CoMik: A Predictable and Cycle-Accurately Composable Real-Time Microkernel

Andrew Nelson[1], Ashkan Beyranvand Nejad[1], Anca Molnos[2], Martijn Koedam[3], Kees Goossens[3]
[1]Delft University of Technology, Netherlands, [2]CEA Leti, France, [3]Eindhoven University of Technology, Netherlands

***Abstract*—The functionality of embedded systems is ever increasing. This has lead to mixed time-criticality systems, where applications with a variety of real-time requirements co-exist on the same platform and share resources. Due to inter-application interference, verifying the real-time requirements of such systems is generally non trivial.**

In this paper, we present the CoMik microkernel that provides *temporally predictable and composable processor virtualisation*. CoMik's virtual processors are *cycle-accurately composable*, i.e. their timing cannot affect the timing of co-existing virtual processors by even a single cycle. Real-time applications executing on dedicated virtual processors can therefore be *verified and executed in isolation*, simplifying the verification of mixed time-criticality systems. We demonstrate these properties through experimentation on an FPGA prototyped hardware platform.

I. Introduction

Mixed time-criticality systems are becoming more common. As embedded systems are typically resource constrained, it is not always possible to provide each application with dedicated hardware resources. They must therefore share resources, such as a processor. The timing interference caused by resource sharing must be taken into account when modelling and verifying the application's timing requirements. On mixed time-criticality systems, non real-time and real-time applications can share the same resource, complicating the verification or even making it impossible.

As a solution to this problem, we propose the CoMik microkernel that virtualises the processor, as illustrated in Figure 1. It cycle-accurately divides the processor into Time Division Multiplexed (TDM) slots. A virtual processor consists of one or more of these slots on the same processor. Partitions consist of a guest Operating System (OS) or an application without an OS. Each partition can perform dynamic memory allocation, receive partition-level interrupts and perform independent power-management, using Dynamic Voltage and Frequency Scaling (DVFS), without interfering with the timing of concurrent partitions.

In this paper, we present the cycle-accurately composable CoMik microkernel. We contribute:

- The CoMik microkernel and its processor virtualisation.
- CoMik's virtual processor context swapping scheme that ensures that absolutely no temporal interference occurs between partitions, not even a single cycle.
- Our experimental analysis of CoMik's predictable and cycle-accurately composable properties on an FPGA prototyped platform.

II. Related Work

Interference between applications is a particular problem for safety-critical applications, such as those found in the automotive [1] and aeronautical [2]–[5] industries. The strictest standards are found in the avionics industry, hence we focus here on ARINC specification 653 [6], which is an avionics industry standard for the implementation of temporal and spatial partitioning [3], [5]. The standard also specifies requirements for interfaces, libraries and programming languages, enabling interoperability between ARINC 653 compliant systems [4]. LynxOS-178 [7], VxWorks 653 [8], INTEGRITY [9] and PikeOS [10] are commercially available ARINC 653 compliant RTOSs.

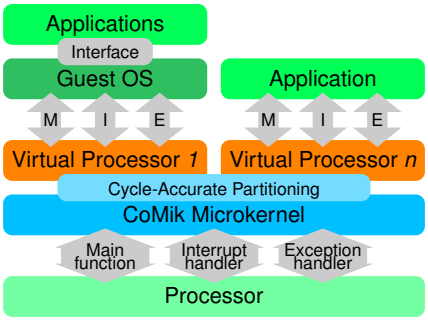

Figure 1. CoMik's cycle-accurately composable processor virtualisation.

ARINC specifies partitions at the application-level. One or more mixed-time criticality applications can belong to a partition [2]. ARINC 653 does not specify that a system must have run-time memory management. Instead, it is expected that a partition's memory usage has been verified prior to run-time [2]. ARINC 653 specifies that time on the processor is split into periodic major time frames. Partitions are allocated one or more time slices to execute within the major time frame. This guarantees the minimum amount of service that a partition receives, but not when it will receive it within the frame. The time at which a partition is serviced is therefore affected by the presence/absence of other partitions and is therefore not cycle-accurately temporally isolated. Inter-partition communication is permitted, allowing the timing of data dependent applications to be affected by applications in other partitions.

Outside of safety critical domains, temporal and spatial partitioning is used for embedded system virtualisation [11]. The OKL4 microvisor [12] is a virtualising microkernel, that is developed for use on mobile phones. It enables mixed time-criticality applications to execute on virtual machines as if they were running directly on the hardware platform. OKL4 uses thread-level partitions, with time slices allocated per-thread. Threads are scheduled following a priority based pre-emptive schedule. As with ARINC 653, precisely when the partition receives service depends on the presence/absence of other partitions. OKL4 permits inter-partition communication, enabling timing interference between the communicating partitions.

