-to-end execution times against the assigned timing budget, is hampered by the advent of complex platforms comprising high-performance features. In the automotive domain, on-board systems already embed software in the order of hundreds of millions of lines of code [2], with some complex functionalities, such as Advanced Driver Assistance Systems (ADAS), projected to increase their computational needs by 100x in coming years [3]. These levels of performance are met with high-performance multicore processors comprising features like caches and accelerators (historically not used in automotive) like the NVIDIA DrivePX, RENESAS R-Car H3, QUALCOMM SnapDragon 820 and Intel Go.

Software's timing behavior on top of high-performance hardware is hard to model accurately due to overly extensive and highly dispersive documentation of the latter, with possibly incomplete and inaccurate coverage of timing information. Further, timing behavior cannot be verified to a sufficient extent with end-to-end measurements due to complex interactions among hardware resources. For instance, the fact that a given task might incur long memory latencies due to cache misses, can be concealed by the same task experiencing (luckily) low contention delays in other resources, e.g. in the bus, preventing the task itself from violating its overall expected budget. End-to-end observations might then fail in capturing the task miss rate as a possible cause of exceeding the budget during operation. As a result, end-to-end timing behavior (i) fails to expose the main features affecting execution time, (ii) can hide the fact that some factors that contribute to the overall timing may compensate each other, and hence (iii) cannot be considered anymore as a reliable indicator for deriving WCET estimates. In this same line of reasoning, end-to-end measurements cannot be even used as means of evidence for sufficient independence between mixed-criticality software elements, referred to as "freedom from interference" [1].

Finer-grained metrics (e.g. shared resources utilization and worst-case delays) are required to support timing-related safety arguments in the face of an independent certification/qualification authority or in-house department. The ability to break

down task execution time at a finer granularity level than end-to-end observations provides stronger timing evidence, fundamental to increase the level of confidence on WCET estimates and thus on the adequateness of timing budgets.

While the Performance Monitoring Units (PMUs) in mainstream processors do offer a promising baseline for this low-level analysis, the historical role and limited relevance PMUs have been given in mainstream systems – from where PMU design in automotive chips is inherited – is in strident contrast with the critical role they would acquire for timing analysis. In fact, PMUs and Performance Monitoring Counters (PMCs) have been traditionally intended to capture average behavior rather than the worst-case one and have been used as cursory, low-level debugging support by the chip manufacturer (hence with reduced need for detailed documentation). Moreover, the fact that PMU and PMCs do not directly impact the timing and functional behavior of applications running on top of the platform has a twofold consequence: first, PMU's inclusion in the hardware design usually occurs in late design phases, with reduced flexibility to incorporate new counters or to fix potential deviations; and second, the PMU does not need to comply with high-integrity constraints and can be designed according to low-integrity (e.g., ASIL-A) requirements. This difference in integrity level exposes system designers to the evident paradox of using low-integrity, poorly-documented PMUs as the basis for timing analysis mechanisms that are expected to guarantee that the system achieves enough freedom from interference for higher-integrity tasks (e.g., ASIL-C/D).

Our claim is that high-integrity (i.e. ASIL C/D), WCET-aware, well-documented PMUs will become an instrumental tool to simplify and consolidate the arguments in support to timing V&V in the presence of automotive multicore complex processors. In this paper we make the case for new requirements and guidelines in the design and implementation of PMUs that can be reliably used to collect timing information. In particular, our proposal focuses on three major aspects:

1) *WCET focus*: PMUs shall be oriented towards the analysis of the worst-case behavior rather than the average one only. This means, PMCs should allow deriving worst-case delays and execution time. We report our experience in implementing some of those events in an FPGA implementation of a real-time multicore processor;
2) *Quality of documentation*: PMU documentation has to be significantly improved with respect to current practice. Emphasis should be put on availability (i.e., disclosure) and clarity of the information. To support this argument, we report our experience in mastering the PMU of an automotive processor;
3) *PMU criticality level*: PMUs must be designed, implemented and tested under the highest ASIL constraints given their central role in software timing validation.

