Mechanisms for hardware virtualization in multicore architectures

Solutions of identified limitations of the hardware virtualization

Project acronym :	MultiPARTES
Project Number:	287702
Version:	v1.0
Due date of deliverable:	Month 14
Submission date:	15/11/2012
Dissemination level:	Public
Author:	Imanol Martínez (IK4-IKERLAN)

SEVENTH FRAMEWORK PROGRAMME

Part of the Seventh Framework Programme
Funded by the EC-DGINFSO

Table of Contents

1 DOCUMENT HISTORY 3

2 EXECUTIVE SUMMARY 4

3 ABOUT THIS DOCUMENT 5

3.1 PURPOSE OF THIS DOCUMENT 5

3.2 STRUCTURE OF THIS DOCUMENT 5

4 INTRODUCTION 7

5 REVIEW OF DELIVERABLE D6.1 8

6 HARDWARE MECHANISM DESCRIPTION 10

6.1 PREDICTABLE AND COMPOSABLE MEMORY HIERARCHY FOR COTS PLATFORMS 10

6.1.1 Composable and predictable cache 11

6.1.2 Composable and predictable bus 13

6.1.3 Composable and predictable DRAM memory 15

6.1.4 Conclusion 15

6.2 THE ACROSS MPSOC 15

6.2.1 Major Design Objectives 15

6.2.2 Architectural Approach 16

6.2.3 The Time-Triggered Network-on-Chip 16

6.2.4 Proposed extension to support memory intensive applications 18

6.2.5 Conclusion 19

7 THEORETICAL ANALYSIS OF HOW THE XTRATUM BENEFITS WITH THESE MECHANISMS 20

7.1 CPU CACHE 20

7.2 BUS ARBITRATION SCHEMA 21

7.3 MULTI-PORT MEMORY ARBITRATION SCHEMA 21

7.4 TIME-TRIGGERED SYSTEM-ON-CHIP (TTSOC) APPROACH TO INTERCONNECT MULTIPLE CORES 22

7.5 PREDICTABLE AND COMPOSABLE MEMORY HIERARCHY 22

8 THEORETICAL ANALYSIS OF HOW A VIDEO SURVEILLANCE APPLICATION SHOULD BENEFIT WITH THESE MECHANISMS 23

8.1 CPU CACHE 23

9 CONCLUSIONS 25

10 BIBLIOGRAPHY 26

1 Document History

Version	Status	Date
V0.0	Draft	17/09/2012
V0.1	Draft. First complete version	23/10/2012
V0.2	Draft. Modifications to complete version	29/10/2012
V0.3	Draft. VTools and Fentiss modifications to complete version	1/11/2012
V0.4	First partners complete review	09/11/2012
V1.0	First Official Version	14/11/2012

Approval		
	Name	Date
Prepared	Imanol Martínez	1/11/2012
Reviewed	Javier Coronel / Salvador Trujillo / Christian El-Salloum	14/11/2012
Authorised	Salvador Trujillo	15/11/2012

Circulation	
Recipient	25/10/2012
Project partners	1/11/2012
European Commission	15/11/2012

2 Executive Summary

Virtualization is the process of abstracting computing resources. Using virtualization a logical partition can be defined to represent a collection of actual hardware resources. Virtualization has been used in servers for some years. Recently the explosion of embedded systems and the increase in their performance power and number of resources have moved the virtualization into these systems. Furthermore, the rapid proliferation of multicore devices, homogeneous and heterogeneous, has created an increased demand for applying virtualization technology.

When virtualizing a heterogeneous multicore architecture, a resource sharing problem arises due to the existence of different logical partitions accessing the same resources concomitantly. To provide deterministic behavior equivalent to that of single-core devices, virtualized multicore architectures implement mechanisms to cope with this sharing issue.

Most of existing virtualization techniques are based on hardware virtualization extensions. These features ease significantly the hypervisors for the virtualization and reduce the maintenance overhead of the guest operating system. In the coming future, it is expected that the hardware-assisted virtualization would further develop for the CPU, memory and devices I/O.

Even if the virtualization of multi-core architectures provides benefits for engineering mixed-criticality systems, they are not yet ready to be used in actual systems due to the complex WCET analysis which makes the process of verification very complex and uncertain. For a single core platform, the main sources of unpredictability are pipelining, branch prediction, caches and DRAM refreshing; features which are used to accelerate the average case execution time (ACET) of the system. For multi-core based systems this list of unpredictability is even longer due to cache coherency mechanisms, bus arbitration, and shared DRAM making WCET analysis and verification process even harder. The need of spatial and temporal isolation among the different partitions is the key for enabling the virtualization in heterogeneous multicore embedded systems to be used in mixed criticality systems.

Based on the previous analysis, this deliverable identifies solutions for the limitations determined for the hardware virtualization. A proposal of one hardware-based virtualization mechanism is introduced where the selected mechanism is a group of memory related hardware, composing a memory hierarchy that can be used in a virtualized multi-core platform. This memory hierarchy is shared between the different cores providing temporal and spatial isolation of the memory for each logical partition. This makes the memory hierarchy composable and therefore suited for mixed-criticality integration

3 About this document

This section describes the purpose and the structure of the document.

3.1 Purpose of this document

This document is the output of Task 6.2 (Subtask 1). It contains the definition of a solution to the identified limitations of hardware based virtualization techniques. We propose a memory hierarchy that can be used in a virtualized multi-core platform and that provides composability and predictability to the system, bounds WCET, and simplifies the verification process. This document will be used as input for Task 6.2 (Subtask 2) in which an implementation of part of these components is to be done.

The major dependencies among the tasks and their deliverables can be seen in Figure 1.

