Experimental Analysis on the NXP's T2080 Cache Coherence: A Step Towards MPSoCs in Critical Systems

Roger Pujol*†, Mohamed Hassan§, Francisco J. Cazorla†‡,
*Universitat Politècnica de Catalunya, Barcelona, Spain
†Barcelona Supercomputing Center, Barcelona, Spain
‡Maspatechnologies S.L., Barcelona, Spain
§McMaster University, Hamilton, Canada
E-mail: {roger.pujol, francisco.cazorla}@bsc.es, mohamed.hassan@mcmaster.ca

Keywords—Cache Coherence, Critical Systems, MPSoC, Certification.

I. Extended Abstract

The adoption of complex MPSoCs in critical real-time embedded systems [1], [2] mandates a detailed analysis of their architecture to facilitate certification [3]. This analysis is hindered by the lack of a thorough understanding of the MPSoC system due to the unobvious and/or insufficiently documented behavior of some key hardware features [4], [5].

Confidence in those features can only be regained by building specific tests to both, assess whether their behavior matches specifications and unveil their behavior when it is not fully known a priori.

In this line, in this work we develop a thorough understanding of the cache coherence protocol in the avionics-relevant [6] NXP T2080 [1] architecture.

II. Setup

The NXP T2080 SoC [1] (see Figure 1), assessed for its use in avionics, features four PowerPC e6500 cores [7], each with its private instruction and data cache (IL1 and DL1, respectively) as well as a private Memory Management Unit (MMU). Each core communicates with the shared L2 cache via the cache-core interface (CCI). The L2 cache is shared between all the cores. A "CoreNet" coherence fabric (CCF) provides access to the DDR SDRAM memory controller as well as other interfaces present in the board like the Direct Memory Access (DMA). The L2 has a single port to the CCF whose access is controlled by the Bus Interface Unit (BIU). The L2 cache is the point of coherency in the cluster. DL1 caches contain no modified data as they are write-through. The L2 is inclusive of the DL1 of each core, so if a data line is evicted from the L2 cache, it is invalidated in the corresponding DL1 caches. The T2080 implements a 4-state Modified-Exclusive-Shared-Invalid (MESI) cache coherence protocol. The coherence granularity is a L2 cache line, such that each line has its own coherence state information.

DMA transfers can 'generate' snoop requests to the L2. This can lead to invalidations of data in the DL1 caches and/or the L2 since this data becomes not up to date with respect to the latest value written by the DMA to memory.

A. Observability

Hardware Monitors. Several hardware event monitors in the T2080 [7] provide information about the L2 cache coherence activity, which we show in Table I.

Debugger Support. Using CodeWarrior, the standard IDE for the T2080, we access several flags for each cache line with information about their coherence state: Dirty (i.e. Modified), Valid, Shared, and Exclusive.

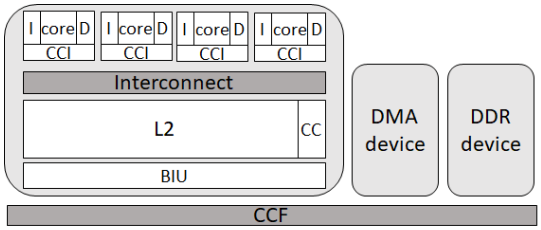

Fig. 1. Simplified block diagram of the T2080

TABLE I. Coherent related event counters in for the L2

Counter	Description
L2DA	Number of L2 data accesses.
L2DM	Number of L2 data misses.
L2SH	Number of L2 cache snoop hits.
L2SP	Number of L2 cache snoop pushes.
ESR	Externally generated snoop requests.
L2SM	L2 snoops causing MINT (Modified INTervention).
L2SS	L2 snoops causing SINT (Shared INTervention).
L2RC	Number of L2 reloads from CoreNet.
BL	BLINK requests from L2 to core (e.g. back invalidates)
CL	CLINK requests from L2 to core(CoreNet data forwarding)

B. Experimental Setup

We seek to observe the behavior of the L2 cache when transitioning from one coherence state to another. To that end, we execute a benchmark during the warm-up phase that sets several lines in the L2 cache to a given coherence state. Afterwards, during the execution phase, another benchmark is executed to force the coherence state of those cache lines to change. Such benchmark accesses a subset of the addresses accessed in the warm-up phase. Moreover, the benchmark in the execution phase can access other addresses not shared with the benchmark executed in the warm-up phase. This is explained on a per experiment basis.

