

Heterogeneous MPSoCs for Mixed-Criticality Systems: Challenges and Opportunities

Mohamed Hassan
Intel Corporation

Editor's note:

This article presents the challenges and the opportunities in designing mixed-criticality systems with heterogeneous Multiple-Processor System-on-Chip (MPSoC) architectures.

—Tulika Mitra, National University of Singapore

—Jürgen Teich, University of Erlangen-Nürnberg

—Lothar Thiele, Swiss Federal Institute of Technology, Zurich

Mixed-criticality systems (MCSs)

These domains are no longer solely hosting isolated safety-critical tasks. Instead, they execute various tasks with different criticalities, where the criticality of a task is determined

■ **REAL-TIME SYSTEMS ARE** those systems whose proper behavior depends not only on their functionality but also on their response time. Until recently, real-time systems have been limited to safety-critical domains such as avionics and spacecrafts. However, with the emanating cyber-physical systems (CPS) and Internet-of-Things (IoT) revolution, real-time systems are becoming ubiquitous in many emerging domains. Examples include transportation such as smart vehicles, infrastructures such as power grids, healthcare such as implantable devices, and industrial environment such as robots. These domains pose two new major aspects that did not traditionally exist in real-time systems: the mixed-criticality nature of its software applications and the multiple-processor SoC (MPSoC) architecture of its hardware components.

Based on the consequences of the failure to meet its requirements. For instance Figure 1 illustrates a subset of the tasks embedded on a modern vehicle. Tasks such as the antilock braking system (ABS), the steering, and the engine control units are of high-criticality. Meeting the timing requirements of those tasks [historically known as hard real-time (HRT) tasks] is a life-safety condition. Other tasks such as the infotainment system and the connectivity box (such as internet, radio, WiFi, etc.) are of low criticality in the sense that they do not require strict timing guarantees. Instead, their proper functionality requires a high average-case performance. A third class of tasks contains tasks with medium criticality, known as soft real-time (SRT) tasks, such as the navigation system and the instrument cluster in a vehicle. They require a predictable execution time, which is not as strict as higher critical tasks, as well as a reasonable average-case performance. The number of criticality levels is domain specific and is not limited to three. For instance, the DO-178C avionics standard defines five levels of

Digital Object Identifier 10.1109/MDAT.2017.2771447

Date of publication: 8 November 2017; date of current version: 13 July 2018.

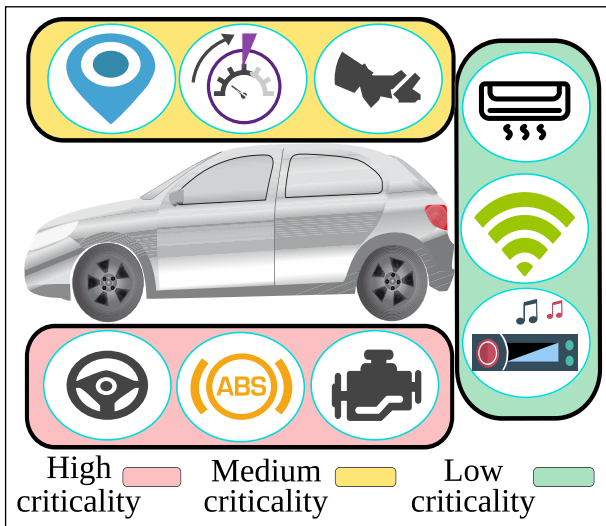


Figure 1. Examples of tasks running on a modern vehicle.

assurance, whereas the ISO-26262 defines four automotive safety integrity levels.

MPSoCs

MPSoCs are appealing platforms for emerging MCS domains primarily due to the benefits they provide in cost, area, power consumption, and performance compared to traditional computing systems. In addition, heterogeneous MPSoCs allow customized solutions to increase these benefits. The main intuition is that designing a single processing element (PE) to meet conflicting requirements of MCS tasks is inefficient due to the limited cost, area,

and battery budgets of MCS. Contrarily, designing custom PEs toward meeting those requirements has already proved its efficaciousness in current SoCs. Examples include specialized digital signal processors (DSPs), cipher, and multimedia PEs. In fact, one of the early motives of evolving MPSoCs was the real-time, low area, and low-power demands from embedded systems [1]. The envision for MCS is that a task with a particular criticality can be scheduled in an expedient core with the appropriate level of hardware predictability. Figure 2 delineates an example of such heterogeneous MPSoC architecture. In the near future, MPSoCs are expected to be used in all embedded system domains [2], [3]. To make this a reality, researchers made sincere efforts to provide MPSoCs tailored for safety-critical tasks (e.g., [2]–[4]). Companies started to develop MPSoCs that include dictated real-time processing units such as the Zynq UltraScale+MPSoC [5] from Xilinx, and the heterogeneous Nona-Core SoC from Renesas [6]. Safety standards are also slowly shifting toward considering multiple PEs. For instance, the AUTOSAR standard from the automotive industry released a guide to deploy software tasks onto multi-core architectures in a recent revision [7].

