HAL
open science

Making Full use of Emerging ARM-based Heterogeneous Multicore SoCs

Felix Baum, Arvind Raghuraman

To cite this version:

Felix Baum, Arvind Raghuraman. Making Full use of Emerging ARM-based Heterogeneous Multicore SoCs. 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016), Jan 2016, TOULOUSE, France. hal-01292325

HAL Id: hal-01292325

https://hal.science/hal-01292325v1

Submitted on 23 Mar 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Making Full use of Emerging ARM-based Heterogeneous Multicore SoCs

Felix Baum
Embedded Systems Division
Mentor Graphics Corporation
Fremont, California USA
felix_baum@mentor.com

Arvind Raghuraman
Embedded Systems Division
Mentor Graphics Corporation
Fremont, California USA
arvind_raghuraman@mentor.com

***Abstract*— The complexity and pace of heterogeneous SoC architectures is accelerating at blinding speed. While these complex hardware architectures enable product vision, they also create new and difficult challenges for the system architects. Running and debugging an Operating System and application code on a single core is child's play. This is also true for running Synchronous Multiprocessing (SMP) capable Operating Systems on homogenous multicore processors.**

The modern day SoC combines asymmetric multiple cores, graphics processing units, offload engines and more on a single piece of silicon. This paper will discuss opportunities for system partitioning and consolidation, and some of the key issues and challenges of architecting, developing and debugging software on these complex systems.

Keywords*— *Multicore SoC, Synchronous Multiprocessing, Asynchronous Multiprocessing, AMP, SMP, ARM, Mentor Embedded Multicore Framework, MEMF, OpenAMP

I. Introduction

Heterogeneous multicore systems combining two or more different types of microprocessors (MPUs) and microcontrollers (MCUs) are quickly becoming the de-facto architecture in the embedded industry today. The quick emergence of these systems can be attributed to a number of factors, with one of the main one being consolidation. Over the last few years, there have been explosion of adoption in embedded designs of ARM cores. One of the benefits of using ARM is the ease with which SoC hardware designers can efficiently solve computing problems using systems of heterogeneous cores. Designers are able to allocate the right amount of compute for a given problem at the point that compute power is needed. And while Symmetric Multiprocessor (SMP) operating systems provide the infrastructure required to balance application workload symmetrically or asymmetrically across multiple homogeneous cores present in a multiprocessing system, in order to leverage the compute bandwidth provided by the heterogeneous processors present in the system, Asymmetric Multiprocessing (AMP) software architectures should be employed.

Asymmetric Multiprocessing architectures typically entail a combination of dissimilar software environments such as Linux®, a real-time operating system (RTOS), or bare-metal running on homogeneous or heterogeneous processing cores present in the SoC – all working in concert to achieve the design goals of the end application. Typical designs involve a software context on a master core bringing up a remote software context on a remote core on a demand-driven basis to offload computation. The master, remote processors, and their associated software contexts (i.e., OS environments) could be homogeneous or heterogeneous in nature. In order to effectively deal with the complexities of managing life cycle of several different operating systems on possibly dissimilar processors, and to provide an enabling Inter Processor Communications (IPC) infrastructure for offloading compute workload, new and improved software capabilities and methods are required.

The Multicore framework presented in this paper is not limited to work with ARM architecture or heterogeneous systems only. One can utilize these concepts with PPC or IA cores, but due to the popularity of the ARM devices in embedded space and emergence of SoCs with heterogeneous cores, we focus on those devises. This framework is a commercial implementation of the OpenAMP - an emerging API standard managed under the umbrella of Multicore Association.

To ensure that developers can efficiently solve compute problems with heterogeneous ARM SoCs, it is important to standardize some of the frameworks used to engineer various aspects of heterogeneous ARM systems. The Multicore Framework described in the paper is a software framework that provides two key capabilities to AMP system developers: 1) It provides the **remoteproc** component and API for life cycle management of remote processors and their associated software contexts to enable control the reset, load, execute and reboot states of the processors and cores; and 2) It provides remote messaging - **rpmsg** component and API for Inter-Processor Communications (IPC) between OS contexts in the AMP environment.