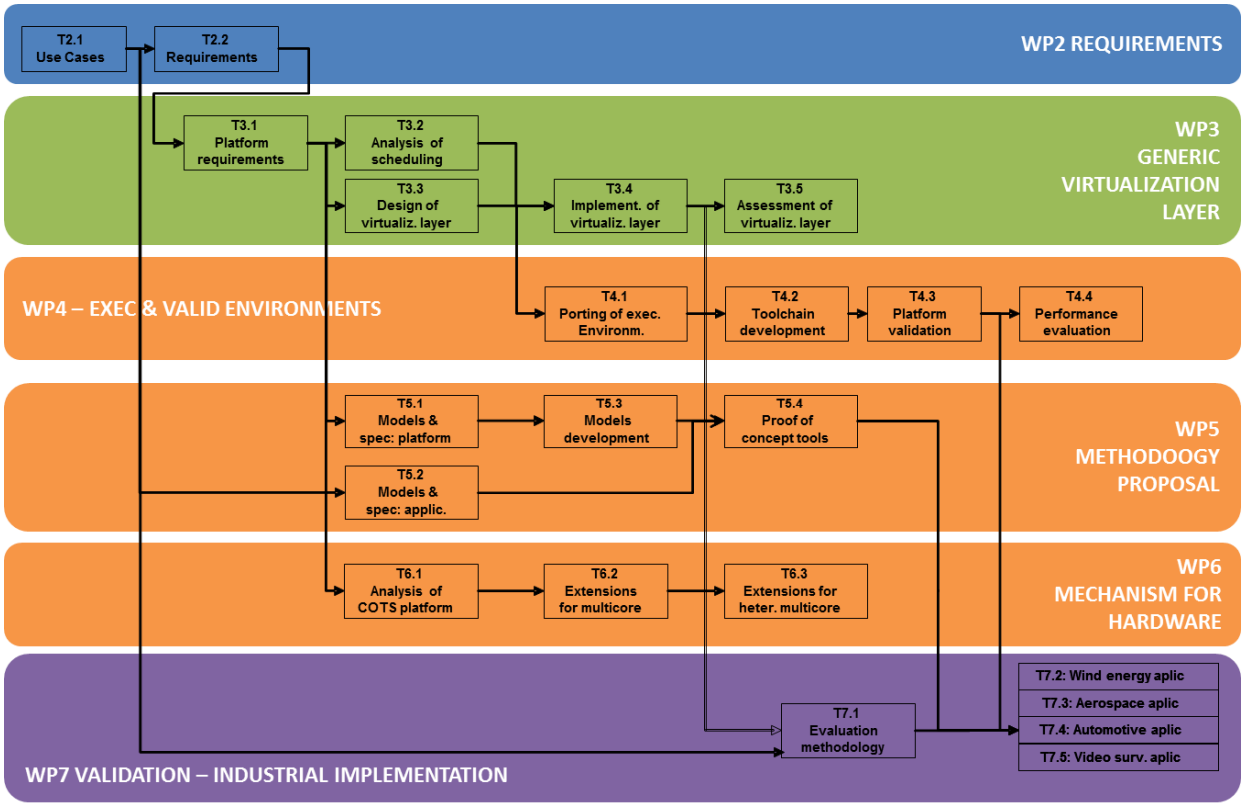

Figure 1. MultiPARTES work package and task dependencies.

Deliverable D6.1 is the main input to this document. Deliverable D6.1 presents state-of-the-art hardware components with a focus on memory hierarchy and inter-core communication, hardware assisted virtualization approaches, covering ISA extensions and I/O virtualization and a discussion covering their properties for spatial and temporal isolation.

3.2 Structure of this document

Section 4 gives a short introduction to general concepts such as multicore, virtualization, virtualization of multicore systems and some examples of constraints and difficulties when trying to virtualize these systems.

Section 5 presents a review of deliverable D6.1, main covered issues (spatial and temporal isolation) and an enumeration of hardware properties and components which influence on virtualization for embedded systems.

After introducing the main components that are used to build a memory hierarchy and analyzing the benefits and limitations of each of them with respect to spatial and temporal isolation, section 6 describes a solution combining all these elements and observing them as a unique component taking into account composability and predictability.

Due to the importance of the application of these mechanisms in the different application domains covered by the project, section 7 and section 8 describe the benefit of using a deterministic memory hierarchy by an hypervisor such as the XtratuM and how a video surveillance application should benefit of it.

Finally section 9 summarizes the main conclusions derived from our work.

4 Introduction

Today's SOC (system-on-chip) processors integrate a diversity of cores, accelerators, and other processing elements. These heterogeneous multicore architectures provide increased computational capacity, but the resulting complexity also poses new challenges for embedded-system developers. Discrete cores each have full access and control of their resources. Such predictable access allows straightforward management and deterministic performance in applications with real-time constraints. In a multicore architecture, however, cores share access to resources, and potential contention complicates many design factors, such as resource access latency and deterministically handling of these accesses.

Virtualization is the process of abstracting computing resources. Using virtualization a logical partition to represent a collection of actual hardware resources can be defined. Each logical partition consists of shared or private memory, one or more processors and a collection of accelerators and I/O devices. Each of these resources, including the processors, may be actual dedicated hardware or may be implemented by partially or completely virtualized hardware.

When virtualizing a heterogeneous multicore architecture we are facing a resource sharing problem created due to the existence of different logical partitions accessing the same resources as well as a synchronization problem between partitions. To provide deterministic behavior equivalent to that of single-core devices, virtualized multicore architectures should implement mechanisms to cope with this sharing and synchronization issues.

There are two ways to share these common resources among logical partitions. The first way to share hardware is to use the hypervisor to virtualize the resource. An OS driver could use register emulation or a hypercall to the hypervisor's global driver which would then manage sharing amongst all logical partitions. Some external interfaces can be challenging to share. This is especially true for address-mapped interfaces such as PCI Express. In this case, the lack of a standard channelized logical layer makes it difficult to decide how to direct inbound transactions to a particular logical partition. As a result, these interfaces are typically dedicated to one logical partition and all other partitions that wish to use the interface must do so through the dedicated partition. The second way, and the one analyzed in this document, is to create hardware based mechanisms that provide ways to the partition to interact with these resources as if it was the only device using it. External interfaces can also be shared through these hardware-based mechanisms. By example, an Ethernet network interface can classify inbound packets to a queue associated with a specific logical partition. Hardware then places the packet in a queue owned by that partition. The process is similar for transmission where each logical partition can submit packets using its private hardware queue.

In our case, the selected mechanism is a group of memory related hardware composing a memory hierarchy that can be used in a virtualized multi-core platform. This memory hierarchy is shared between the different cores providing composability and predictability to the system, with a bounded WCET and simplifying the verification process.

5 Review of Deliverable D6.1

Virtualization has recently resurged in popularity mainly in the server-side market due to improved utilization by deploying multiple virtual machines on a single hardware platform, usually multi-core CPUs. Recently virtualization has moved into the embedded arena and specifically mixed-criticality real-time systems due to a shift from federated to integrated architectures. The main difference to the server market is that, apart from strict spatial isolation, also temporal isolation has to be provided to preserve real-time guarantees and safety implications.

Spatial isolation between partitions means that the data private to one partition cannot be read or altered by any other partition. This ensures the confidentiality and integrity properties.

Temporal isolation between partitions means that there is no interference between partitions in the temporal domain. This means that neither concurrent nor non-concurrent partitions can alter the temporal characteristics of a partition. In the strictest sense, it means that the execution time of a partition is independent from the behavior of any other partition. If complete temporal isolation is guaranteed, the WCET analysis of any partition can be conducted independently of any other partition in the system. **This is the major requirement for modular certification and mixed-criticality integration.**

From a generic point of view, benefits and limitations of current virtualization techniques were discussed in this deliverable. The main finding was that currently new hardware assisted virtualization technologies are emerging from hardware manufacturers due mainly to the increase of the interest to use virtualized system in the market. However, this technology is still available in a reduced number of processor models, and it is mainly aimed at the large server market. Therefore, these new extensions are being incorporated slowly into embedded systems used commonly in the industrial sector. A clear trend in virtualization is the use of hardware virtualization extensions. These features make it much easier to build hypervisors and reduce the maintenance overhead of the guest operating system. Over time, it is expected that the hardware-assisted virtualization is further developed for the CPU, memory and devices I/O. However, the main challenges consist in improving the low performance obtained when using these extensions compared with the use of paravirtualization techniques. The hardware-assisted virtualization is presented as the future of virtualization.

Taking these issues into account deliverable D6.1 presents state-of-the-art hardware components with a focus on memory hierarchy and inter-core communication, their properties for spatial and temporal isolation, as well as a discussion of hardware assisted virtualization approaches, covering ISA extensions and I/O virtualization. This also includes a section on hardware platforms currently in use in the application domains for the demonstrators.

The discussion of the (I/O and ISA) virtualization extensions shows how they ensure spatial isolation between partitions and devices and reduce the associated software complexity, but do not yet provide adequate temporal isolation as is additionally required for applying virtualization in mixed-criticality systems.