These two aspects together (MCS and MPSoCs) of emerging embedded systems bring out a number of challenges that has to be carefully repelled. The focus of this paper is to highlight those challenges, the proposed solutions in literature to address them, and the open issues yet to be addressed. We limit our discussion to four aspects of MCS: theoretical modeling, timing interference, data sharing, and security.

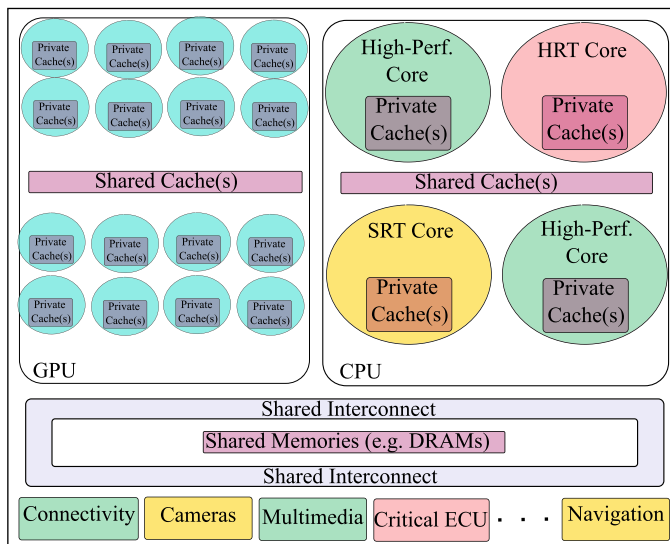


Figure 2. Example of a heterogeneous MPSoC platform.

MCS model

Current model

As identified by Vestal [8], the MCS model differs from the traditional real-time task model because of the uncertainty in considered worst-case execution time (WCET). Basically, the computed WCET of a task is an estimate calculated using extensive experimental testing and/or static analysis methods. Hence, based on the accuracy and pessimism levels of these methods, different estimates may exist (Figure 3). The higher the criticality of a task is, the more pessimistic its WCET estimates are. This observation resulted in representing the WCET as a function in the criticality level, $C(I)$. The majority of MCS papers consider a model of only two CLs, LO and HI [9]. Each task has $C(LO)$ and $C(HI)$, where

$C(LO) < C(HI)$. The system operates initially in a normal mode, where it considers the $C(LO)$ of each task and both higher and lower critical tasks utilize the hardware resources. Runtime techniques are used to monitor execution times of running tasks. If a higher critical task exceeds its $C(LO)$, the system switches to a degraded mode, where it suspends all lower critical tasks and considers the $C(HI)$ of the higher critical ones. This dynamic migration between various modes is a key characteristic of MCS as compared to single-criticality systems.

Issues with the current model

There exist a number of issues with approaches adopting this simplified widely considered model. Figure 4 highlights the three issues we believe are of most importance in MPSoCs.

Suspension

Upon switching to the degraded mode, no guarantees are given to lower critical tasks. The dual-criticality model deems lower-critical as noncritical; hence, there are no consequences of suspending them. Nevertheless, in systems with multiple CLs such as in the ISO-26262 standard, which has four ASILs A (lowest) to D (highest), suspension of tasks of ASIL A may be acceptable, while suspension of tasks of ASIL C may be prohibitively unacceptable solution as it may result in safety issues. Changes to the model have been proposed to provide certain guarantees to the lower critical tasks either by assigning different WCETs [10] or different periods [11] at different modes. An alternative approach is followed in [12], where instead of directly switching the mode and suspending lower critical tasks, the memory service guarantees of those tasks are degraded to reduce the interference on the higher critical tasks to accommodate for the increase in the execution time. These approaches consider a system-wide mode switch, where all the system components and tasks migrate to the new mode.