In Linux Operating system, when user wants to start, stop or execute another task, the remoteproc command is used. When one application needs to communicate with another application, they use the rpmsg APIs which are by now are part of the mainstream Linux and found in all current kernel distributions. The Framework hides the complexities of managing heterogeneous hardware and software environments providing a simplified application level interface to the user.

II. Origins

Compliance to open standards and adoption by the open source Linux community are important considerations when choosing an appropriate API for the multicore framework outlined in the paper. With these considerations, we choose the remoteproc and rpmsg API present in the Linux 3.4.x kernel and newer. The Linux remoteproc and rpmsg infrastructure was originally conceived and committed to the Linux kernel by Texas Instruments. The infrastructure allows Linux OS on a master processor to manage life cycle and communications with remote software context on a remote processor. However, the Linux provided infrastructure has some caveats; Linux rpmsg implicitly assumes that Linux will always be the master operating system and does not support Linux as remote OS in an AMP configuration. Further, the remoteproc and rpmsg APIs are available from Linux kernel space only – there is no equivalent API or library usable with other OSs and run-times. We developed a standalone library written in C language that provides a clean room implementation of the remoteproc and rpmsg functionality usable with RTOS or Bare-metal software environments, with API level compatibility and functional symmetry to its Linux counterpart.

The Multicore Framework is a standalone library written in the C language. It provides a clean room implementation of the remoteproc and rpmsg functionality usable with RTOS or bare metal software environments with API level compatibility and functional symmetry to its Linux counterpart. Figure 1.a shows a software stack diagram of the Multicore Framework and its usage in RTOS or bare metal environments. As shown, the Framework contains a well abstracted porting layer which consists of a hardware interface layer and an OS abstraction (environment) layer allowing users to easily port the Framework to other processors and operating systems.

Figure 1.b shows the remoteproc and rpmsg infrastructure present in the Linux kernel. The remoteproc and rpmsg drivers are kernel space drivers that provide services to the remoteproc platform driver and rpmsg user device driver. The remoteproc platform driver allows for remote life cycle management, and the rpmsg user device driver exposes IPC services to user-space applications.

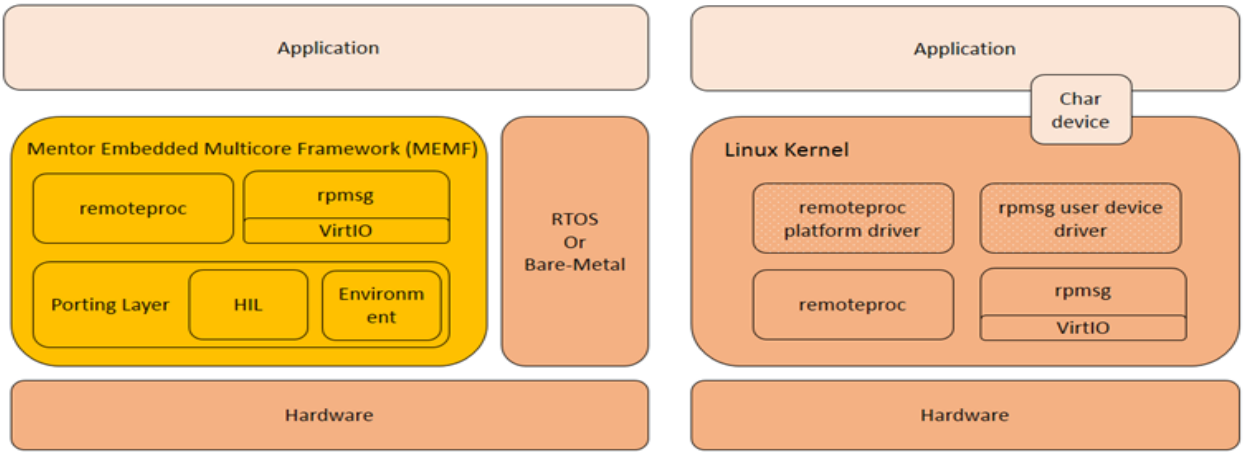

Figure 1: Multicore Framework in RTOS and Bare Metal Environments (a) and remoteproc and rpmsg in the Linux kernel (b)

In addition to enabling RTOS and bare metal environments to inter-operate with Linux remoteproc/rpmsg infrastructure in AMP architectures, the Multicore Framework provides workflows and runtime infrastructure to package and boot Linux as a remote OS in AMP configurations. Figure 2 shows the various AMP configurations supported by the Framework. In most of the diagrams of this article, the Multicore Framework is identified as MEMF or Mentor Embedded Multicore Framework.