Approaches to ensure temporal isolation like cache partitioning, scratchpad, or static bus arbitration were described. These techniques were originally developed in isolation without considering their impact on other layers of the memory hierarchy. However, to ensure temporal isolation it is required to take the behavior of the whole memory hierarchy into account, specifically the interference created by ensuring isolation on one particular level with

the other levels. Moreover it is important to consider the memory hierarchy together with the actual interconnect in order to ensure the required temporal properties satisfying the stringent requirements of safety-critical hard real-time systems and certification requirements for mixed criticality systems.

From another point of view, relevant platform-specific requirements (from deliverable D3.1) are revisited and a short explanation of why the implementation of this requirement should be done by using a hardware-based technique is described.

To summarize, by using cache partitioning and a composable memory controller, interference resulting from data accesses can be removed. This way a deterministic execution of the partitions could be guaranteed.

6 Hardware mechanism description

Even if the virtualization of multi-core architectures provides benefits for building an integrated architecture, they are very hard to be used in real time systems due to the complex WCET analysis which makes the process of verification very hard. For a single core platform, the main sources of unpredictability are pipelining, branch prediction, caches and DRAM refreshing; features which are used to accelerate the average case execution time (ACET) of the system. For multi-core systems this list of unpredictability is even longer due to cache coherency mechanisms, bus arbitration, and shared DRAM making WCET analysis and verification process even harder.

WCET analysis of one virtual partition in virtualized multi-core architecture cannot be calculated without taking into account temporal interference caused from the other virtual partitions due to the shared hardware resources. Considering that the WCET for a single core platform has been studied for more than 20 years, still a solution for WCET analysis of virtualized multi-core architecture is an open issue. The project aims at building a platform that prevents temporal and spatial interference among virtual partitions so that each partition can be observed in isolation. This permits single-core WCET analysis techniques to be used in multi-core platforms. To address the problem of WCET analysis and verification in virtualized multi-core platforms, two complexity concepts are introduced: composability and predictability.

Virtual partitions are temporally composable if they are completely isolated and cannot affect each other's temporal behavior. Such partition can be analyzed and certified independently form the other partitions which is a major requirement for mixed criticality integration.

In order to reduce the complexity of mixed-criticality integration, in this chapter we propose a memory hierarchy that can be used in a virtualized multi-core platform and that provides composability and predictability to the system, bounds WCET, and simplifies the verification process. Based on the definition of composability the memory hierarchy is composable if each access to the memory from any virtual partition in virtualized multi-core architecture is not temporally disturbed by any access of the other virtual partitions. Considering predictability, the memory hierarchy is predictable if the maximum latency and the minimum bandwidth of each memory transaction is bounded.

In the previous deliverable D6.1 we have presented the main components that are used to build a memory hierarchy and analyzed the benefits and limitations of each of them with respect to temporal and spatial isolation. Here we give a solution on combining these elements and observing them as a whole component with respect to composability and predictability. In particular we will present two solutions, (i) a COTS-inspired memory hierarchy which is closer to the architecture defined in this project and which is ideally suited for legacy integration, and (ii) an extension of the ACROSS MPSoC architecture (Christian El Salloum 2012) which meets temporal and spatial isolation requirements but entails an entirely new hardware platform and programming model.

6.1 Predictable and composable memory hierarchy for COTS platforms

Figure 2 shows the proposed architecture and how each component is designed to prevent interference. The colors indicate the mapping of virtual partitions to partitioned components.

For example the virtual partition 1 marked in yellow is assigned one partition in the L1 cache as well as in the TLB. Also it shows how one DRAM bank is assigned to one core. Interference on the shared bus is removed by having a hierarchical TDMA based arbitration policy, which combines temporal isolation for critical tasks with higher utilization for non-critical tasks. The subsequent chapters will describe those mechanisms in more detail.

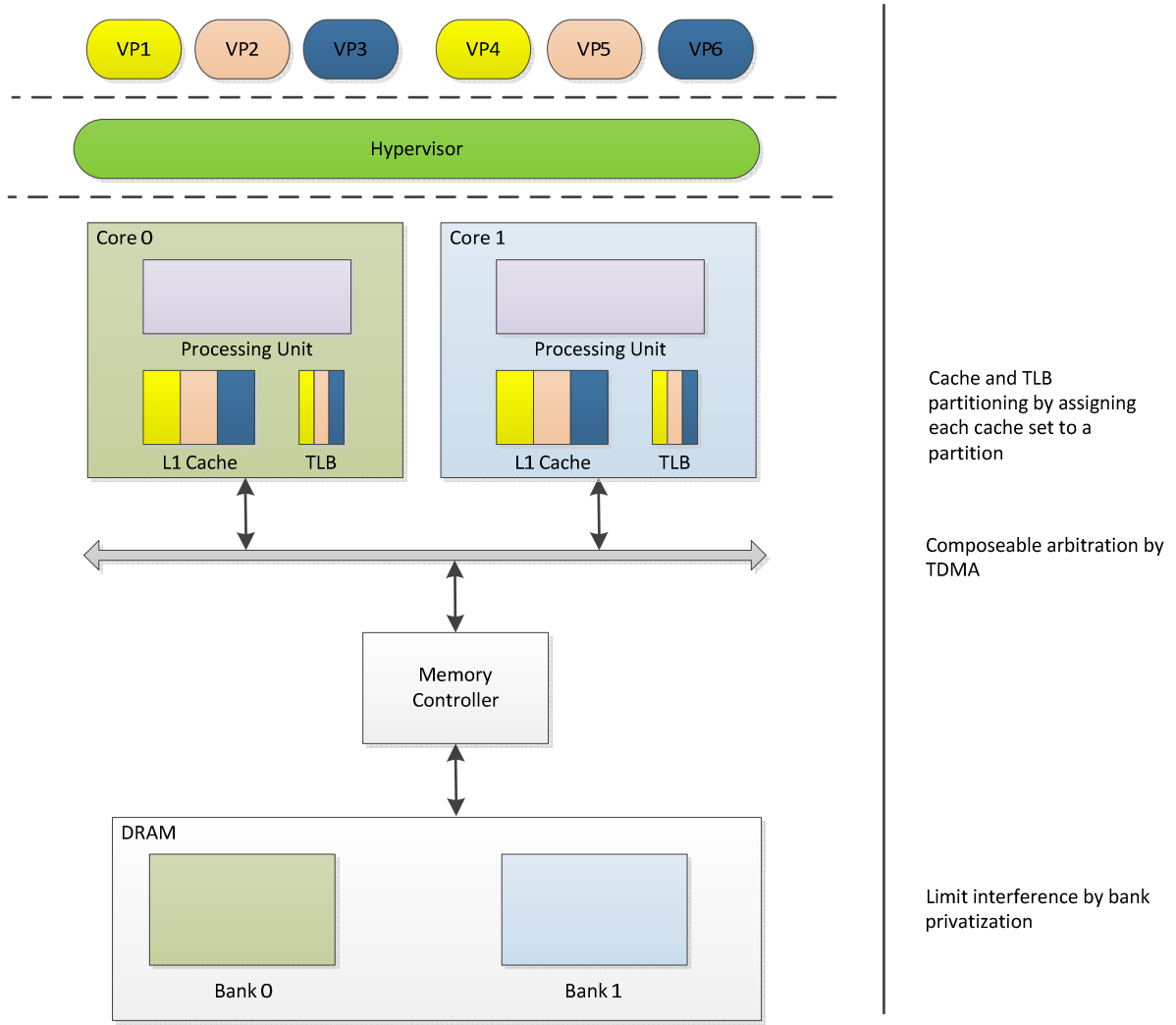

Figure 2. Proposed partitioned memory subsystem.

