AURIX TC277 Multicore Contention Model Integration for Automotive Applications

Enrico Mezzetti*, Luca Barbina†, Jaume Abella*, Stefania Botta†, Francisco J. Cazorla*
* Barcelona Supercomputing Center (BSC), Spain
† Magneti Marelli S.p.A., Italy

***Abstract*—Embedded systems industry needs reliable and tight worst-case execution time (WCET) estimates for critical applications running on multicores, as a prerequisite to their adoption. While industry already uses reliable tools for single-core WCET estimation and several multicore contention models (MCMs) have been proposed, their combination have not been shown to be fully compatible with the automotive industrial practice yet. This paper reduces this gap by presenting a framework for the integration of MCMs into industrial WCET estimation practice. We illustrate such integration for a Magneti Marelli powertrain control unit on an Infineon AURIX TC277 multicore platform.**

I. Introduction

Multicore microcontrollers, such as the Infineon AURIX TC27x family [1], are increasingly adopted in the automotive domain to respond to the high-performance needs of Automotive Driving Assistance Systems (ADAS) and connected cars, among others. Verification and validation (V&V) of automotive systems against ISO26262 functional safety standard [2] requires proving that the risk of failing to meet their timing constraints for critical real-time tasks is residual. In the case of multicores, contention effects on the access to hardware shared resources cause that co-running tasks affect each others' timing behavior, and hamper deriving tests scenarios in which worst-case contention effects are properly captured.

Recently, several approaches have been proposed to capture contention effects in commercial off-the-shelf (COTS) multicores by means of measurement-based approaches, thus fitting automotive timing analysis practice. Those approaches, that target multicores such as the NXP P4080 [3], [4], the Cobham Gaisler LEON4 [5], and the Infineon AURIX TC27x processor family [6], however, have not addressed in detail their integration with real industrial use cases.

In this work, we tackle the challenge of integrating a measurement-based multicore contention model (MCM) on an industrial application as a vehicle to illustrate how integration can be successfully achieved. In particular, we consider the integration of the Infineon AURIX TC27x MCM [6] on a powertrain automotive use case of Magneti Marelli. The integration approach presented in this paper has allowed the end user integrating the MCM on the engine control and transmission control applications (part of the powertrain control unit) on its premises, without external intervention. Results show that tight and reliable contention bounds can be obtained for AURIX TC277 platforms; and the quantitative evidence obtained allows scheduling applications efficiently and going through the certification process.

II. Integration Approach

Target Platform. The AURIX™ TC277 [1] (see Figure 1) equips three TriCore™ cores: a low-power (1.6E) and two high-performance cores (1.6P). All cores have their own code scratchpad (PSPR), data scratchpad (DSPR), code cache (ICache) and data cache (DCache for 1.6P and DRB for 1.6E). Cores access the memory system through a crossbar (named SRI). The memory system includes a SRAM device, interfaced with the Local Memory Unit (LMU) and a FLASH device,

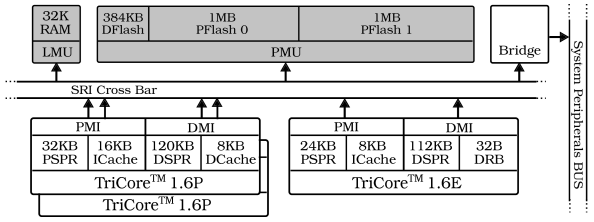

Fig. 1. Block Diagram of the AURIX™ TC-27x

interfaced with the Program Memory Unit (PMU), which includes 3 regions, one for data (DFlash) and two for code (PFlash0 and PFlash1). Segments in both the PMU and LMU can be regarded as cacheable or uncacheable. Finally, applications are configured so that their data, stack and functions are mapped explicitly to specific devices, either shared memories or local scratchpads, together with their cacheability options.

Target MCM. MCMs builds on a timing estimate of the execution time of the task (τ_a) in isolation (C_a^{isol}), i.e. without contention, to derive a multicore estimate of τ_a's execution time (C_a^{muc}). To that end, the MCMs bound the contention τ_a's requests can suffer in the access to the hardware shared resources, Δ_a^{cont} so that $C_a^{muc} = C_i^{isol} + \Delta_a^{cont}$.