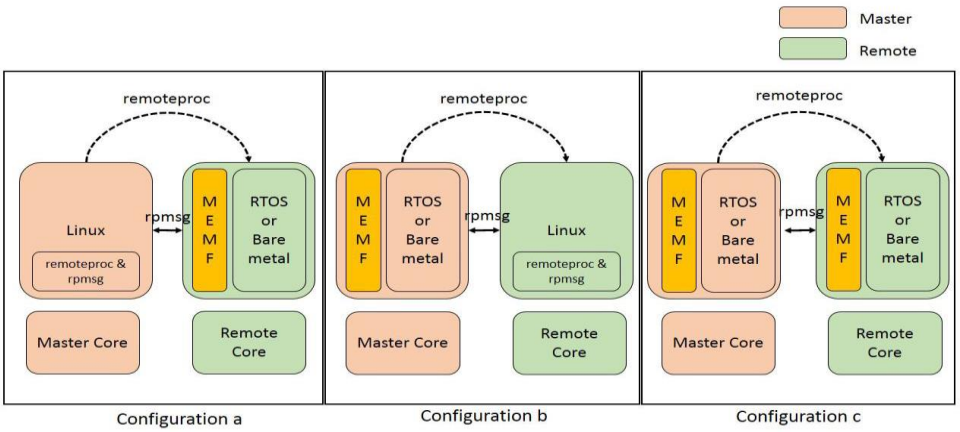

Figure 2: Multicore Framework supported AMP configurations

III. Use Cases and Appliations

The Multicore Framework is well suited for both un-supervised and supervised AMP architectures. To demonstrate the use cases below, a Xilinx MPSoC hardware platform will be used as it has 4 Cortex A53 cores and 2 Cortex R5 cores along with FPGA fabric.

Un-supervised AMP (uAMP) architecture is applicable to applications that do not require a strong separation between the participating operating system contexts. In this architecture, the participating operating systems run natively on the processors present in the system. As shown in Figure 3.a, the Multicore Framework provides a simple and effective infrastructure using which a master software context on a master (boot) processor can manage the lifecycle and offload computation to other compute resources present in the SOC in typical uAMP systems.

The Multicore Framework is well-suited for applications requiring demand driven off-load of compute functions to specialized cores present on a multiprocessing chip. In case of power constrained devices, the Framework enables on-demand bring up and shut down of compute resources allowing for optimal power usage.

The Framework also provides an easy path for consolidation of legacy uni-core based embedded systems onto powerful and more capable multiprocessing SoCs. With very little effort, the Framework allows for migration of legacy software originally developed for uni-core silicon to easily interoperate with enhanced system functionality developed on newer and more powerful multiprocessing chips.

Lastly, the Framework facilitates implementation of fault tolerant architectures. For example, the Framework can enable

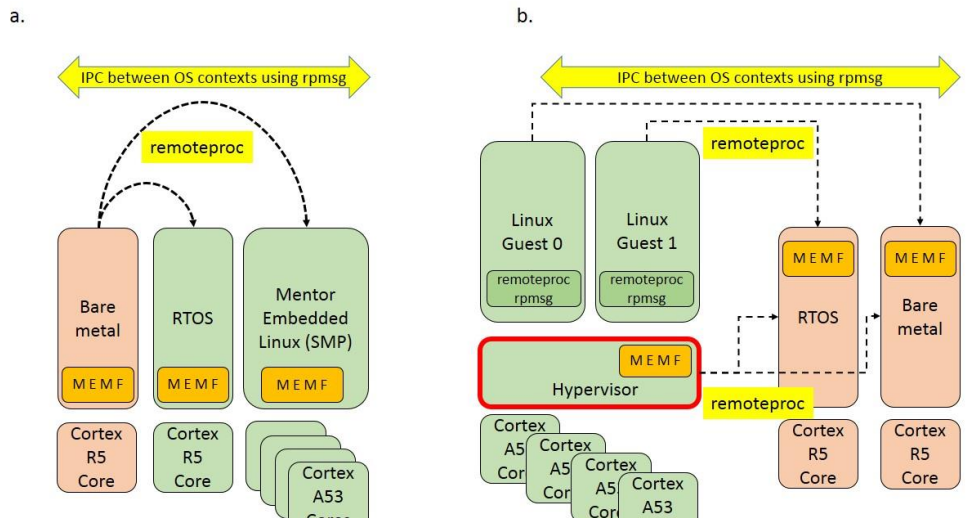

Figure 3: Multicore Framework use-cases: a) In uAMP architecture, and b) In sAMP architecture.