II. PMU Properties in Mainstream Systems

The design and philosophy in the PMUs in automotive chips are inherited from mainstream high-performance systems, for which PMCs were first introduced. In the following we analyze relevant common features in modern PMUs and discuss how they fit in the emerging scenario where PMUs are increasingly used for timing V&V of multicore automotive chips.

Number of events and PMCs. Events and PMCs address different concepts: events refer to those activities (e.g. cache misses, bus access count) that can be monitored, while PMCs are user-visible and configurable registers that count events.

Events. The number of events in modern mainstream processors reaches hundreds. Despite their high number, their design philosophy is not monitoring worst-case delays and contention across tasks. Instead, they are typically intended for debugging purposes and optimization of average system performance. With currently tracked events, one can easily derive whether a task was delayed because of some type of stall in the memory system, which blocked a core resource (e.g. the load/store unit). By contrast, it is hard – if at all possible – to derive how long a task running in a core was delayed due to the requests of other tasks to the memory system (e.g. bus, cache, memory controller). The latter, however, is fundamental to articulate whether the delays and worst-contention delays observed for the execution of a task come from its intrinsic behavior or were induced by the activities of other tasks. Hence, existing events in mainstream processor architectures do not allow analyzing the (worst) contention that tasks can create on each other [4] (more details in Section III).

PMCs. In general, processors feature a limited number of PMC registers, e.g. 4-8 [5] in IBM POWER series and 64 in the IBM BlueGene [5] series. To track more events than available PMCs, multiplexing techniques can be used. Alternatively, the end user is required to carry out several runs to collect several event sets. Depending on the processor, PMC multiplexing can be implicit, allowing to measure more events than the physical registers, in exchange of accuracy due to event sampling. Other processors expose this limitation to the programmer, limiting the collection of events to the number of physical registers, frequently allowing only certain events to be counted with a given register. This results in scenarios where a particular combination of events cannot be monitored in the same execution, due to the fact that they are assigned to the same physical register. Timing experts have to factor in the WCET estimates the sampling inaccuracy (deriving bounds to it) and/or design experiments to collect the set of events required, accounting for the discrepancies across experiments (e.g. several runs of the *same* experiment may produce different event counts due to limited controllability).

PMUs not directly contributing to average performance. Although in recent years several works have shown that PMUs have positive repercussions on software performance and software engineers productivity [6][7], PMUs per se are not mechanisms directly increasing average performance. This naturally relegates PMUs to late chip design phases with limited time to solve issues that can arise, test PMUs deeply and document them conveniently. This translates into several problems for the end user. Reduced – or hard to find documentation – causes users working out costly reverse-engineering solutions. Further, limited documentation contributes to the lack of understanding of some sources of variability in some events, under apparently the same system setup. This causes some PMC features to be reported as unknown [8], reducing the confidence the user can reasonably place on PMUs.

PMUs not affecting functional safety. In real-time systems PMUs do not have a direct impact on hardware reliability, i.e. a faulty PMU does not cause the chip to become functionally inoperative. This naturally makes chip vendors (in the real-time market) reluctant to design PMUs under the requirements of ASIL levels. For example, PMCs neither feature protection against faults such as ECC or hardware redundancy, nor are deeply assessed against design faults, potentially creating another source of uncertainty in the measurements and, consequently, in the trustworthiness of their values.

III. Crossroads for PMU design in Automotive Chips

PMUs design and implementation in high-performance automotive processors should meet the requirements coming from current practice on WCET analysis.

Analytic modeling (more frequently used in other domains than in automotive) faces a *complexity wall* when building reliable and tight timing models of complex hardware. The latter, not only builds on information about hardware internals, in many cases subject to IP protection, but also requires models whose complexity is well beyond any timing model developed to date. This is confirmed by recent studies of major companies in domains such as Automotive for the ARM-based SABRE Lite multicore system by Renault [9] and Avionics for the NXP P4080 by Airbus [10], which build on some form of measurement-based analysis to derive timing bounds.