Let us assume a multicore with a single shared resource that accepts a single type of request. Further assume a task under analysis τ_a and one contender task τ_b. In this simplified scenario, and building on the time predictable round-robin arbitration in the AURIX, the worst-case contention τ_b can cause on τ_a, i.e. $\Delta_{b\to a}^{cont}$, is shown in Equation 1, where n_b is the number of τ_b requests to target shared resource and l is the latency of those requests.

$$\Delta_{b\to a}^{cont} = min(n_a, n_b) \times l \tag{1}$$

In the more general (and complex) case, several resources exist, accepting requests with different latencies. In the AURIX, the target resource of each request is a configuration dependent subset of (*ram*,*dflash*,*pflash0*,*pflash1*). Note that since the crossbar supports parallel transactions on different interfaces, contention may happen only between requests targeting the same interface. Furthermore existing monitors might not capture the exact access latencies and access count to each of the shared resources in the AURIX. To capture these complexities, we build on the ILP (Integer Linear Programming) MCM presented in [6]. Furthermore, the model in Equation 1 builds on two sets of parameters:

- *Platform dependent* parameters capture the worst contention an access to a shared resource can suffer due to other accesses to any shared resource (l). Those values need to be derived just once for each platform, so they are already available along with the MCM.
- *Application dependent* parameters capture application characteristics – mostly related to the number of accesses to each hardware shared resource (n_a) – needed to leverage the amount of contention that the application can experience. Optionally, the model admits application dependent parameters also for applications running simultaneously to leverage the actual maximum contention

Algorithm 1 Excerpt of the integration code framework

```
1: start_profile_N();
2: User code
3: stop_profile();
4: write_profile((word *) LOCAL_DSPR);
5: signal(LOCAL_DSPR);
6: wait(CPU0_DSPR);
```

that can be experienced with such *contender* rather than the maximum contention against *any* contender.

Deriving application-dependent parameters requires the instrumentation of the application to collect information with the Debug Support Unit (DSU). In particular, the events monitored from the application are: `CCNT` (cycle count), `ICNT` (instruction count), `PCACHE_HIT` (program cache hit), `PCACHE_MISS` (program cache miss), `DCACHE_HIT` (data cache hit), `DCACHE_MISS_CLEAN` (data cache miss clean), `DCACHE_MISS_DIRTY` (data cache miss dirty), `PMEM_STALL` (stall cycles for the program interface), and `DMEM_STALL` (stall cycles for the data interface). Further details can be found in the AURIX TC27x documentation.

Practical Integration Framework. The integration has been performed with a flexible approach providing users with a skeleton where they can embed the analyzed task, either inlining the code or placing a function call. An excerpt of this code is shown in Algorithm 1, and the location for the integration of users' code is line 2. Since the number of events to be monitored (the 9 events described before) is larger than the number of Performance Monitoring Counters (PMCs) available (only 3), three rounds of measurements are needed. This requires replacing N in line 1 by 1, 2 or 3 for each measurement round so that the appropriate events are monitored. The call in line 4 dumps PMC readings to a specific memory region, which are extracted automatically with appropriate scripts. Hence, after the 3 sets of measurements, events needed by the multicore contention model are available.

The contention model has been implemented as a Python script to avoid portability issues. Its computational cost is negligible (largely below 1 second). The Python script operates the event counts to estimate the access counts to the shared hardware resources, and how much contention could be experienced (cycles). The script accepts either one input file with the PMC readings of the task being analyzed or several input files where the additional files include PMC readings for contender tasks (those that would be run simultaneously in the other cores). If only the first file is provided, the contention model estimates execution time bounds valid regardless of the software run on each other core. Instead, if files are provided for contenders, then tighter bounds can be obtained, but they are only valid if tasks running in the other cores are those contenders, thus trading off between tightness and flexibility.