6.1.1 Composable and predictable cache

In a virtualized multi-core platform with private L1 caches, virtual partitions sharing a core will also share its cache. When a virtual partition accesses the memory, it first accesses the cache to check if the requested data is already there, if not, the request is passed to the memory controller. To make the cache composable, virtual partitions must not interfere with each other. This means that one virtual partition cannot evict cache data belonging to another partition.

Deliverable D6.1 described cache placement policies which can be transparent or software controlled. The behavior of a transparent cache is based on a heuristic policy and if two or more virtual partitions access the cache concurrently they can evict each other's cached data in an unpredictable way generating temporal interference between virtual partitions. Regarding

composability, the cache has to be partitioned so virtual partitions have their own cache space. This can be done by hardware or software. The decision is a tradeoff considering that hardware solution needs extra hardware and potentially limits the number of virtual partitions in one core with the number of cache partitions while software solution will add additional overhead (OS-solution) or additional execution time by adding jump instructions (compiler solution).

The second solution, is cache context switching. When a core switches between different virtual partitions it has to preserve the cache content by saving it in an additional memory location (e.g., local scratchpad) and preloading cache containment of the following virtual partition. This solution is beneficial only when cache context switching happens rarely and the cache has a small capacity.

As regards to predictability, if composability is established between virtual partitions, then WCET analysis for cache can be done using abstract interpretation (Martin Alt 1996), IPET (Yau-Tsun Steven Li 1996) or any other techniques used for single core processor.

In a virtualized multi-core architecture with mixed critical virtual partitions it's important to guarantee the execution time of virtual partitions that are highly critical. Observing on the level of one core that runs two or more critical virtual partitions, the cache can be split between virtual partitions in different way. The cache partitions can have different sizes, depending on the performance requirements of the given virtual partitions. Figure 3 shows cache and TLB partitioning on the level of one core with three virtual partitions where VP1 and VP3 are critical and VP2 is a non-critical virtual partition. Because the caches are private to the cores no partition on a different core can cause interference on any other core.

Furthermore by having both the cache and the TLB partitioned the temporal behavior of a virtual partition can be more easily reevaluated after system integration. A context switch does not change the temporal characteristics of a virtual partition as much as simple cache flushing would.

This also means that a context switch does not cause further accesses to the main memory, which would otherwise occur due to the additional cache misses during refilling of the cache state after a context switch.

This partitioning is achieved by changing a n-way set associative cache to give software control over the replacement policy which decides in which set data of a given partition is stored. This can be done by exposing a software controlled register that specifies in which cache set data is currently active. By having the hypervisor statically assign one virtual partition to one set, cache related interference is prevented.

The first solution is partitioning the cache into equal sizes and assigning one cache partition to each VP. The second solution is more flexible: If the performance requirements of VP1 are higher, a cache partition with a larger size can be assigned to VP1 and a smaller cache partition can be assigned to VP3. Both solutions provide composability between critical partitions and make it possible to analyze the WCET of each virtual partition in isolation. Nevertheless, the second solution will generally result in a more efficient platform.

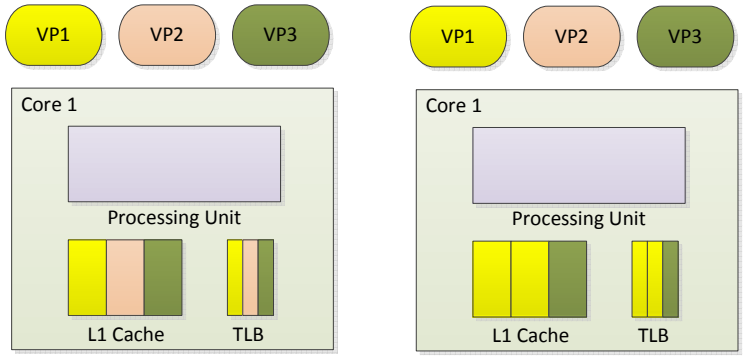

Figure 3. Cache partitioning in the level of one core

6.1.2 *Composable and predictable bus*

Buses are one of the most widely used means for communication between components in a multi-core architecture. Different systems can have different topologies of bus structures which affect cost, complexity, power, and the performance profile of the communication architecture.

The simplest topology for interconnection in a multi-core architecture is a shared bus (Figure 4). In a system with more than one master component (multi-core architecture) an arbiter is needed to determine which master gets the permission to access the bus. Deliverable D6.1 describes two main classifications of arbitrations policies (static and dynamic priority) for bus accessing. Using a dynamic policy for bus arbitration improves the system performances but on the other side increases the complexity of WCET analysis and prevents composability. With such architecture, a partition cannot be analyzed in isolation because of the temporal interference between critical virtual partitions and the WCET analysis has to be done for the entire system as whole. On the other hand, static arbitration schemes like TDMA provide composability but reduce the flexibility of the system and increase latency of memory access making the system often inefficient.

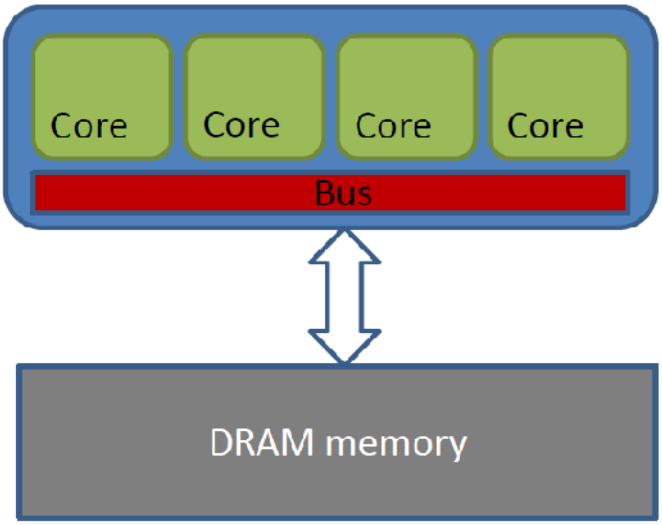

Figure 4. Multi-core architecture with shared bus

For a virtualized multi-core platform that has only one critical partition and all the others are non critical, arbitration can be solved very simple. The arbitration mechanism can always give the highest priority to the critical partition. In this case the arbiter always preempts any ongoing transaction by assigning the bus to the critical core and allowing it to use the bus as much as it needs it. This solution makes the critical partition behave like running on a single core system.

For virtualized multi-core platforms with critical virtual partitions residing on two or more cores we are proposing a bus arbitration policy that provides composability and predictability for critical partitions and also good performance for non-critical ones. Our arbitration scheme is a two-level policy that combines a static priority scheme with a dynamic priority scheme.

6.1.2.1 Static arbitration for cores with critical virtual partitions

The static priority scheme defines time slots of equal length. The length is defined by the worst case time to service a single request. Whenever the cache controller of a core generates a request for memory access it has to wait for its time slot to be served.