MPSoC Reconsiderations. In an MPSoC, there may be no need to deploy such full-system mode migration. Assume a scenario where a task, τ_i running on the medium-criticality (SRT) core in Figure 2 such that it exceeds its depicted C for the current mode, say because of a soft fault or a temperature increase in the SRT core. There exist opportunities to keep other noninterfering cores running the same set of tasks (i.e., no effective mode switching), while switching only the

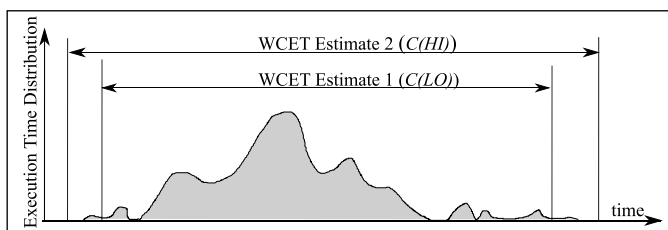


Figure 3. Different WCET estimates.

necessary core(s). Other techniques can be migrating tasks to “more-predictable” cores to avoid more switching at all. For the exemplified scenario, tasks running on that particular SRT-core can be migrated to an HRT-core (if possible) upon the monitored increase in τ_i 's C . So, a set of runtime decisions now exist, thanks to the heterogeneity nature of MPSoCs. Such MPSoC-related opportunities are yet to be explored.

Number of CLs and sources of uncertainty

Restricting the model to only two criticalities is not sufficient to meet industry standards, which define up to five levels as previously mentioned. It may seem that extending approaches that

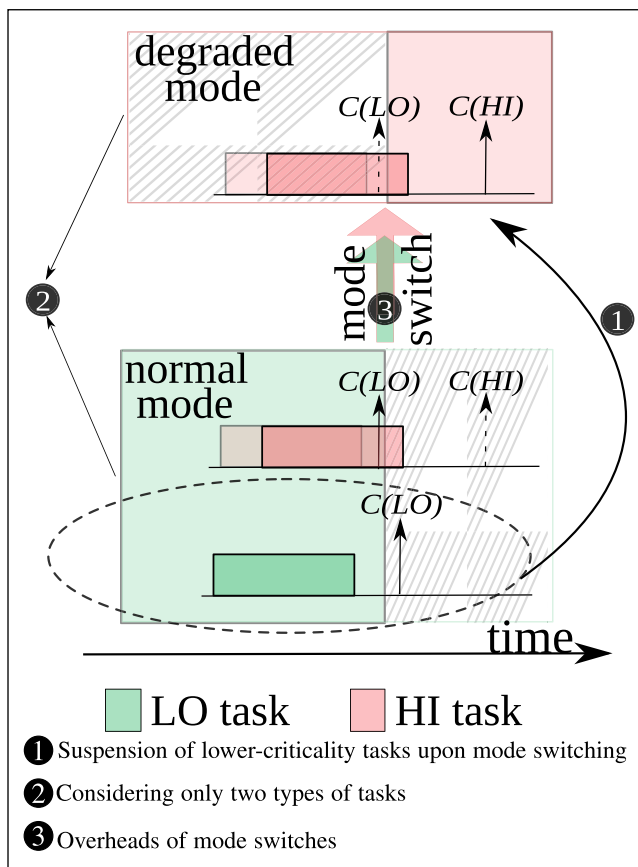


Figure 4. Current MCS model and its issues.

consider this model to more than two criticalities is straightforward. However, in most cases it is not. For instance, the suspension issue discussed in the first point is an outcome of a dual-criticality model, in which lower critical tasks are deemed noncritical. For systems with multiple levels, different approaches than suspension have to be considered.

MPSoC Reconsiderations. The heterogeneous nature of MPSoCs has a direct effect on the number of CLs. The standard model assumes that the uncertainty in WCET does not come from the system itself; rather, it comes from our inability to measure (or compute) it with complete confidence [9]. Although the latter part of this assumption still holds for MPSoCs, the former does not. Yes, the WCET estimate in MPSoCs is still a function of our confidence level of the used tools. However, we argue that the architecture of MPSoCs originates uncertainties as well. In traditional single-core or symmetric multiple-processor (SMP) architectures, where core (or cores) executing a task does not affect its measured execution time. However, in a heterogeneous MPSoC (e.g., in Figure 2), the decision of which cores are used to execute a task directly affects the level of certainty in its WCET. For instance, an HRT core is usually simple in terms of micro-architecture with almost no implemented architectural optimizations. This is necessary to allow for high level of analyzability, which leads to tight WCETs for safety-critical tasks. Contrarily, a high-performance core usually deploys speculative optimizations such as out-of-order execution and branch prediction. As a result, the confidence level in a task's WCET when it runs on an HRT core definitely differs from the case when the same task runs on a high-performance core. The interdependency dilemma that requires investigation here is that the WCET estimates become a function in the task-to-core mapping, which is part of the scheduling algorithm that relies on these estimates as inputs.