Measurement-based timing analysis modeling – the most common industrial practice in automotive – typically uses the highwater mark (HWM) measurement and adds to it a safety margin to account for unobserved behavior. This approach suffices for simple (e.g. SCADE-like) software that incurs limited execution time variability when runs on simple processors, but hits an *uncertainty wall* with the emerging systems. For complex hardware, the confidence of WCET estimates obtained with measurement-based timing analysis builds on the user's ability to reason on the application's timing behavior and, in particular, on its interaction with hardware components. The bounds to the maximum contention delay that a given request (or cumulatively all the requests of a task) can suffer are needed to provide evidence of the correctness of WCET estimates for ISO-26262. Deriving such bounds requires constructing ad-hoc test cases where empirical execution-time measurements are shown not to violate those bounds.

PMUs will be instrumental to cope with some of the current limitations of timing analysis on complex high-performance hardware platforms. Some preliminary studies in the real-time domain already explore the role of PMUs in deriving evidence on software correct timing behavior. To that end, specific events such as per-task (and possibly per-type) access counts to different shared resources are tracked [10], [11]. Those figures can be used to define shared-resource access or usage *quotas* that are later enforced during system operation, via specific run-time enforcement mechanisms. While existing PMUs already provide valuable features for timing analysis, current (average-performance centric) PMU design, implementation and documentation does not suffice to keep up with (i) the hardware and software platform complexity increase and (ii) the more stringent safety requirements as ADAS autonomy levels increases. Additional more reliable and suitable PMUs are needed to obtain solid evidence on the timing behavior on top of future high-performance automotive processors.

A. Events, PMCs and their usage

Mainstream processors, such as IBM POWER, Intel x86 and ARM A series, provide a set of PMCs that count specific events such as floating-point operations, cache misses or processor pipeline stalls [4]. The particular events monitored allow tuning average performance, but their focus is not monitoring worst-case delays, e.g. contention across tasks.

From those events a Cycle stack or CStack (aka CPIStack) can be built. The CStack provides a breakdown of how a given application 'has consumed' each cycle following a hierarchical approach. Focusing on the example provided in Figure 1(a), the *total execution cycles* bar on the left corresponds to the end-to-end execution time of the application. This component is divided into few first-level sub-components, which usually include: running time (covering the cycles in which one or more instructions were committed), idle cycles (in which no instruction was committed because the pipeline was empty), stall cycles (in which no instruction was committed because some resources got full), and other (that covers the remaining cycles that cannot be classified in any of the previous categories). Each first-level component can be further subdivided into several

second-level components. For instance, stall cycles normally are split per main resource on which a stall can be incurred: floating point unit, fetch unit, load/store unit, etc. The process can continue for few additional levels depending on the PMC/event support.

By operating (i.e. adding and/or subtracting) on PMC readings from different events, the user can build the CStack. Interestingly, the CStack has evolved from models proposed by researchers to more reliable models provided by chip vendors, e.g. IBM provides the CStack for its POWER7 processor [6]. Based on the same type of events, models to identify the performance criticality (different from the integrity-criticality described in the Introduction) of each thread in a given task have also been proposed [7]. Further extensions for energy tracking have also been considered [12].

Per-core contention delay: High-performance hardware has high complexity from safety standards perspective, meaning that it is difficult to control and verify. The trend towards multi- and many-cores makes contention in shared resources to become one of the most difficult elements to analyze and control. For worst-case (contention) delay analysis, we advocate for adequate PMU support in way similar to what has been proposed in [4], focusing on the relevant hardware events that need to be tracked for a reference platform in the aerospace domain. The following illustrative example shows how worst-case related information can be derived. The well-known Advanced Microcontroller Bus Architecture (AMBA) protocol defines the signals HBUSREQ$_i$, which is set by master M_i to request the bus, and HGRANT that is set by the arbiter to identify the master (e.g. HGRANT= j) that is granted access to the bus in each cycle. By tracking the number of cycles during which HBUSREQ$_i$ is set and HGRANT= j, with $i \neq j$, we obtain the number of cycles M_i is stalled by M_j [4]. Tracking this information, including the time M_i uses the resource, for each of the N_M masters requires $N_M \times N_M$ registers.