To be able to assign memory bandwidth in a flexible way, individual cores can be assigned multiple slots. In Figure 5 it can be seen that if one core with a critical partition (in this example Core1) needs to access memory more frequently it can be scheduled more often by the arbiter. This scheduling is static and defined during design time. For static slots, a request has to be issued before its time slot starts, otherwise it has to wait for its next dedicated time slot. If it would be allowed to serve a request in the middle of a static time slot the data transfer could overlap with the time slot reserved for the next access.

By knowing the time of one slot and their frequency in one round of arbitration we can derive the bandwidth and the maximum latency which makes the system predictable for calculating WCET.

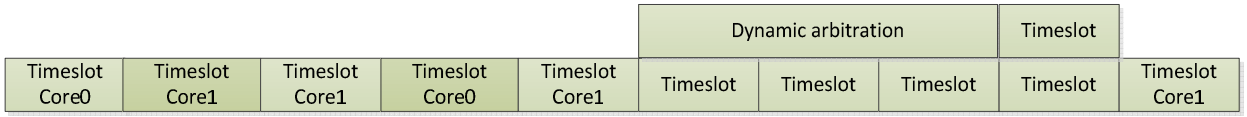

Figure 5. Arbitration scheme

6.1.2.2 Dynamic arbitration for cores with non-critical virtual partitions

To improve system performance without losing composability and predictability, we employ a dynamic arbitration scheme above the static arbitration scheme. In the dynamic arbitration scheme a number of time slots are reserved for noncritical partitions. During these time slots dynamic priority-based arbitration will be used to achieve better performance.

To ensure that the dynamic arbitration cannot interfere with the static arbitration, an additional rule has to be enforced. The dynamic arbiter must not serve any request that was issued after the start of the last timeslot in the sequence of dynamic timeslots. Otherwise, this request could overlap with the next static time slot for the critical cores (e.g., the slot of Core1 in Figure 5).

6.1.3 Composable and predictable DRAM memory

Composability with respect to shared DRAM memory means that an access of one core cannot interfere with the accesses of any other core. However, DRAM is highly variable considering access times because the duration of each access depends from the previous memory access. This variability increases the worst case latency for memory accesses and has to be considered for calculating the length of the time slots in the above mentioned arbitration scheme. To reduce this variability we propose to use closed page policy so the next row access to be faster.

Considering the predictability of DRAM, the main issue is the refreshing process because it is asynchronous with memory accesses and can delay an access if the bank is already in the process of refreshing. In DRAMs were refreshing is programmable we can schedule the refreshing process in the memory controller by giving it a separate time slot in schedule. During this time slot no transaction is allowed on the bus, and the memory controller just sends the refreshing command.

6.1.4 Conclusion

Our proposed hardware architecture guarantees temporal isolation of each virtual partition. This makes the memory hierarchy composable and therefore suited for mixed-criticality integration. By partitioning the private caches, unpredictability due to context switches is removed, and determining the temporal characteristics of the whole system is made easier. By using a TDMA based bus arbitration policy interferences due to concurrent accesses is prevented. Furthermore interference due to the state of the DRAM is hidden within each timeslot. By combining the static arbitration with a dynamic arbitration policy we gain additional efficiency while still retaining composability for critical tasks.

6.2 The ACROSS MPSoC

The ACROSS MPSoC[1] constitutes a new generation of multicore processors designed especially for safety-critical embedded systems with stringent hard real time requirements. This architecture has been developed in the partner project ACROSS. In contrast to MultiPARTES this architecture is not based COTS platforms, but on an entirely new design. In MultiPARTES it serves as a reference, since it provides perfect temporal and spatial isolation among all cores on the MPSoC.

6.2.1 Major Design Objectives

The design objectives of the ACROSS MPSoC architecture (Christian El Salloum 2012) are derived from the domain of safety-critical applications and include:

- Certifiability: The architecture shall enable time-efficient and cost-efficient certification of systems belonging to the highest criticality class. The failure rate in such systems is required to be lower than 10−9 failures per hour, which means approximately less than one failure in 100 000 years.
- Complexity management: The architecture shall support the independent development and verification of subsystems (i.e., different application functionalities). An example hereof is that the temporal behavior of an already integrated subsystem should not be changed by integration of a new subsystem.

[1] ACROSS Homepage: http://www.across-project.eu/

- Temporal determinism: The architecture shall enable the efficient construction, validation and certification of systems with a temporally deterministic behavior (e.g., guaranteed deadlines).
- Mixed-criticality integration: The architecture shall enable the efficient integration of subsystem with different criticality levels into a single MPSoC. This requires temporal and spatial partitioning mechanisms that prevent any non-intended interference and error propagation between subsystems.

6.2.2 Architectural Approach

The ACROSS MPSoC architecture is based on the following design decisions:

- Increased level of abstraction: In contrast to traditional MPSoC architectures, the MPSoC is considered as a networked multicomputer on a chip. To reflect this point of view, the notion of a Micro Component (µComponent) was introduced. A µComponent is a highly autonomous component, like a network node in a distributed system. It has sufficient local memory for executing its program code, and a precisely defined message-based interface as described below.
- Message-based Linking Interfaces (LIFs): A µComponent has an explicitly defined LIF via which it interacts with the other µComponents on the ACROSS MPSoC. The LIF is a message-based interface, which is not only defined in the value domain but also precisely in the temporal domain (e.g., periodic points in time when the message is sent).
- Restrict interactions to LIFs: The architecture restricts the interaction between the µComponents to occur exclusively via the explicitly defined LIFs described above (i.e., there are no other physical connections between the µComponents). This restriction prevents any non-intended interference or hidden channels among the µComponents. Restricting the interaction eases system integration, verification and certification.
- Deterministic Interconnect: In order to enable temporally deterministic behaviour of the system, the network infrastructure interconnecting the LIFs of the different µComponents has to be temporally deterministic. The ACROSS architecture implements a time-triggered Network-on-Chip (NoC) that satisfies this requirement.
- Fault and Error Containment: Fault and error containment is one of the most important objectives to enable efficient certification. With respect to systematic design faults, each µComponent constitutes a Fault Containment Unit, which means, that any systematic design fault within a µComponent can neither disrupt the correct behaviour of any other µComponent, nor the communication among other the µComponents. Fault containment for design faults is achieved by the temporal and spatial partitioning mechanisms provided by the time-triggered NoC.

6.2.3 The Time-Triggered Network-on-Chip

The time-triggered network-on-chip (TTNoC) (H. Kopetz 2001) constitutes the core of the ACROSS MPSoC architecture. This section describes the basic functionality of the TTNoC and its properties.