We set a breakpoint right after the warm-up phase (right before the execution phase) to confirm that the state is the one we expect. We also set another breakpoint right after the execution phase to verify that the cache lines have changed to the desired state.

Benchmarks. We use two micro-benchmark types, CPU and DMA, each with read (r) and write (w) variants, resulting in 4 combinations (CPUr, CPUw, DMAr, DMAw). The CPU benchmarks perform 128,000 accesses mapped to a set of either 4,096 or 8,192 different addresses. We refer to these benchmarks as CPUx(4K) and CPUx(8K) respectively, where 'x' is either 'r' or 'w' and the number of accesses is omitted as it is the same in all variants.

The DMAr benchmark reads data from a memory address range including the one used by the CPU benchmarks and writes it to a not overlapping memory address range, while the DMAw benchmark reads from the non-overlapping range and writes to the overlapping one. Both benchmarks, transfer either 2MB or 4MB of data. Hence, they perform 32,768 or

65,536 64B accesses to the overlapping range and the same number to the non-overlapping. We refer to these benchmarks as DMAx(32K) and DMAx(64K) respectively. Note that DMAr and DMAw perform necessarily both read and write operations.

Workloads. A workload comprises the execution of one or several benchmarks during the warm-up phase and one during the execution phase. For instance, $[CPUw(4K), DMAr(32K); DMAw(32K)]$ shows that during the warm-up CPUw(4K) and DMA4r(32K) are executed one after the other; and in the execution phase the DMAw(32K) is executed.

III. Analysis

We analyzed all the Coherence Protocol transitions; here, we only show the two transitions with the most interesting details. The full analysis of all transitions can be found in [8].

TABLE II. PMCs for "M to S" and "S to M" transitions.

State		L2DA	L2SH	L2SP	ESR	L2SM	L2SS	L2RC	L2DM	BL	CL
M2S	A	13	3968	3968	65536	3968	0	6432	0	0	6447
	B	12	8064	8064	65536	8064	0	6418	0	0	6418
	C	13	3968	3968	32768	3968	0	3226	0	0	3241
	D	13	8064	8064	32768	8064	0	3226	0	0	3242
S2M	A	127998	0	0	0	0	0	3968	0	3968	0
	B	127998	0	0	0	0	0	8064	0	8064	0

Modified to Shared. During the warm-up phase, we run CPUw that causes the data to be dirty in L2. In the execution phase, we run DMAr to force snoops from the CoreNet addressed to the L2 that has to (i) send the data to the main memory, (ii) clean the dirty flag and (iii) move the cache line to the shared state. We evaluate four scenarios: (A) [`CPUw(4K); DMAr(64K)`], (B) [`CPUw(8K); DMAr(64K)`], (C) [`CPUw(4K); DMAr(32K)`], and (D) [`CPUw(8K); DMAr(32K)`].

As we start measuring right after the warm-up (i.e. the execution of CPUw), there can be few accesses that are pending to L2, causing the few L2DA, as shown in (M2S) case in Table II.A GetS/GetM message is sent for every address read or written by the DMAr [1]. Hence, the number of expected ESR is 65,536 in (A) and (B) and 32,768 in (C) and (D), matching the observed results in Table II (M2S).The GetS messages hit in the L2 for the memory locations that were accessed previously by the CPUw, which results in 4K[2] L2SH in (A) and (C) and 8K[3] in (B) and (D).

Unexpectedly, the L2 responds to each of these snoop hits in the L2 by requesting other coherent devices to set such lines as shared, but there is no other coherent device in the T2080. This matches L2SP and L2SM values.