Overheads

Monitoring tasks and switching between running modes engross high overheads. However, to simplify the scheduling problem of MCS, most approaches ignore these overheads. Although this may be a theoretically acceptable assumption as these overheads are implementation related, a practical adoption of these approaches in industry mandates careful quantification of these overheads. Recently, a few efforts have been proposed to bridge this gap [13], [14].

While the former [13] focuses on single-core, the latter [14] evaluates multicore platforms. Both efforts consider the implementation of a subset of proposed scheduling mechanisms; thus, more studies are required to identify implementation-related issues of theory-based novel MCS scheduling techniques.

MPSoC Reconsiderations. Deployment of MCS onto MPSoCs requires addressing the scalability challenges associated with these scheduling and monitoring techniques. Furthermore, mode switching in MPSoCs may incur task migrations or reassignment of heterogeneous cores to tasks; thus, the effects of these decisions on the switching overhead need to be quantified. On the other hand, MPSoCs open the door for customized solutions. For instance, dedicating a PE for the runtime monitoring possibly helps in faster detection of exceptional events, therefore enabling the system to react in a timely manner. The architecture of this PE can be further tailored to optimize the behavior of the monitoring techniques. Specialized PEs are widely used by current MPSoCs, which usually dictate PEs for security, connectivity, data signal processing, and other tasks.

Shared resources: Timing interference

In MPSoCs, different PEs in the system interfere with each other, while competing to access memory resources that are shared amongst them. As shown in Figure 2, these shared resources include interconnects, on-chip caches, and off-chip dynamic random access memories (DRAMs). This interference is a challenge for real-time systems because operations of one core affect the temporal behavior of other cores, which complicates the timing analysis of the system.

Since the aforementioned MCS model originally evolved for single-core systems, most of the proposed approaches adopting it do not incorporate these interferences in their scheduling or analysis [15]. Experiments show that memory interferences can contribute up to 300% to the WCET [16], while the memory bus interference can solely increase the WCET up to 44% [17]. Consequently, it is of unavoidable necessity to account for these interferences for MPSoC MCS. There exist proposals to address this interference in multicore MCS at the interconnect (see [12], [15]), the shared cache (see [18], [19]), and the shared DRAM (see [20]). However, most of these approaches consider SMP architectures and do not account for the heterogeneity of MPSoCs.

MPSoC Reconsiderations. Bounding the timing interference in MPSoCs is a burdensome goal for many reasons.

- The interference exaggerates with the increase in the number of PEs competing on the shared resources.
- Each type of PEs has its own memory access behavior, which complicates the analysis, thus leading to more pessimism. For instance, data-intensive PEs such as multimedia and DSP processors can saturate system queues by their memory requests if not carefully arbitrated. Thus, unlike most of the current approaches, a requirement- and criticality-aware arbitration is a must to deliver differential service to PEs.
- Understanding the architectural details of shared resources (such as the interconnect and the memory hierarchy) is inevitable to derive realistic bounds.

On the other hand, MPSoCs provide unique opportunities that do not exist in SMPs. Efforts to investigate these opportunities will facilitate the deployment of MCS onto MPSoCs. Examples of research directions are as follows.

- Which memory levels should be shared amongst which cores to meet various requirements of MCS. For instance, does the GPU share the last-level cache with the CPU?
- How to distribute the cache architecture? Would implementing a nonuniform cache architecture (NUCA), which is a norm in manycore systems, be an adequate approach for MCS (e.g., helping in achieving different levels of isolation)?
- Usually, MPSoCs integrate different types of on-chip memories such as hardware-managed caches and software-managed scratch-pad memories (SPMs). Most of the currently available approaches focus on a single type. Similarly, MPSoCs support different types of available off-chip memories such as double data rate (DDR), graphics DDR (GDDR), and low-power DDR (LPDDR). Investigating the cooperation of these types is also worth investigating.