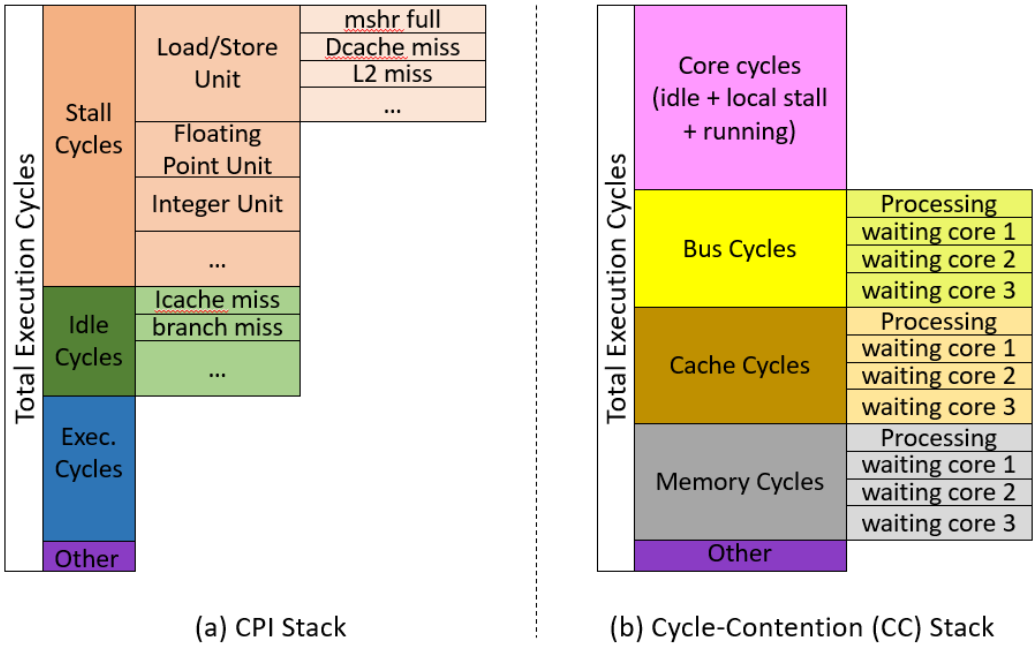

Fig. 1. Illustrative Examples of CPIStack (or CStack) and CCStack

Following this philosophy in the events to track, we can also derive how long each task has been delayed by others in the access to some hardware resources. This information is illustrated in Figure 1 (b) as a Cycle-Contention Stack (CCStack) [4] for a multicore with private first level instruction and data caches and whose main shared resources are the bus, the L2 cache and the memory interface. The CCStack splits the time that the application was running into two main categories: cycles in which the application was actually running or stalled due to its intrinsic behavior; and cycles in which the application was stalled due to the contention generated by tasks running in other cores. In the former category, we fit the first component in Figure 1 (b) that captures the time the task was running or stalled/idle due to local (intrinsic) events and the time the task was processing a request in shared resources. The second category covers all those components that capture the contention delay experienced by the task, labeled as 'waiting [for] core X' in Figure 1 (b).

With the CCStack, the time a critical task is delayed in a given resource collectively by tasks running on any other core is given by the addition of all 'waiting core X' components for that resource. Likewise, the time the critical task is delayed collectively across all resources by another task running on a given core, e.g. the one running in core 3, can be derived by

adding up all ‘waiting core 3’ CCStack components. Both are very valuable pieces of information to provide evidence about the correctness of the timing behavior of a task according to its safety timing specification.

Evidence for certification: The proposed WCET-centric PMCs can also track the longest contention delay a request from a given core can cause on others, which is fundamental for validation of WCET estimates. Several techniques have been proposed to *estimate* the worst-contention delay in the accesses to hardware shared resources, which are used as building blocks for multicore WCET estimation. Tests can be built to add high load on a resource (e.g. the bus), checking with the proposed PMCs whether the contention delay observed for any request goes beyond the estimate made. The absence of this scenario, together with an explanation of the experiment carried out to cause high load on the bus, serves as evidence on the correctness of the estimation to the worst-case contention delay.

The benefits expected from the sought PMU support are not limited to guiding the analysis process and consolidating its results. PMU support can also be exploited after production and deployment when in-field observations are collected. The cause of a detected software timing violation is often as opaque as the hardware internals and can only be explained with the support of WCET-centric PMUs. Moreover, insightful information from PMUs can also be exploited to identify for example anomalous (i.e., unobserved at analysis time) magnitude of contention from a given core and on a given resource and trigger corrective actions.