Shared to Modified. During the warm-up, we run CPUw that causes the data to be dirty in L2, and then we run DMAr, which sends a request for the data and the L2 sends the dirty data to the CoreNet moving to Shared state. In the execution phase, we run CPUw, which sends GetM requests to the CoreNet to invalidate all the valid copies in any potential device with that data set as shared, despite there is none. As a simultaneous modification of this data could have occurred in another device with a shared copy of the data, the L2 sends a data forward request to CoreNet. This request might bring either no new data or an updated copy if a remote modification occurred between the local modification and its notification to other coherent devices. We evaluate the following four scenarios: (A) [`CPUw(4K),DMAr(64K); CPUw(4K)`], and (B) [`CPUw(8K),DMAr(64K); CPUw(4K)`].

The task analyzed (CPUw) generates accesses from the CPU as captured by the L2DA counter in Table II (S2M). Each of these accesses triggers the L2 to send a GetM message to ask all other sharers to invalidate their own copy of the shared data to the CoreNet, generating around 4K BL in (A) and around 8K in (B). As explained, since other coherent devices could be performing simultaneous modifications of the shared data, the L2 performs 4K L2RC in (A) and around 8K in (B). Those L2RC receive no answer since there is no other coherent device in the platform.

[1]Recall that DMAr also performs writes that result in GetM requests.

[2]In reality, we expect $4,096 - 128$ and $8,192 - 128$ respectively, as we perform 31 and 63 iterations respectively of 128 accesses each.

IV. Conclusions

Our empirical analysis of cache coherence in the T2080 brings some lessons learned. First, we can identify the events triggered by each coherence state transition, providing a clearer understanding of the implemented cache coherence behavior. Second, there are some hardware monitors with ambiguous or incomplete descriptions of the events tracked. And third, we detect unexpected coherence messages for a single L2 coherent cache processor. All these elements help validate the cache coherence protocol itself and allow building other further validation evidence on top of it.

V. Acknowledgment

This work has been partially supported by the Spanish Ministry of Science and Innovation under grant PID2019-107255GB; the European Union's Horizon 2020 research and innovation programme under grant agreement No. 878752 (MASTECS) and the European Research Council (ERC) grant agreement No. 772773 (SuPerCom); the HiPEAC Network of Excellence; and the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] Freescale Semiconductor, "QorIQ T2080 Reference Manual," 2016, Also supports T2081. Doc. No.: T2080RM. Rev. 3, 11/2016.

[2] Xilinx, "Zynq UltraScale+ Device Technical Reference Manual," https://www.xilinx.com/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf, 2019, UG1085 (v2.1).

[3] G. Fernandez et al, "Contention in multicore hardware shared resources: Understanding of the state of the art," in *WCET Workshop*, 2014.

[4] J. Barrera et al., "On the reliability of hardware event monitors in mpsocs for critical domains," in *ACM SAC*, 2020.

[5] N. Sensfelder et al., "On how to identify cache coherence: Case of the NXP qoriq T4240," in *ECRTS*, 2020.

[6] D. Radack et al. (Rockwell Collins), "Civil Certification of Multi-core Processing Systems in Commercial Avionics," 2018.

[7] Freescale Semiconductor, "e6500 Core Reference Manual," https://www.nxp.com/docs/en/reference-manual/E6500RM.pdf, 2014, E6500RM. Rev 0. 06/2014.

[8] R. Pujol et al, "Empirical evidence for mpsocs in critical systems: The case of nxp's t2080 cache coherence," in *DATE*, 2021.

Roger Pujol received his BSc degree in Computer Engineering with specialization in Computation from Universitat Politècnica de Catalunya (UPC), Spain, in 2018. That same year, he started working as Research Student at the Computer Architecture - Operating Systems (CAOS) group of Barcelona Supercomputing Center (BSC). He also completed his Master in Innovation and Research in Informatics with specialization in Advanced Computing from UPC, Spain, in 2020. Since then, he remained in BSC's CAOS group and started as a Ph.D. student at the Department of Computer Architecture of UPC, Spain.