Shared resources: Shared data

One of the most challenging burdens for computer architects is to maintain correctness of shared data

stored in memory hierarchies of multiple-PEs platforms, which is known as *cache coherence*. Although cache coherence has been extensively investigated for conventional performance-oriented platforms, embedded systems introduce new challenges from the predictability perspective. For instance, empirical studies show that the data interference and the coherence effects can make the parallel execution of an application 3.87× slower than its sequential execution [21], while the worst-case coherence latency exhibits quadratic growth with increasing number of PEs [22]. Real-time community has introduced various solutions to address this problem, which we categorize into three approaches identifying the applicability of each approach to MPSoCs.

Prevention

This approach avoids the problems resulting from data sharing by completely disallowing it through enforcing complete isolation between tasks. At the shared cache, mechanisms such as strict cache partitioning and coloring are used [23]. At the DRAM level, bank privatization is utilized to uniquely map tasks to different banks. Isolation is an attractive solution as it simplifies the analysis and minimizes timing interference, while ensuring data correctness. However, data isolation suffers from three limitations: 1) it adopts the independent-task model, thus disabling any communication amongst tasks; 2) it may result in a poor memory or cache utilization (For instance, a task can keep evicting its cache lines if it reaches the maximum of its partition size, while other partitions may remain underutilized.); and 3) it does not scale with increasing number of cores. For example, the number of cores in the system has to be less than or equal to the number of DRAM banks to be able to achieve isolation at DRAM.

MPSoC Reconsiderations. It might be acceptable for traditional real-time task models to assume isolation in order to achieve uniprocessor-equivalence [24] in multiprocessor platforms, thus allowing the reuse of maturely developed scheduling approaches for uniprocessors. However, with the emerging technologies that continuously adopt new functionalities, complete isolation seems to be a prohibitively costly solution. Considering the automotive applications in Figure 1, they utilize data collected by various sensors to conduct the appropriate act. This data is usually shared and used by applications with

different criticalities. For instance, the brake sensors are utilized by both the high-critical ABS and the driver assistance and cruise control tasks which are of medium-criticality [25]. In addition, with the massive concurrency of MPSoCs, solutions preventing simultaneous running of dependent (i.e., data sharing) tasks are becoming evidently ill-suited as they diminish the performance gains of MPSoCs due to the aforementioned three limitations. Accordingly, researchers recognized that in order to have any practical impact, scheduling techniques must permit data sharing [22], [26] and the following two approaches are recently proposed to solve data sharing problems without enforcing isolation.

Shared-data aware scheduler

This approach combines operating system techniques, profiling, and hardware performance counters to envision the effects of data sharing. Accordingly, the scheduler is constructed such that it minimizes those effects [21], [26]. For example, if two tasks share data, a simple solution is to schedule them such that they do not simultaneously run on different PEs. This can be achieved either by postponing the execution of one of them, or mapping them to the same PE [26].

MPSoC Reconsiderations. These scheduling-based solutions are promising since they address the shared data problem, while not enforcing isolation. However, some of these solutions have limited applicability to MPSoCs. For example, given the high level of parallelism in MPSoCs, mapping dependent tasks to same core may not be a viable solution as it limits performance gains. Similarly, collecting runtime readings from performance counters may not be cost effective in terms of overheads given the large number and heterogeneity of PEs in MPSoCs.

Predictable hardware cache coherence

A recent work [22] manages shared data in real-time systems through deploying a hardware cache coherence protocol. Identifying the sources of unpredictability due to coherence interference, this paper promotes certain invariants to maintain toward allowing simultaneous and predictable accesses to shared data. These invariants are satisfied by augmenting the classic modify-share-invalidate (MSI) protocol with transient coherence states and minimal architectural changes. The advantage of

this approach is that programmers do not need to explicitly manage coherence of shared data in the application. In addition, it does not require any modifications to existing scheduling algorithms.

MPSoC Reconsiderations. This approach does not address coherence in MCS. In addition, this approach, along with all the discussed approaches, considers SMPs. Effects of heterogeneity on these approaches are not investigated yet. For example, how coherence operates across different types of PEs? One coherence protocol might not fit all types and allowing for different coherence protocols can be a better solution for MCS if the interaction between these protocols is carefully designed.

Security of MPSoC MCS

Security is one of the biggest challenges encountering researchers and engineers of CPS. The more ubiquitous the CPS become, the more concerning their security is. To exemplify, various vulnerabilities have been reported in industrial supervisory control and data acquisition (SCADA) systems [27] and smart vehicles [28], [29]. Researchers also managed to reverse engineer architectural details in embedded platforms by running hand-crafted C-programs [30].