978–3–9815370–2–4/DATE14/©2014 EDAA

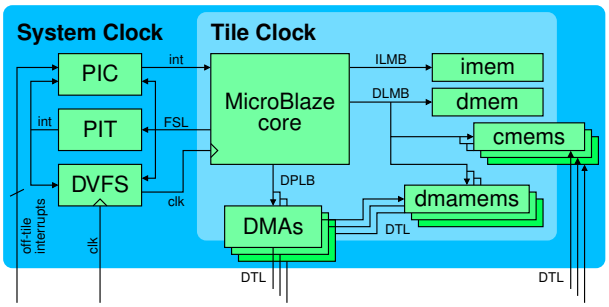

Figure 2. CompSOC processing tile

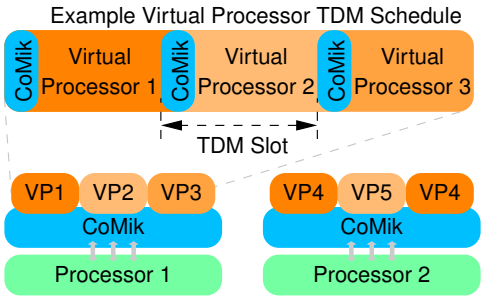

Figure 3. Temporal processor virtualisation.

CoMik combines partition-level cycle-accurate temporal isolation with a virtualised processor interface. Partitions are allocated one or more dedicated virtual processors on one or more shared physical processors. A limitation of CoMik's cycle-accurate isolation is that guaranteed partitions may not receive information from any other partitions, but they may send information in a non-blocking manner to best-effort partitions. Best-effort partitions may communicate freely, but can experience inter-partition timing interference due to the communication.

III. Background

The CompSOC hardware platform [13], [14] is designed for temporal composability and predictability and when coupled with CoMik's virtual processors enables virtual hardware platforms that are cycle-accurately isolated. Processing and memory tiles are connected via the composable and predictable Æthereal NOC [15]. Memory tiles can be used as a dedicated resource, or shared using a composable arbitration scheme, such as TDM or CCSP with delay blocks [16].

As illustrated in Figure 2, CompSOC processing tiles consist of a single MicroBlaze processing core, some local memory and one or more DMA to access off tile memory. The MicroBlaze core is a soft core that we configure to have a single 5-stage in-order pipeline with no branch prediction. The in-order pipeline simplifies the core's predictability, while disabling the branch prediction enables CoMik to maintain composability between partitions, as the branch predictor's state, which is influenced by application execution, would be carried over between partitions. These are general requirements that can be fulfilled on other processors. While not presented in this paper, we have also successfully ported CoMik to ARM's Cortex-M3 [17] processor.

In the CompSOC processing-tile, local memories are used to store application instructions (imem) and data (dmem), and are accessed via single-cycle Local Memory Buses (LMBs). A Memory Protection Unit (MPU) is not strictly necessary for temporal isolation, but can be used if required. Otherwise, it is assumed that the partitions memory space has been verified prior to run-time, similar to ARINC 653. DMA modules are used for inter-tile communication, and are accessed via the Processor Local Bus (PLB). They read/write directly from/to a dedicated local memory (dmamem) and the NOC. Inter-tile communication is achieved using Memory Mapped Input/Output (MMIO) transactions, using the Device Transaction Level (DTL) protocol. Data is transferred either from/to a shared memory tile or directly from/to a scratchpad communication memory (cmem) on another processing tile.

To maintain partition-level composability, all tile hardware involved in inter-tile communication must be dedicated resources or composably arbitrated. Figure 2 illustrates a suitable memory architecture using dedicated resources. Dual-port memories are used for the dmamems and cmems enabling single cycle memory access via the DLMB. One single port memory could be used with composable arbitration, but the choice is a trade-off between the cost of dual-port memories and a higher memory access latency due to arbitration.

A Programmable Interrupt Timer (PIT) is used to generate partition context-switch interrupts according to the processor's TDM schedule. The context-switch interrupt and off-tile interrupts are passed through the Programmable Interrupt Controller (PIC). It is programmed with a mask via the Fast Simplex Link (FSL), to only allow interrupts intended for a partition to pass through. The DVFS module enables per-tile voltage and frequency scaling.

IV. CoMik Microkernel

The purpose of the CoMik microkernel is to create multiple virtual processors that can be used as dedicated resources by partitions. These virtual processors are cycle-accurately temporally-isolated, meaning that activity on concurrent virtual processors that do not belong to the same partition, cannot affect each other's timing by even a single cycle. A partition can therefore be temporally verified in isolation as the presence/absence of concurrent partitions does not affect the partition's timing.