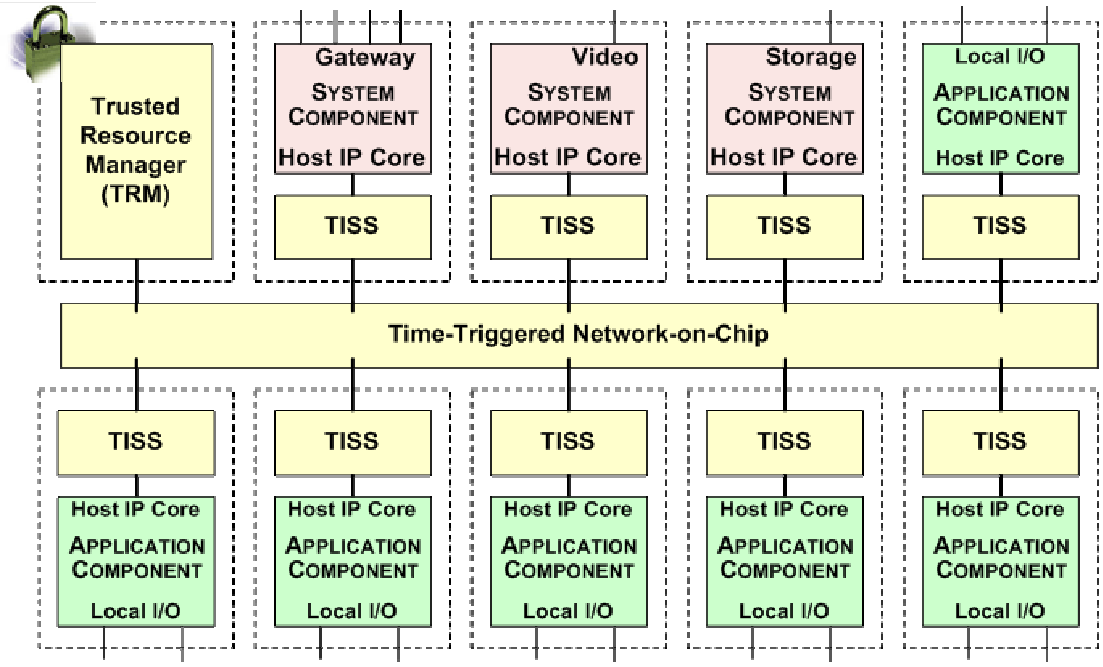

Figure 6. The Time-Triggered Network on Chip

6.2.3.1 Encapsulated Communication Channels:

The TTNoC provides a configurable set of independent logical communication channels on top of a shared physical communication infrastructure. These communication channels provide spatial and temporal partitioning, and are therefore called encapsulated communication channels. Spatial partitioning means that messages sent on a given communication channel are only visible on that channel and cannot be overwritten by messages from other channels. Temporal partitioning means that the temporal behaviour (e.g., bandwidth or latency) of any communication channel is independent of the communication activities of any other channel.

An encapsulated communication channel is unidirectional, has a defined sender, a defined set of receivers and is associated to a specific periodic or sporadic message. Communication is performed according to the time-triggered paradigm. Each µComponent has access to a consistent chip-wide notion of time which is called the Macro-Tick. All communication activities are driven by the Macro-Tick and follow an a priori defined time-triggered message schedule. The time-triggered message schedule defines for each encapsulated communication channel: the sender, the set of receivers, the message length, and the periodic or sporadic send instants of the associated message with respect to the Macro-Tick.

The time-triggered schedule is defined at design time. It is free of any temporal conflicts, which means that there is no point in time where any two messages would collide on any shared physical link. The a-priory defined schedule has the advantage that no on-line arbitration is required. Whenever the message is scheduled, it is guaranteed that all the required links in the NoC from the sender to all receivers will be free, and the message can pass on without being blocked or discarded. Thus, the temporal behaviour is ”designed”, into the system and not wearisomely analyzed after the implementation by complex methods (e.g., methods from queuing theory). This property eases certification to a significant extend.

6.2.3.2 The Trusted Interface Subsystem (TISS)

The TISS implements a message based interface that acts as a temporal firewall (H. Kopetz 2001) between each µComponent and the shared communication resources. It is the responsibility of the TISS to achieve temporal coordination of communication activities and task executions between different µComponents. Therefore, all TISSes within the MPSoC share the same global notion of time (i.e., the Macro-Tick). Based on this global time communication activities are only allowed at predefined instants. The time-triggered schedule defines at which points in time a message has to be sent to/received from the TTNoC, or when tasks have to be triggered. In order to prevent a faulty host from disrupting the communication between correct µComponents, the memories in which the schedules are stored cannot be modified by the µComponent itself.

6.2.4 Proposed extension to support memory intensive applications

In the pure ACROSS architecture each core has only local memory, and cores can only communicate via the deterministic TTNoC. This is the major concept how temporal and spatial isolation is achieved.

Some legacy applications might require more memory than is locally available to the core where the application is running. In particular this could be the case if a hypervisor with multiple partitions and multiple applications should be running on core.

To make the ACROSS architecture more aligned with the MultiPARTES approach we propose as an extension, a dedicated component that constitutes a *deterministic shared memory provider (DSMP)*.

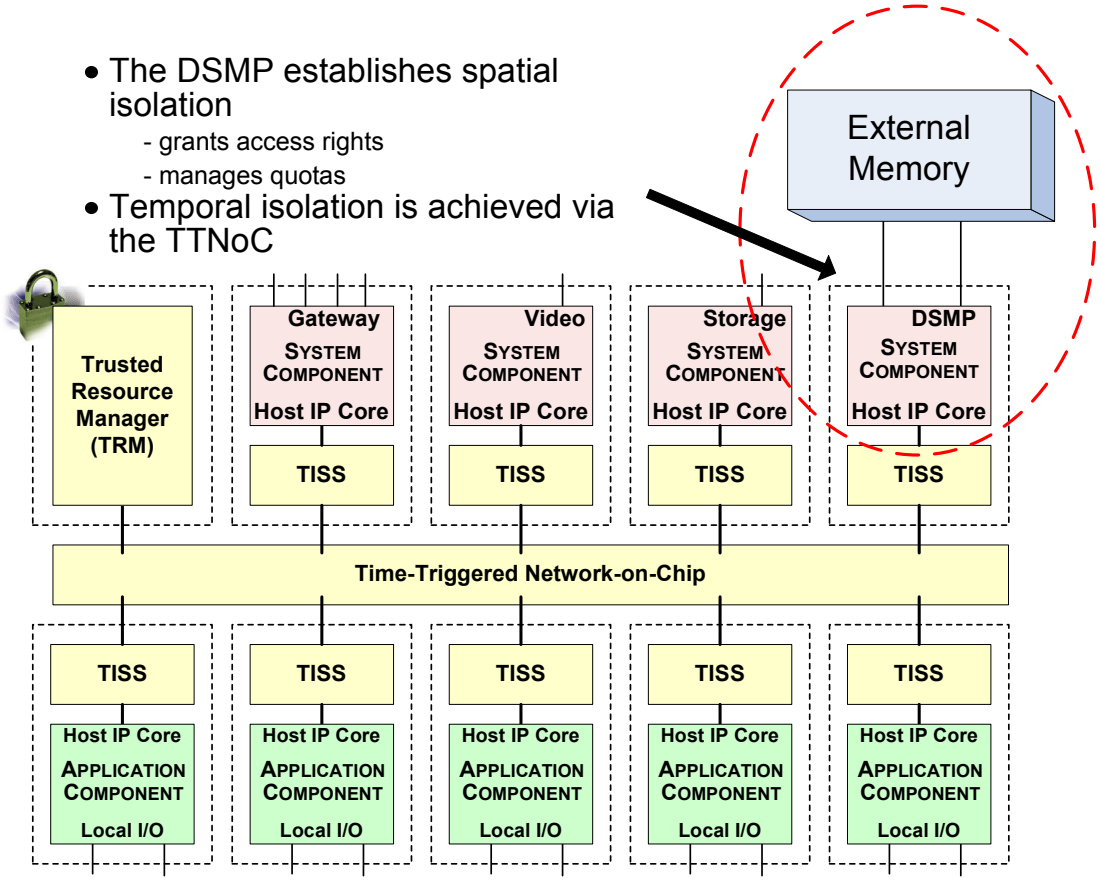

Figure 7. Deterministic Shared Memory Provider

The component is attached to a large external memory. Request to that memory will be encapsulated in messages exchanged via the TTNoC. The same is true for data that has to be transferred from the shared memory to back the cores. The DSMP assures temporal and spatial partitioning on the shared memory by two means:

a) Partitioning with respect to the concurrent accesses of the cores is done by employing deterministic communication service of the TTNoC. According to the bandwidth requirements of the individual core, each core is assigned exclusive slots of adequate length to communicate with the DSMP. Since the TISS do allow communication only during the allocated time slots, the temporal isolation property is always guaranteed. This is even true if the software of a host is faulty, because a host cannot override the a-priory defined schedule in the TISS.
b) Spatial partitioning is achieved by the following paging mechanism. The DSMP can be configured to define multiple memory regions. For each memory regions access rights can be defined for each core. If total spatial isolation between two cores is required, there must not be any memory region with access rights to both of the cores. If shared memory is desired for some of the cores a memory region will be explicitly defined that gives access rights to the selected set of cores.

6.2.5 Conclusion

The ACROSS architecture provides perfect temporal and spatial isolation among cores, but requires an entirely new programming model. Due to its temporal and spatial isolation properties it serves as a reference in MultiPARTES.

Moreover we proposed an extension that overcomes two limitations. The Deterministic Shared Memory Provider (DSMP) enables to run applications that require more memory that is locally available to a given core, and permits to retain the shared memory programming model by transparently encapsulating requests to the shared memory in messages exchanged via the TTNoC.

7 Theoretical analysis of how the XtratuM benefits with these mechanisms

Although some of the mechanisms presented during deliverable D6.1 are already available in current commercial processors, their main aim is not preserving temporal isolation and determinism of mixed criticality systems but improving the average case performance. Some of them as caches, and bus arbitration work in a transparent way, their configuration/behaviour come already predefined by default. This configuration makes sense in general purpose environments; otherwise the complexity for implementing any kind of software could become inaccessible.

On the other hand, high-critical systems require a rather high level of determinism, something sometimes incompatible with the existing "off-the-self" products. The theoretical exercise performed in this section assesses the way a virtualization hypervisor, XtratuM, could take advantage of the different hardware mechanisms previously described.

The processor, running at a high frequency, is connected through this system bus to slower peripherals (RAM, ROM, serial port, network-card, etc). Even this bus tends to run at slower frequencies. Due to this, the system bus is taken into account as a classical system bottleneck. This bottleneck effect can become even worse in a SMP system, where several symmetric processors are bound to the same system bus. In this section we evaluate the effect of the different mechanisms which are present or could be improved, in order to enhance system determinism: CPU cache, bus arbitration, multi-port memory arbitration scheme, etc.

7.1 CPU cache

The concept of CPU cache appears in order to mitigate the effect of a repeated use of the bus to retrieve the same data from main memory.

Cache is a smaller, faster area of memory usually located between the CPU and the bus which stores copies of the most commonly accessed data. In general CPU caches are transparent, having the OS the capability of enabling, disabling and flushing them. Additionally, the write policy is usually software-selectable.

Although CPU caches improve the average execution case, their use in high-critical system is discouraged since they increase the system indeterminism: in order to calculate the worst case execution time (WCET), the content of cache must be foreseen. Even though there are models of the cache effects, this prediction is even harder in SMP systems, the cache snooping mechanism which is watching the memory accesses performed by the rest of CPUS can stall the cache in order to update/invalidate its content.

Apart from all these issues, partitioned systems introduce an additional problem, inter-partition cache-content disturbances. A partition context switch can easily leave the cache content in an indeterministic state to calculate the WCET of a partition.

The European Space Agency (ESA) conducted three cache studies (PEAL (Prieto 2007), PEAL2 (Bernat 2007) (Betts 2009) and COLA (Mezzeti 2010)) where the objectives were:

- Exposing the cache indeterminism causes (particularly in the LEON processor).
- Mitigate/remove their effect.

The conclusions from these three studies and others around XtratuM (M. Soler 2012) should be considered as discouraging since, although the indeterminism causes were identified, the

proposed mitigation approaches were only useful in the case of very small applications in a controlled environments: cache locking/partitioning techniques heavily rely on the application memory layout and the result of an adequate layout is from far, hard to achieve and very memory consuming.

Therefore taking these results as our starting point, and identifying the cache as intrinsic source of indeterminism, as far as we know, the best option which can be used to improve indeterminism consist in replacing CPU cache memories by CPU scratch pad ones.

A scratch pad memory is, conceptually, a non-transparent cache, whose content is managed by software.

Undoubtedly the correct use of scratch pad memory would require additional analysis tools out of the scope of this theoretical analysis. However, since their content is always known, the source of indeterminism introduced by cache could be ignored.

Moreover, the use of such memories would allow the implementation of memory partitioning strategies which cannot be currently tackled but in very controlled environments.

7.2 Bus arbitration schema

On the other hand, although the use of CPU caches or scratch-pad memories can mitigate the effect of accessing to the system bus, processors still need to access it. In the case of a SMP system, the bus itself becomes a source of indeterminism since a processor can be stalled if the bus is being used at that moment by another processor.

In this kind of systems, the bus arbitration algorithm becomes crucial. In order to be usable this algorithm should allow to:

- Control the bus access: high-critical partitions should be priorized against lower-critical ones.
- Decrease indeterminism: Common used algorithms such as round-robin or fair ones, although acceptable for average cases, are a source of indeterminism which should be prevented in our case.

Therefore, concluding current estimations of the WCET taking account the most commonly used arbitration algorithms raise quite pessimist figures, therefore, the use of a more deterministic algorithm such as TDMA or similar which would conduct to better WCET estimations.

Additionally, in the case of mixed-criticality systems, a prioritization schema would highly improve system analysis, since non-critical petitions could be attended after higher critical ones.

7.3 Multi-port memory arbitration schema

When dealing with external RAM, it is common to find two kind of scenarios:

- A mono-port RAM connected to a single bus. In this case, the bus arbitration mechanism already selects who perform this access. Therefore, it is not necessary to take any decision at memory controller level.
- A multi-port RAM device connected directly to the processor. In this case, the processor is always able to access. Yet, multiple accesses still have to be serialized by the memory controller.

It is in the second scenario where the hypervisor could take advantage of using a schema similar to the one described in the case of the bus arbitration schema, that is, a priorization mechanism plus a second deterministic algorithm (such as, for example, TDMA) in the case of petitions with the same priority.

7.4 Time-triggered System-on-Chip (TTSoC) approach to interconnect multiple cores

As described in D6.1 the TTSoC is designed in order to provide a synchronous communication channel among heterogeneous cores. Although there is not the only existing alternative, since share memory is also a valid mechanism, TTSoC already solves those problems still present in the other approaches: high bandwidth, determinism, etc.