MPSoC Reconsiderations. Three aspects make the development of secure MPSoC MCS a burden challenge.

Cyber-physical nature of MCS

MCS in many emerging domains interact with the physical world, which makes them CPS. Embedded components in these CPS manage sensitive tasks; therefore, any security breach could lead to catastrophic consequences. These consequences range from revealing personal information (e.g., from wearable devices) to a global threat (e.g., compromising a nuclear plant). Consequently, ensuring the security of these systems is a first-class mission. In addition, the interaction with the physical world allows for threats that did not exist in traditional computing systems. For example, researchers successfully managed to gain access to locked cars by only eavesdropping a single signal from the original remote keyless entry unit of the car [28].

Heterogeneity of MPSoCs

On the one hand, each PE of MPSoC has different characteristics and can even be an intellectual

Table 1. Opportunities and challenges.

Aspect	Challenges	Opportunities/Research directions
Theoretical Model	<ol style="list-style-type: none"> 1. WCET becomes a function of task-to-core mapping 2. Scalability of monitoring techniques and switching overheads 3. Effects of heterogeneity on task migration upon switching 	<ol style="list-style-type: none"> 1. Spatial/partial instead of system-wide mode switching 2. Alternatives to mode switch such as migrating tasks to more-predictable cores 3. Customizable solutions: e.g. dedicating specialised PE for execution-time monitoring
Timing Interference	<ol style="list-style-type: none"> 1. Large number of PEs 2. PEs have different memory behaviors 3. Hard to derive tight bounds 	<ol style="list-style-type: none"> 1. Which levels should be shared amongst which PEs for MCS? 2. Distribution of cache architecture. e.g. uniform or NUCA? 3. Different types of on- (caches and SPMs) and off-chip memories (DDR, GDDR, LPDDR)
Data Sharing	<ol style="list-style-type: none"> 1. massive parallelism deems isolation ill-suited 2. Different memory patterns complicates the analysis 3. Scheduling-based approaches are hard to adopt 	<ol style="list-style-type: none"> 1. Different PEs have different transaction sizes and BW requirements 2. Different memory types with different features (high bandwidth vs. low latency) 3. Possibility for different approaches for each PE type
Security	<ol style="list-style-type: none"> 1. CPS entails possible catastrophic threats 2. New heterogeneity-exploiting threats 3. New vulnerabilities because of Shared components between different criticalities 	<ol style="list-style-type: none"> 1. Identifying MPSoCs specific vulnerabilities 2. Developing cost- and performance-effective methodologies to face threats 3. Adopting security as a first-class citizen in designing MPSoCs for MCS

proprietary of a third-party entity. Accordingly, the security problems of each of these PEs are inherited. On the other hand, these PEs share system components and interact with each other. This opens the door for new across-PEs threats. Accordingly, threats and vulnerabilities in MPSoCs are harder to analyze, detect, and assess compared to traditional systems. To exemplify, the well-known Stuxnet attack exploited the authentication of the Siemens programmable logic controller to access a Windows machine [27].

Shared hardware components in MPSoCs

Historically, security was not considered as a concern for MCS because isolation was a design aspect of these systems, where each task (or group of tasks with same criticality) is running on a PE (or a partition of PEs) that is completely isolated from other PEs (or partitions). As a consequence, sensitive tasks that require high levels of security are isolated from nonsecure tasks. However, in MPSoCs this is not the case. Different PEs, and hence tasks, share hardware components and isolation is considered a costly solution as previously explained. Again, this creates new potential threats. To exemplify, researchers were able to control sensitive (considered secure) electronic control units (ECUs) such as the engine control in a Jeep Cherokee car by compromising the (considered insecure) radio unit because the radio unit shares the controller area network (CAN) with these ECUs [29]. To address these challenges, three research directions are necessary

toward secure deployment of MPSoC MCS: 1) identifying new vulnerabilities of MPSoCs, which did not exist in traditional platforms; 2) developing cost- and performance-effective methodologies to prevent or mitigate them; and 3) adopting security as a first-class citizen in designing MPSoCs for MCS (secure-by design concept).