CoMik's virtual processors provide a similar interface to the physical processor. They require a pointer to a main function and optionally pointers to partition-level interrupt and exception handlers. As with a physical processor, the pointer to the main function points to the start of the partition's instructions that represent its functionality. Similarly, the pointer to the interrupt handler points to instructions that execute whenever a partition receives an interrupt, and the pointer to the exception handler points to instructions that execute whenever an exception is raised.

On multi-core platforms, such as CompSOC, CoMik creates multiple virtual processors per physical processor. CoMik operates in a distributed manner per physical processor, configuring each virtual processor using its slot allocation in CoMik's TDM scheduling table, as illustrated in Figure 3.

Partitions consist of either a guest OS, or an application without an OS, as illustrated in Figure 1. The timing of a partition is designated as being *guaranteed* or *best-effort*. Virtual processors allocated to guaranteed partitions only use the TDM slots allocated to them, whereas the virtual processors allocated to best-effort partitions may use otherwise unused

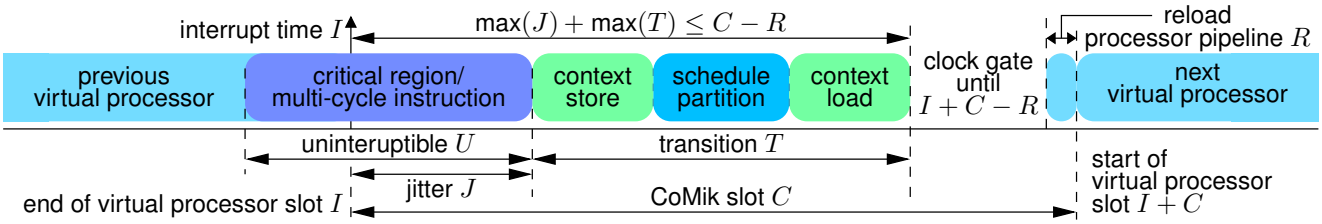

Figure 4. Cycle-accurate temporally-isolated virtual processor context switch.

TDM slots in addition to their TDM allocation. By only using their allocated TDM slots, guaranteed partitions ensure that the behaviour of other partitions do not affect their timing.

A. Virtual Processor and Partition Scheduling

As multiple virtual processors must share the same physical processor resource, TDM arbitration is used to decide which virtual processor is scheduled. TDM arbitration not only ensures a level of service but when the service will be delivered. It is also relatively simple to formally analyse.

Figure 3 illustrates how virtual processors are scheduled following a TDM schedule. In the diagram, physical processor 1 is virtualised as processors 1-3 and physical processor 2 is virtualised as processors 4 and 5. The TDM schedule for processor 1 is illustrated, showing that each virtual processor has one slot in the TDM table. At the start of each TDM slot, CoMik switches context from the previously scheduled virtual processor to the next virtual processor, ensuring cycle-accurate temporal isolation between them. This is explained in more detail in Section IV-B. Figure 3 also illustrates how a virtual processor may have multiple TDM slots, as shown for virtual processor 4 on physical processor 2. The slots do not need to be consecutive and may have any distribution within the TDM schedule.

If a slot in the TDM table has not been allocated to a virtual processor, or the slot is allocated to a virtual processor that will be clock gated for the entirety of the slot, the slot is deemed to be unused. These slots can be used by virtual processors that belong to partitions with best-effort timing. A round-robin arbitration scheme is used to decide which best-effort virtual processor gets the slot. Virtual processors allocated to partitions that are guaranteed to be cycle-accurately temporally isolated from other partitions cannot use the otherwise unused slots, as the availability of these slots depends on the presence/absence of other partitions and their timing.

CoMik does not perform any scheduling at the partition-level. Virtual processors are scheduled with cycle-accurate guaranteed or best-effort timing, but CoMik is agnostic to what a partition executes on them. This provides a clear separation of concerns. For instance, a guest Real-Time OS (RTOS) is free to use any partition-level scheduling scheme, but the timeliness of whatever processes/tasks/threads that it schedules is solely the responsibility of the partition. Care must also be taken at design time when dimensioning the TDM table. For instance, shorter TDM slots allow for a higher throughput of virtual processor context switches, enabling lower virtual processor response times, but increases context switching overhead.

B. Cycle-accurate Temporal Isolation

Partitioning and cycle-accurate temporal isolation simplifies the verification of real-time applications that share resources. This is partially achieved through scheduling, as explained in Section IV-A. CoMik's TDM scheduling scheme is regulated by a periodic interrupt that signifies a virtual processor context swap. Critical regions and multi-cycle instructions prevent the interrupt from being handled immediately. CoMik ensures that this jitter does not permeate to the next scheduled virtual processor, providing cycle-accurate temporal isolation.