Supervised asymmetric multiprocessing (sAMP) architecture is applicable to applications that require isolation of software contexts and virtualization of system resources in an AMP system. In sAMP architecture, the participating guest operating systems run in guest virtual machines that are managed and scheduled by a hypervisor (aka virtual machine monitor). The hypervisor provides isolation and virtualization services for the virtual machines. The Multicore Framework enables sAMP architectures to manage computation on heterogeneous compute resources present in the SoC. As illustrated in Figure 3.b, the Framework can be used in two ways; 1) The Framework can be used from the guest OS context for un-supervised management of heterogeneous compute resources, and 2) Alternatively, the Framework can be used from within the hypervisor for supervised management of heterogeneous compute resources, allowing the hypervisor to supervise interactions between the guest operating systems and remote contexts involved.

a certified RTOS context (master) handling critical system functionality to manage a Linux context handling non-critical system functions. Upon failure of the Linux-based subsystem, the RTOS can simply re-boot the failed subsystem without causing any adverse effects to critical system functions.

IV. System Level Considerations in designing AMP Systems

Multicore Framework APIs provide the required software infrastructure to manage computation in AMP systems. However, in designing AMP systems, certain system level considerations must be taken into account before developing application software using the Framework APIs.

During the initial design phase the AMP topology is to be determined. The Framework can be used in a star topology – a single master managing multiple remotes, or in chain topology – with chained master and remote nodes as shown in Figure 4.a.

Once a suitable topology is chosen, the memory layout is to be determined. Memory regions should be assigned for each participating OS runtime, and shared memory regions should be assigned for IPC between the OS instances. Once the memory layout is finalized, the platform specific configuration data for the Framework should be updated to reflect the chosen memory architecture. Figure 4.b shows an example of a high-level memory architecture layout.

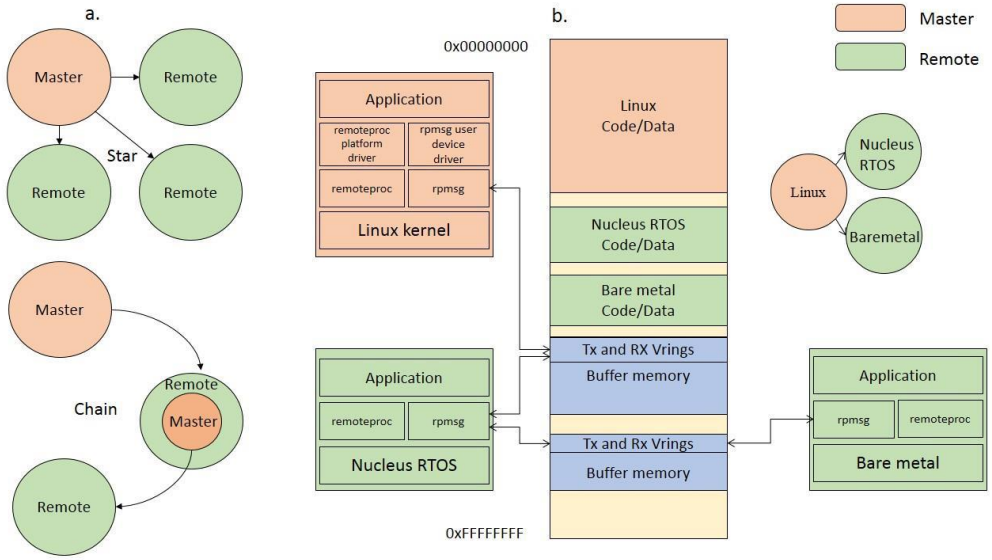

Figure 4: Designing AMP: possible topologies (a) and memory architecture (b).