From the point of view of our system, the TTSoC would be used as a transparent communication mechanism for partitions. It should be abstracted to the already existing queuing and sampling ports. Additionally, this mechanism is also useful in order to communicate the different instances of the hypervisor in the different cores in order to improve system homogeneity. For example, a system partition should be able to halt a partition running a different core just executing the XM_halt_partition hypercall; the way this hypercall is implemented (for instance, through message passing) must be kept completely transparent to the partitions.

7.5 Predictable and composable memory hierarchy

By using the proposed predictable and composable memory hierarchy the effects of all the different mechanisms, CPU cache, bus arbitration, multi-port memory arbitration scheme, which have been presented are covered enhancing the system determinism.

8 Theoretical analysis of how a Video Surveillance application should benefit with these mechanisms

As it was mentioned in deliverable D6.1, strictly speaking there is not much written about videomonitoring and videosurveillance using virtualisation techniques. It is even more difficult to find specific material when analyzing mechanisms related to spatial and temporal isolation and their benefits for the videomonitoring sector.

From an industrial perspective, one of the most important benefits of a partitioned system is the capability of allowing third party applications in the videosurveillance equipment without affecting the normal functioning of the videomonitoring. This allows the manufacturers to reserve some CPU and memory for running applications which may complement the videomonitoring system (i.e., more business with the same system). The second benefit is that the partitioning reduces the time to market and the customisation and testing efforts, since the videomonitoring manufacturers have the possibility of developing innovative products for different sectors just installing specific applications to the non-critical partition, which hosts the third party applications.

At a lower level, for the videomonitoring sector it would be possible to focus on a few tasks related to video encoding and decoding which could be enriched with virtualisation techniques and the different extensions proposed in deliverable D6.1.

8.1 CPU Cache

H.264 (Wiegand 2003) is a video coding standard aimed to provide high-quality video at very low bit rates. One of his drawbacks is the computation power required to encode/decode the videos. Regarding H.264, Zhou et al. (Zhou 2003) presented some optimisations to improve the processing of the algorithm, some of them are related to the mechanisms proposed in MultiPARTES. In fact, they show a great deal of L1 and L2 cache misses in the motion compensation module, one of the most time consuming modules of H.264, because the reference frames where no totally loaded. In order to clarify this issue different test were made for evaluate the performance with different cache sizes, showing that with the same computing power (Pentium 4 processor) a reduction on the size of the L2 cache (from 512KB to 256KB) impacts on the overall performance. The runtime with 512KB was a 5% shorter.

If we take into account these values and find acceptable a small reduction on the run-time, we could play with partitioned caches of different sizes in order to check the overall performance of a H264 decoder.

Following with idea of the improvements on the video decoding, some efforts related to the bus arbitration have been made on AVS and H.264 decoding. For instance Dianfu Li et al. on (Li 2005) detail an algorithm for controlling the allocation of the SDRAM bus, reducing both the conflicts on the bus and the buffer size. Their simulation shows that it can be develop low-cost, low clock frequency real time decoder for embedded device. MPEG-2 video standard is another video decoding architecture which may benefit from efficient bus arbitration schemes. In fact Li and Ning (Li 1999) presented a MPEG-2 video decoding architecture with different bus arbitration schemes designed to meet MP@ML decoding performance requirements.

Together bus arbitration, memory access for video decoding was another relevant topic in the past years. For instance, studies like (Takizawa 2001) revealed a 95% efficiency accessing to memory thanks to the new arbitration algorithm they propose.

All these optimisations reveal that efficient mechanisms in the hypervisor layer will benefit applications making use of video decoding capabilities, like the video monitoring use case.

9 Conclusions

This document presented a hardware based virtualization mechanism. The selected mechanism is a group of memory related hardware, composing a memory hierarchy that can be used in a virtualized multi-core platform. This memory hierarchy is shared between the different cores providing temporal and spatial isolation for each logical partition. This deliverable provides a solution combining these elements and observing them as a whole component with respect to composability and predictability. In particular, two solutions were introduced: (i) a COTS-inspired memory hierarchy which is more close to our architecture in this project and which is ideally suited for legacy integration, and (ii) an extension of the ACROSS MPSoC architecture which provides complete temporal and spatial isolation but entails an entirely new hardware platform and programming model.

Our proposed hardware architecture guarantees temporal isolation of each partition, providing composability and predictability to the system. This makes the memory hierarchy suited for mixed-criticality integration. By partitioning the private caches, unpredictability due to context switches is removed, and determining the temporal characteristics of the whole system is made easier. By using TDMA based arbitration policy interferences due to concurrent accesses is prevented. Furthermore interference due to the state of the DRAM is hidden within each timeslot. By combining the static arbitration with a dynamic arbitration policy we gain additional efficiency while still retaining composability for critical tasks.

Due to the importance of the application of these mechanisms in the different application domains covered by the project we have described the benefit of using a determinist memory hierarchy by an hypervisor such as the XtratuM and how a video surveillance application should benefit of it.

10 Bibliography

Bernat, G., Colin, A., Esteves, J., Garcia, G., Moreno, C., Holsti, N. (2007). Considerations on the LEON cache effects on the timming anlaysis of on-board applications. Proceedings of the Data Systems In Aerospace Conference, Noordwijk, DASIA.

Betts, A. (2009). PEAL2 Final Report., PEAL2.

Christian El Salloum, M. E., Oliver Höftberger, HAris Isakovic and Armin Wasicek (2012). The ACROSS MPSoC - A New Generation of MultiCore Processors designed for Safety. 15th EUROMICRO Conference on Digital System Design. Izmir, Turkey.

H. Kopetz, G. B. (2001). "The Time-Triggered Architecture." Proceedings of the IEEE Special Issue on Modeling and Design of Embedded Software.

Li, D. a. Y., L. and Dong, J. (2005). "A decoder architecture for advanced video coding standard." Proceedings of SPIE **5960**: 9.

Li, J. H. a. L., N. (1999). "Architecture and bus-arbitration schemes for MPEG-2 video decoder." Circuits and Systems for Video Technology, IEEE Transactions on **9**(5): 9.

M. Soler, A. C., M. Masmano, R. Llorca-Cejudo (2012). Cache Management Techniques for Time Isolation in Partitioned Systems. Proceedings of Data Systems In Aerospace. DASIA 2012., Dubrovnik, European Space Agency.

Martin Alt, C. F., Florian Martin, and Reinhard Wilhelm (1996). Cache behavior prediction by abstract interpretation.

Mezzeti, E. (2010). COLA Final Report. Padua, University of Pauda.

Prieto, M., Guzman, D., Meziat, D., Sánchez, S., Planche, L. (2007). "LEON2 cache characterization. A contribution to WCET determination." IEEE International Symposium on Intelligent Signal Processing: 6.

Takizawa, T. a. H., M. (2001). "An efficient memory arbitration algorithm for a single chip MPEG2 AV decoder." Consumer Electronics, IEEE Transactions on **47**(3): 5.

Wiegand, T., Sullivan, G.J., Bjontegaard, G., Luthra, A. (2003). "Overview of the H.264/AVC video coding standard." Circuits and Systems for Video Technology, IEEE Transactions on **13**(7): 16.

Yau-Tsun Steven Li, S. M., and Andrew Wolfe (1996). Cache Modeling for Real-Time Software: Beyond DirectMapped Instruction Caches.

Zhou, X. a. L., E.Q. and Chen, Y.K. (2003). "Implementation of H. 264 decoder on general-purpose processors with media instructions." Proceedings of SPIE **5022**: 11.