Overall, a good set of contention-aware events is the basis for timing V&V to assess that all worst-case delays are not violated, hence providing evidence for ISO-26262 qualification.

B. Documentation Quality

Several factors are specifically critical for automotive in terms of documentation depth, accuracy, and technical content.

In terms of depth, the amount of information currently provided in relation to PMUs is evidently scarce when compared with, for example, the information on other debug support functionalities. It is quite common to have event description limited to a single line of text or a short entry in a table. This trend reflects the fact that, in contrast to other functional debug features (e.g. intercepting communication errors), performance monitoring has been historically considered useful only for high-level verification of the hardware design and coarse-grained profiling information.

In terms of availability, PMU information is usually not fully provided in the processor programmer’s manual (or alike). Many events and PMCs are provided in documents only available upon request or directly subject to NDA. The availability of the information may also reduce when there is a hardware IP provider company and a manufacturer company (as opposed to a single company holding the IP and implementing the chip). In fact, some events are marked as *implementation dependent*. This may result in several documentation sources, not necessarily consistent among them.

In terms of accuracy and technical content, relevant information is sometimes unavailable or blurred, to avoid PMUs to indirectly disclose sensitive details on the underlying IP. While the rationale is clearly shareable, this should not prevent from exploiting the PMU altogether. Timing information in processor manuals can even be inaccurate, with disruptive effects on modeling of timing. For instance, the ARM Cortex-R5 – specifically targeting real-time systems – was accompanied by errata specification of timing: in revision *r1p2* the stall cycles caused by the divider unit and the latencies of some operations (VDIV.F64 and VSQRT.F64) were incorrect and only rectified in subsequent revisions. Next, we provide our first-hand experience when trying to master the PMUs of a processors used in automotive.

The **QUALCOMM SnapDragon** family of processors, based on the ARM big.LITTLE architecture , are used in automotive chips like RENESAS R-Car H3. These processors consist of two clusters with 4 cores each, one of them optimized for performance whereas the other is optimized for power efficiency. In-core events are monitored with PMCs documented by ARM, whereas most off-core events (including those beyond the L2 cache) are implemented – and so documented – by QUALCOMM. Unfortunately, our practical experience is that ARM documentation for the PMCs of this architecture, which has become no longer public, is scarce and incomplete. For instance, each data cache miss may produce two L2 accesses even if data are present in L2. While we suspect this effect relates to the prefetcher, incomplete documentation does not allow even to turn it off. QUALCOMM documentation for PMCs is simply not available.

All in all, automotive chip providers should find a good balance between extended PMU documentation and IP protection, and promote more fluent interactions with timing analysis experts, to better understand how typical usage of PMUs is evolving and which pieces of information are sought. Providing this support is fundamental for software timing verification and will become even more important as the autonomy of ADAS increases up to level five - under level 5 autonomy, ADAS control cannot be given back to the human driver under any driving scenario – carrying a significant increase in the evidence required for safety assurance.

C. *ASIL-compliant design*

The ISO26262 safety lifecycle captures safety-related activities during the concept, development and production phases (the latter is not covered in this document). The concept phase is concerned with the items to be developed in the course of the lifecycle, with emphasis on formulating high-level safety requirements (aka safety goals) to prevent hazardous situations. ISO26262 dictates that each safety goal is attached an ASIL, and used to derive a set of functional safety requirements, and to allocate them to specific item's architectural elements. The development phase determines how the hardware and software layers implement the safety functionality required to achieve the safety goals. ISO26262 defines hardware and software development workflows as shown in Figure 2.

An important step in PMU design and implementation is determining its target ASIL. With the trend towards mixed-criticality software, automotive chips are going to support the simultaneous execution of software with different and apparently incompatible ASIL (from A to D). Further, our proposed WCET-aware PMU contributes to timing correctness, and hence may also influence the probability of violation of safety goals. For these reasons, PMU ought to be designed and verified following the same practices as for the other hardware components part of ASIL C/D subsystems. The assessment of the ASIL compliance for a subsystem (HW2) defines the ASIL of its components, which in turn depends on their architecture. For instance, using redundant and diverse implementations is an effective safety measure to reach higher ASIL than those of individual components. This holds because individual random hardware faults or residual design faults do not cause a system failure, mitigating the risk due to single-point failures.