Off-the-shelf operating systems generally assume they own the entire SoC, and are not readily suited for operation in unsupervised AMP environments where co-operative usage of shared resources and mutually exclusive usage of non-shared resources are key requirements. Each participating OS in an AMP system needs to be modified so that shared resources are used in a cooperative fashion. For example; the remote OS should not reset and re-initialize a shared global interrupt controller which could already be in use with respect to the master context, globally used timers cannot be reset, etc. These changes will typically require modifications to the OS kernel and/or BSP sources.

Further, system partitioning should be performed. System resources such as memory and non-shared IO devices available on the platform should be partitioned between the participating OSes so that each OS has visibility and access only resources that are assigned to it. This can be accomplished by modifying the platform-specific device and memory definitions provided to the participating OSes. For example, modify memory and device definitions in: a) Linux device tree source (DTS) file for Linux OS; b) platform definition file for Nucleus RTOS; and c) the platform-specific headers in bare metal environments.

V. Tools for Development of AMP Systems

Development of AMP application software presents a unique set of challenges. System developers typically find themselves in situations having to simultaneously debug several different OS environments deployed on dissimilar processors on heterogeneous SoCs.

Having a unified debugging environment with awareness of operating systems involved will not only enhance the debug experience, but improve productivity. As one of such tools, Mentor Embedded Sourcery CodeBench tools provide a unified IDE with OS awareness for all supported OS environments (including Mentor Embedded Linux, Nucleus RTOS, and bare metal). Sourcery CodeBench also supports a multitude of debug options which include: JTAG-based debug for debugging Linux kernel space, Nucleus RTOS kernel and applications, and bare metal contexts; GDB-based debug for Linux user space, and Nucleus RTOS based applications.

While developing AMP systems, software profiling is a valuable tool to gain insight into how various applications deployed on heterogeneous operating systems interact with each other during runtime. Each OS instance is typically based of an independent clock source, and any profiling data collected within a given OS context will be based on a time base that is local to the OS. Mentor Embedded Sourcery Analyzer host-based tools and Mentor's operating systems contain built-in algorithms that enable users to graphically visualize and analyze trace data collected from disparate OS sources in a unified time reference. This capability allows users to gain interesting insights into complex interactions, and hard to find timing issues typically encountered in developing AMP software.

VI. Related Work

While there are other approches to adressing multicore software chalenges, this particular way is a bit unique. Here is a short summary of some of them:

OpenCL™ (Open Computing Language) is the industry's open standard for writing data-parallel code in heterogeneous computers. AMD and Intel both promote OpenCL as a primary approach towards programming their parallel computing hardware offerings. OpenCL requires a similar level of low-level understanding and competence to write efficient parallel software. OpenCL is primarily targeted at leveraging data-parallelism of devices, and additional considerations must be made to use the multiple cores available on the CPUs in the system, for example offloading cpu intensive algorithms to GPUs. The multicore framework listed in this paper differs from OpenCL as it does not try to paralelize processing for the sake of performance but tries to isolate and separate execution blocks based on performance or power consumption requirements.

OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most desktop and server platforms, processor architectures and operating systems. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior. OpenMP preselects a set of base-language constructs, for example do-construct, as a basis for parallel computing. User identifies supported constructs, and manually inserts directives to assist compiler for a reconstruction of supported constructs into parallel. User does not need to create threads, and does not need to consider the work assigned to threads. OpenMP allows users who do not have a sufficient knowledge of parallel computing to explore parallel computing.

VII. Conclusion

The initial implementation of the Multicore Framework described in this paper was open-sourced under the OpenAMP open-source project with support for the Zynq 7000 SOC. OpenAMP as an emerging API standard managed under the umbrella of Multicore Association. This project is jointly maintained by Mentor Graphics, Xilinx and other software and hardware vendors. A current reference implementation of the proposed OpenAMP standard is available at: https://github.com/OpenAMP/open-amp. Mentor Embedded Multicore Framework (MEMF) is a proprietary implementation of the OpenAMP standard.

MEMF is tightly integrated with and readily supported by all Mentor provided OS run-times. It supports a diverse set of ARM based SOCs and platforms. Using MEMF with Mentor's tools and operating systems obviates users from having to design their AMP system from scratch. i.e., perform tasks discussed under the System level considerations section. Users can focus on AMP application development with one of the pre-canned reference configurations and later customize the system configuration to fit their needs.