WE ARGUE THAT MPSoCs will be soon the dominating platform for emerging embedded systems domains. Despite the tremendous benefits and opportunities they provide, certain challenges have to be addressed and new research directions need to be explored. Four aspects are of great importance upon deploying MCS on MPSoCs: theoretical modeling, timing interference, data sharing, and security. For each of these aspects, Table 1 highlights both the remarkable challenges and the opportunities that MPSoCs uniquely create. These challenges and opportunities can be back traced to three characteristics of MPSoCs: 1) large number of PEs; 2) heterogeneity of PEs; and 3) different types of shared resources amongst PEs. We believe that seizing these opportunities and addressing associated challenges will enable enormous advances for MCS applications. ■

References

- [1] A. Jerraya and W. Wolf, *Multiprocessor Systems-On-Chips*. New York, NY, USA: Elsevier, 2004.
- [2] P. Axer, M. Sebastian, and R. Ernst, "Reliability analysis for MPSoCs with mixed-critical, hard

- real-time constraints,” in *Proc. IEEE/ACM/IFIP Int. Conf. Hardware/Softw. Codesign Syst. Synthesis (CODES+ISSS)*, Taipei, Taiwan, Oct. 2011, pp. 149–158.
- [3] C. El Salloum, M. Elshuber, O. Höftberger, H. Isakovic, and A. Wasicek, “The across MPSoC—A new generation of multi-core processors designed for safety-critical embedded systems,” *Microprocessors Microsyst.*, vol. 37, no. 8, pp. 1020–1032, 2013.
- [4] R. Pellizzoni, P. Meredith, M.-Y. Nam, M. Sun, M. Caccamo, and L. Sha, “Handling mixed-criticality in SoC-based real-time embedded systems,” in *Proc. 7th ACM Int. Conf. Embed. Softw. (EMSOFT)*, Grenoble, France, Oct. 2009, pp. 235–244.
- [5] V. Boppana, S. Ahmad, I. Ganusov, V. Kathail, V. Rajagopalan, and R. Wittig, “Ultrascale+ MPSoC and FPGA families,” in *Proc. IEEE Hot Chips Symp. (HCS)*, Cupertino, CA, USA, Aug. 2015, pp. 1–37.
- [6] C. Takahashi et al., “4.5 a 16nm FinFET heterogeneous nona-core SoC complying with ISO26262 ASIL-B: Achieving 10⁻⁷ random hardware failures per hour reliability,” in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, San Francisco, CA, USA, 2016, pp. 80–81.
- [7] AUTOSAR, “A guide to multi-core systems.” [Online]. Available: http://www.autosar.org/fileadmin/files/standards/classic/4-1/software-architecture/general/auxiliary/AUTOSAR_EXP_MultiCoreGuide.pdf
- [8] S. Vestal, “Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance,” in *Proc. IEEE Int. Real-Time Syst. Symp. (RTSS)*, Porto, Portugal, 2007, pp. 239–243.
- [9] A. Burns and R. Davis, *Mixed Criticality Systems—A Review*, Dept. Comput. Sci., Univ. York, York, U.K., Tech. Rep., 2013.
- [10] A. Burns and S. Baruah, “Towards a more practical model for mixed criticality systems,” in *Proc. Workshop Mixed-Criticality Syst. (WMCS)*, Waco, TX, USA, Apr. 2013, pp. 1–6.
- [11] H. Su and D. Zhu, “An elastic mixed-criticality task model and its scheduling algorithm,” in *Proc. IEEE Design Autom. Test Eur. Conf. (DATE)*, Grenoble, France, Mar. 2013, pp. 147–152.
- [12] M. Hassan and H. Patel, “Criticality-and requirement-aware bus arbitration for multi-core mixed criticality systems,” in *Proc. IEEE Real-Time Embed. Technol. Appl. Symp. (RTAS)*, Vienna, Austria, Apr. 2016, pp. 1–11.
- [13] H.-M. Huang, C. Gill, and C. Lu, “Implementation and evaluation of mixed-criticality scheduling approaches for sporadic tasks,” *ACM Trans. Embed. Comput. Syst. (TECS)*, vol. 13, no. 4s, p. 126, 2014.
- [14] L. Sigrist, G. Giannopoulou, P. Huang, A. Gomez, and L. Thiele, “Mixed-criticality runtime mechanisms and evaluation on multicores,” in *Proc. IEEE Real-Time Embed. Technol. Appl. Symp. (RTAS)*, Seattle, WA, USA, Apr. 2015, pp. 194–206.
- [15] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele, “Scheduling of mixed-criticality applications on resource-sharing multicore systems,” in *Proc. IEEE Int. Conf. Embed. Softw. (EMSOFT)*, Montreal, QC, Canada, Sep. 2013, pp. 1–15.
- [16] R. Pellizzoni, A. Schranzhofer, J.-J. Chen, M. Caccamo, and L. Thiele, “Worst case delay analysis for memory interference in multicore systems,” in *Proc. IEEE Design Autom. Test Eur. Conf. (DATE)*, Dresden, Germany, Apr. 2010, pp. 741–746.
- [17] R. Pellizzoni, B. D. Bui, M. Caccamo, and L. Sha, “Coscheduling of CPU and I/O transactions in cots-based embedded systems,” in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Barcelona, Spain, Nov. 2008, pp. 221–231.
- [18] M. Chisholm, B. C. Ward, N. Kim, and J. H. Anderson, “Cache sharing and isolation tradeoffs in multicore mixed-criticality systems,” in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, San Antonio, TX, USA, Dec. 2015, pp. 305–316.
- [19] B. Lesage, I. Puaut, and A. Seznec, “Preti: Partitioned real-time shared cache for mixed-criticality real-time systems,” in *Proc. ACM Int. Conf. Real-Time Netw. Syst. (RTNS)*, Pont à Mousson, France, Nov. 2012, pp. 171–180.
- [20] M. Hassan, H. Patel, and R. Pellizzoni, “A framework for scheduling DRAM accesses for multi-core mixed-time critical systems,” in *Proc. IEEE Real-Time Embed. Technol. Appl. Symp. (RTAS)*, Seattle, WA, USA, Apr. 2015, pp. 307–316.
- [21] G. Gracioli and A. A. Fröhlich, “On the design and evaluation of a real-time operating system for cache-coherent multicore architectures,” *ACM SIGOPS Oper. Sys. Rev.*, vol. 49, no. 2, pp. 2–16, 2016.
- [22] M. Hassan, A. M. Kaushik, and H. Patel, “Predictable cache coherence for multi-core real-time systems,” in *Proc. IEEE Real-Time Embed. Technol. Appl. Symp. (RTAS)*, Pittsburgh, PA, USA, Apr. 2017, pp. 235–246.
- [23] G. Gracioli, A. Alhammad, R. Mancuso, A. A. Fröhlich, and R. Pellizzoni, “A survey on cache management mechanisms for real-time embedded systems,” *ACM Comput. Surv. (CSUR)*, vol. 48, no. 2, p. 32, 2015.