PMUs require undergoing appropriate fault analysis to identify potential faults, mitigate their probabilities with appropriate designs (HW3) and make sure that safety is not affected by placing the appropriate means (HW4-HW6 defines metrics and tests to provide evidence of compliance). This includes (i) dedicated safety measures (e.g. redundancy, diversity, hardened designs), (ii) the ability to shift the system to a safe state in a sufficiently short period of time, and (iii) notifying the driver (or make the fault perceptible) so that harmful consequences of faults can be avoided. Further, PMU fault analysis requires classifying faults according to the severity of their impact on the safety function (hardware fault tolerance), and assessing the coverage attained with the safety measures in place. As a rule of thumb, the higher the ASIL level, the higher the requirements imposed in terms of fault tolerance and diagnosis coverage required: their combined effect must comply with specific levels, intended to guarantee that the failure rate per hour is below ASIL-dependent predefined thresholds.

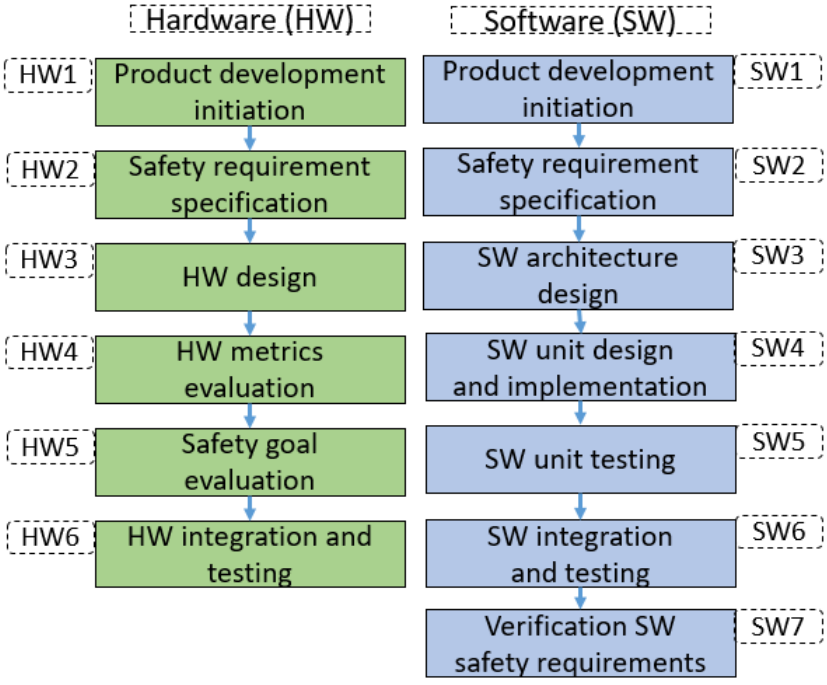

Fig. 2. Sketch of ISO26262 hardware and software development workflows

The interaction with the software workflow, outlined in the right part of Figure 2 is equally critical. During SW2, and in conjunction with HW2, the set of hardware events that demand high ASIL should be identified. These events, including for

instance those used to build the CCStack, together with the PMCs and the required PMU logic, should be designed to meet high ASIL. The rest of the events, could be designed, developed and evaluated (HW3-HW6) under less stringent ASIL. This allows achieving a good balance among hardware overhead and high-ASIL compliance. Coming back to the software workflow, ISO-26262 requires WCET estimates and budgets to be defined during the software architectural design phase (SW3), which could build on the PMU support in place. Conversely, as part of the (timing) unit testing and integration testing (SW5 and SW6), the user needs to provide evidence that the software has sufficient time budget to complete according to its assigned ASIL. Evidence from PMU should expose fine-grain timing metrics such as, for example, contention effects and assess those metrics against predefined bounds.

Overall, PMU deployment for automotive chips requires a suitable combination of design and verification practices, together with safety measures so that the most stringent integrity levels can be achieved when building timing guarantees upon PMU measurements.