III. Evaluation

We assess the proposed MCM against two functions extracted from the complete powertrain application. We did not perform the evaluation on an end-to-end automotive task, as it has been observed that the approach itself is particularly efficient (and naturally applied) at the level of software units, hence during unit-testing, rather than on run-time entities.

We selected two illustrative functions, F_A and F_B, run them on a high-performance core, and assess the contention they could experience due to tasks running in the other high-performance core. The location where code and data are mapped determines the memory they are mapped to and hence, the potential contention in the system. The analyzed functions characteristics are reported in Table I.

The same PMC collection process has been applied to both functions under analysis. Raw numbers are reported in Table II. As expected, F_A completely fits into the core scratchpads and does not generate crossbar traffic (except for 7 DMEM_STALL cycles caused by the measurement protocol). F_B instead fetches code and data from the Flash device and, despite the good cache usage, uses the crossbar, which in turn exposes F_B to inter-core contention.

TABLE I
Characterization of the automotive functions.

	Code	Data	Characterization
F_A	PSPR	DSPR	Function is small enough to fit in the local scratchpads (PSPR and DSPR). No activity is expected on the cross-bar and model should predict no contention
F_B	PFlash0	DSPR (Stack), PFlash1 (Constants)	Function accesses the cross-bar for fetching code and data. Both code and data are mapped to the PFlash, but on separate areas that are accessed from different interfaces.

TABLE II
PMC readings for F_A and F_B.

F_A					
CCNT	16361	PCACHE_MISS	0	DCACHE_HIT	0
ICNT	17538	PMEM_STALL	0	DCACHE_MISS_CLEAN	0
PCACHE_HIT	0	DMEM_STALL	7	DCACHE_MISS_DIRTY	0
F_B					
CCNT	20969	PCACHE_MISS	173	DCACHE_HIT	263
ICNT	23734	PMEM_STALL	1380	DCACHE_MISS_CLEAN	9
PCACHE_HIT	9614	DMEM_STALL	156	DCACHE_MISS_DIRTY	0

We feed the MCM with the PMC readings in Table II using the framework described in Section II. Results show that F_A is insensitive to contention, as expected. Instead, the integration of the MCM with F_B proves that its execution time can only grow up by 7.2% (1,504 cycles) due to multicore contention in the other high-performance core. Note that if execution time growth was too high, the MCM would allow tightening it by accounting for measurements from contenders.

Overall, the successful integration and evaluation of this methodology shows that (1) no roadblock is foreseen to integrate the methodology on industrial use cases; (2) multicore contention bounds can be applied at unit testing, thus enabling the application of the methodology in early design stages; (3) the method and results obtained provide evidence needed for certification; and (4) the application of this methodology can be carried out by end users on their own, thus providing them with independence to analyze their software.

Acknowledgements

The research leading to this work has received funding from the European Union's H2020 programme under grant agreement No 644080 (SAFURE), by the Spanish Ministry of Economy and Competitiveness (MINECO) under grant TIN2015-65316-P and the HiPEAC Network of Excellence. Jaume Abella and Enrico Mezzetti have been partially supported by MINECO under Ramon y Cajal and Juan de la Cierva-Incorporación postdoctoral fellowships number RYC-2013-14717 and IJCI-2016-27396 respectively.

References

[1] http://www.ehitex.de/application-kits/infineon/2531/aurix-application-kit-tc277-tft, *AURIX Application Kit TC277 TFT*.
[2] International Organization for Standardization, *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.
[3] J. Nowotsch and M. Paulitsch, "Leveraging multi-core computing architectures in avionics," in *EDCC*, 2012.
[4] J. Bin et al., "Studying co-running avionic real-time applications on multi-core COTS architectures," in *ERTS*^{2}, 2014.
[5] E. Diaz et al., "MC2: Multicore and Cache Analysis via Deterministic and Probabilistic Jitter Bounding," in *Ada-Europe 2017*, 2017.
[6] ——, "Modelling multicore contention on the AURIX TC27x," in *DAC*, 2018.