Figure 4 illustrates how CoMik swaps virtual processor contexts. In this example, the scheduling interrupt arrives at time I, but cannot be handled immediately, as the processor is uninterruptible for a duration of U. This causes a jitter of time J. U is variable, depending on the critical region or multi-cycle instruction. After the jitter, control passes to CoMik's interrupt routine that performs the virtual processor context switch. The context of the previous partition is stored. This entails storing the state of the physical processor's registers, etc., on the stack of the partition. CoMik then schedules the next virtual processor as described in Section IV-A, before restoring its context. In Figure 4, this transition from one virtual processor context to the other takes time T. Duration T is variable, due to variation in scheduling time.

If the following virtual processor started immediately after its context is loaded, its precise start time would depend on the jitter J and the transition time T. To provide complete cycle-accurate isolation, CoMik ensures that the resumption time of the virtual processor is independent of this variation. We achieve this by splitting the TDM slot into a fixed duration CoMik slot and virtual processor slot, as illustrated in Figure 3. The CoMik slot starts at the time the context change interrupt is raised and lasts for a fixed duration, C in Figure 4. The virtual processor slot starts precisely at time $I + C$. We achieve this by clock gating the physical processor after the next virtual processor's context has been loaded. The processor is ungated at time $I + C - R$, which is a constant R cycles earlier than the start time of the virtual processor slot $I + C$ to allow the processor pipeline to return to the state it was in when the virtual processor was swapped out. For the MicroBlaze processor, R is 2 cycles to account for the instruction fetch and decode stages of the pipeline, enabling the virtual processor slot to start where it left off, with the instruction at the execution stage of the pipeline.

To ensure that the virtual processor slot starts on time, the jitter and the context transition time must be less than or equal to the time at which the physical processor ungates, $max(J) + max(T) \leq C - R$ as is illustrated in Figure 4. A definitive upper bound $max(T)$ can be derived for the duration of the transition time. No inherent upper bound for the interrupt jitter $max(J)$ exists, as there is no inherent limit to the duration of an uninterruptible critical region U. The jitter bound $max(J)$ is therefore a design decision that restricts the maximum length of partition-level critical regions. The jitter

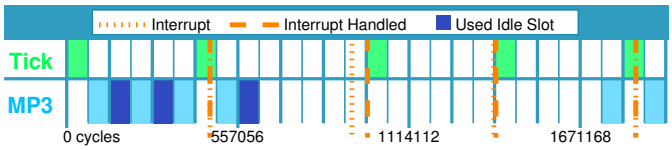

Figure 5. A best-effort MP3 decoder partition that is cycle-accurately isolated from a guaranteed partition responding to PIT interrupt "ticks".

bound should last minimally long enough to accommodate the processor's longest multi-cycle instruction. Increasing the duration of the jitter bound $max(J)$ also increases the necessary duration of the CoMik slot C. A trade-off therefore exists between accommodating a longer worst-case critical region and decreasing the CoMik slot overhead.

V. Experimentation

We continue by experimentally demonstrating CoMik's cycle-accurate temporal isolation and predictability. We demonstrate this for an FPGA prototyped instance of the CompSOC hardware platform, as described in Section III. The clock frequency is set to have an upper bound of 120 MHz. We configure CoMik to use a CoMik slot duration of 4096 cycles and a virtual processor slot duration of 65536 cycles, making a virtual processor TDM scheduling slot 69632 cycles.

We begin by demonstrating the timing of concurrently executing CoMik partitions. We do this by executing a soft real-time MP3 decoder as a best-effort partition and a firm real-time application that generates "ticks" in response to periodic interrupts, as a guaranteed partition. The tick partition clock gates between ticks, leaving full slots between ticks unused. Each partition is allocated a single virtual processor on the same physical processor. Each virtual processor has a single slot in a TDM table with a length of two slots.

Figure 5 presents the resultant timing of the two partitions. The tick is produced at regular intervals, except when the time of the tick does not occur when the tick partition is scheduled. In this instance the tick is produced in the partition's next scheduled virtual processor slot. The TDM slots that the tick partition leaves unused due to clock gating, are given to the best-effort MP3 decoder partition. The MP3 decoder therefore finishes decoding its frame earlier, enabling its power management scheme to temporarily clock gate the processor while still meeting its throughput requirement.