IV. Conclusion

The increasing demand for guaranteed performance in the automotive domain calls for the adoption of mainstream processors whose complexity largely exceeds the capabilities of existing timing analysis techniques. In order to obtain sufficient evidence for safety certification, timing V&V techniques require appropriate hardware support to model contention in shared resources, which are abundant in high-performance multi- and many-cores. However, existing PMUs fail to capture and expose relevant events to measure worst-case access delays (including contention), their design does not meet the highest integrity levels, and their documentation often provide inadequate levels of accuracy, coverage and availability.

We propose leveraging WCET-aware PMUs to obtain evidence for certification in the automotive domain. In particular, we contend that the ability to derive worst-case access delays and per-task contention delay is pivotal for timing V&V. Achieving this requires some changes in PMU design and implementation: WCET awareness, improved documentation, and design and verification towards achieving safety certifiability against the highest ISO-26262 levels (C and D).

Acknowledgments

This work has also been partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO) under grant TIN2015-65316-P and the HiPEAC Network of Excellence. Jaume Abella has been partially supported by the MINECO under Ramon y Cajal postdoctoral fellowship number RYC-2013-14717. Enrico Mezzetti has been partially supported by the Spanish Ministry of Economy and Competitiveness under Juan de la Cierva-Incorporación postdoctoral fellowship number IJCI-2016-27396.

References

[1] International Organization for Standardization, *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.

[2] R. N. Charette, “This car runs on code,” in *IEEE Spectrum Magazine and IEEE Spectrum Online*, vol. 46, no. 2, February 2009, p. 3.

[3] ARM, “ARM Expects Vehicle Compute Performance to Increase 100x in Next Decade,” https://www.arm.com/about/newsroom/arm-expects-vehiclecompute-performance-to-increase-100x-in-next-decade.php, 2015.

[4] J. Jalle, M. Fernandez, J. Abella, J. Andersson, M. Patte, L. Fossati, M. Zulianello, and F. J. Cazorla, “Contention-aware performance monitoring counter´ support for real-time MPSoCs,” in *11th IEEE Symposium on Industrial Embedded Systems (SIES)*. IEEE, May 2016, pp. 1–10.

[5] V. Salapura, K. Ganesan, A. Gara, M. Gschwind, J. C. Sexton, and R. Walkup, “Next-generation performance counters: Towards monitoring over thousand concurrent events,” in *Proceedings of the ISPASS 2008 - IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE Computer Society, April 2008, pp. 139–146.

[6] IBM, *CPI events and metrics for POWER7*, https://www.ibm.com/support/knowledgecenter/linuxonibm/liaal/ iplsdkcpievents.htm.

[7] K. D. Bois, S. Eyerman, J. B. Sartor, and L. Eeckhout, “Criticality stacks: identifying critical threads in parallel programs using synchronization behavior,” in *The 40th Annual International Symposium on Computer Architecture, ISCA’13*. ACM, June 2013, pp. 511–522.

[8] V. M. Weaver and S. A. McKee, “Can hardware performance counters be trusted?” in *2008 IEEE International Symposium on Workload Characterization*. IEEE Computer Society, Sept 2008, pp. 141–150.

[9] A. Blin, C. Courtaud, J. Sopena, J. L. Lawall, and G. Muller, “Maximizing parallelism without exploding deadlines in a mixed criticality embedded system,” in *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE Computer Society, July 2016, pp. 109–119.

[10] J. Nowotsch, M. Paulitsch, D. Buhler, H. Theiling, S. Wegener, and M. Schmidt, “Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement,” in *2014 26th Euromicro Conference on Real-Time Systems*. IEEE Computer Society, July 2014, pp. 109–118.

[11] G. Fernandez, J. Jalle, J. Abella, E. Q. nones, T. Vardanega, and F. J. Cazorla, “Resource usage templates and signatures for COTS multicore processors,” in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. ACM, June 2015, pp. 1–6.

[12] R. Zamani and A. Afsahi, “A study of hardware performance monitoring counter selection in power modeling of computing systems,” in *2012 International Green Computing Conference (IGCC)*. IEEE Computer Society, June 2012, pp. 1–10.