- [24] R. Mancuso, R. Pellizzoni, M. Caccamo, L. Sha, and H. Yun, "WCET (m) estimation in multi-core systems using single core equivalence," in *Proc. 27th IEEE Euromicro Conf. Real-Time Syst. (ECRTS)*, 2015, Lund, Sweden, Jul. 2015, pp. 174–183.
- [25] R. Ernst and M. Di Natale, "Mixed criticality systems—A history of misconceptions?" *IEEE Design Test*, vol. 33, no. 5, pp. 65–74, 2016.
- [26] M. Chisholm, N. Kim, B. C. Ward, N. Otterness, J. H. Anderson, and F. D. Smith, "Reconciling the tension between hardware isolation and data sharing in mixed-criticality, multicore systems," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Porto, Portugal, 2016, pp. 57–68.
- [27] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security Privacy*, vol. 9, no. 3, pp. 49–51, 2011.
- [28] F. D. Garcia, D. Oswald, T. Kasper, and P. Pavlidès, "Lock it and still lose it-on the (in) security of automotive remote keyless entry systems," in *Proc. 25th USENIX Security Symp. (USENIX Security 16)*, Austin, TX, USA, Aug. 2016.
- [29] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, Las Vegas, NV, USA, vol. 2015, 2015.
- [30] M. Hassan, A. M. Kaushik, and H. Patel, "Reverse-engineering embedded memory controllers through latency-based analysis," in *Proc. IEEE Real-Time Embed. Technol. Appl. Symp. (RTAS)*, Seattle, WA, USA, Apr. 2015, pp. 297–306.

Mohamed Hassan is an SoC Research and Development Engineer at Intel Corporation, Toronto, ON, Canada. His current research interests include real-time embedded systems, multicore architectures, hardware validation, and security. He has a PhD from the University of Waterloo, Waterloo, ON, Canada (2017).

■ Direct questions and comments about this article to Mohamed Hassan, Intel Corporation, Toronto, ON M1K 3S5, Canada; e-mail: mohamed.hassan@ieee.org.