We conclude our experimentation by demonstrating that the timing of the tick partition, that is guaranteed to be cycle-accurately isolated, does not change by a single cycle whenever another partition is added to the system. To achieve this, we add another best-effort partition to be executed concurrently with the MP3 and tick partitions. The virtual processor of the added partition is not allocated any TDM slots, so it can only make use of unused slots. In Figure 5, the MP3 partition has a monopoly on unused slots, as it is the only best-effort partition. With the addition of another best-effort partition, the unused slots are allocated to partitions using a round-robin arbitration scheme.

Figure 6 presents the difference between the timings of the MP3 and tick partitions with the added best-effort partition, and the MP3 and tick partition timings that were used for Figure 5. The best-effort MP3 decoder partition shows timing variation between runs. While the MP3 partition still receives the level of service guaranteed by its virtual processor's TDM slot allocation, it must share any unused slots with

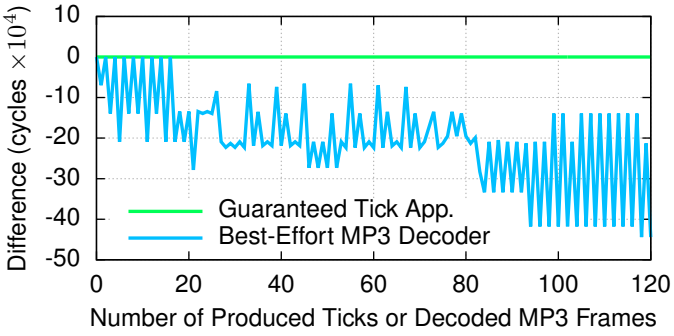

Figure 6. Timing difference when run with and without an additional Best-Effort Application.

the added best-effort partition. This demonstrates that the presence/absence of partitions can affect the timing of best-effort partitions. The guaranteed tick partition has exactly the same timing with or without the additional best-effort partition, demonstrating that it is cycle-accurately temporally isolated.

VI. Conclusion

The CoMik microkernel enables the creation of predictable and cycle-accurately composable virtual processors. Partitions that execute on dedicated virtual processors are cycle-accurately temporally isolated from interference by partitions executing on other virtual processors. Partitions can therefore be verified to meet their temporal requirements, in isolation. This verification remains valid on a system composed of multiple mixed time-criticality partitions. We experimentally demonstrate CoMik's predictable and cycle-accurately composable properties on an FPGA prototyped hardware platform, showing that mixed time-criticality partitions do not interfere with guaranteed partition timings by even a single cycle.

Acknowledgements This work was partially funded by projects EU FP7 288008 T-CREST & 288248 Flextiles, Catrene CA501 COMCAS & CA104 Cobra, CA505 BENEFIC, CA703 OpenES, & NL STW 10346 NEST.

References

[1] G. Pelz *et al.*, "Automotive system design and autosar," in *Advances in Design and Specification Languages for SoCs*, 2005.
[2] S. Samolej, "ARINC Specification 653 Based Real-Time Software Engineering," *e-Informatica*, 2011.
[3] J. Rushby, "Partitioning in avionics architectures: Requirements, mechanisms, and assurance," NASA, Tech. Rep., 1999.
[4] P. Prisaznuk, "ARINC 653 role in integrated modular avionics (IMA)," in *DASC*, 2008.
[5] J. Windsor *et al.*, "Time and space partitioning in spacecraft avionics," in *SMC-IT*, 2009.
[6] ARINC, "653 Avionics Application Software Standard Interface," `http://www.arinc.com`.
[7] LynuxWorks, "LynxOS-178," `http://www.lynuxworks.com`.
[8] Wind River, "VxWorks 653," `http://www.windriver.com`.
[9] Green Hills Software, "INTEGRITY," `http://www.ghs.com`.
[10] SYSGO, "PikeOS," `http://www.sysgo.com`.
[11] G. Heiser, "The role of virtualization in embedded systems," in *IIES*, 2008.
[12] G. Heiser *et al.*, "The OKL4 microvisor: Convergence point of microkernels and hypervisors," in *APSYS*, 2010.
[13] A. Molnos *et al.*, "A composable, energy-managed, real-time MPSoC platform," in *OPTIM*, 2010.
[14] A. Hansson *et al.*, "CoMPSoC: A template for composable and predictable multi-processor system on chips," *TODAES*, 2009.
[15] K. Goossens *et al.*, "The aethereal network on chip after ten years: Goals, evolution, lessons, and future," in *DAC*, 2010.
[16] B. Akesson *et al.*, "Real-Time Scheduling Using Credit-Controlled Static-Priority Arbitration," in *RTCSA*, 2008.
[17] ARM, "Cortex-M3 Processor," `http://www.arm.com`.