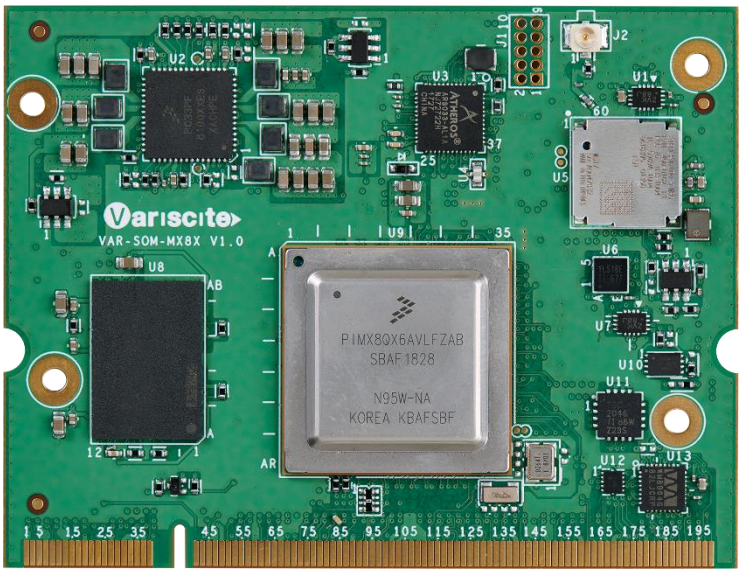
Asymmetric Multiprocessing on Heterogeneous Multiprocessor Systems

Implementation of asymmetric multiprocessing using a practical example on the i.MX8X with OpenAMP

Author: David Kauschke, Mixed Mode GmbH



Variscite

MIXED MODE
an Ingenics company

Asymmetric Multiprocessing on Heterogeneous Multiprocessor Systems

Implementation of asymmetric multiprocessing using a practical example on the i.MX8X with OpenAMP

Heterogenous multiprocessor systems on a chip (MPSoC) have become increasingly popular for industrial applications in recent years due to their high performance, lower cost and energy efficiency. In particular, the use of many different integrated processors running different operating systems poses multiple challenges. This architecture is also known as Asymmetric Multiprocessing (AMP). The two biggest challenges are lifecycle management (LCM) and inter-processor communication (IPC). This paper presents the design of heterogeneous MPSoCs and the use of different operating systems. A framework is selected as the proposed solution to the two challenges. This is followed by a presentation detailing the implementation of the developed AMP system with the selected OpenAMP framework on the NXP i.MX8X MPSoC with embedded Linux on the ARM Cortex-A35 and FreeRTOS on the ARM Cortex-M4, using Variscite's VAR-SOM-MX8X SoM. To evaluate the implemented system, the latency times on the i.MX8X are measured. The results show, among other things, that the maximum latency from Linux user space to FreeRTOS with the use of the RT patch is 628 µs. From the results, it can be concluded that the IPC between the processors is suitable for soft real-time.

From SoC to Heterogenous MPSoC

System-on-chip (SoC) refers to systems that consist of exactly one main processor and include several subcomponents such as I/O interfaces, memory, and graphics and audio components which are installed together on one chip. [1].

In contrast, an MPSoC is made up of at least two or more processors that are integrated together on one chip. Due to the large number of possible processors with their own variations, many different MPSoC architectures exist. The individual processors are classified according to their properties, such as processor architecture, instruction set, clock frequency or cache architecture. [2].

Fundamentally, MPSoCs are divided into two categories: homogeneous MPSoCs and heterogeneous MPSoCs.

Homogeneous MPSoCs consist of several processors with the same instruction set architecture. They offer high flexibility and scalability but are less energy efficient than heterogeneous MPSoCs. Typical applications for homogeneous MPSoCs are server systems, game consoles or cloud computing [1].

Heterogeneous MPSoCs, on the other hand, consist of several processors with different instruction set architectures suitable for targeted applications such as elaborate graphics applications or real-time critical applications. By dividing the tasks between different processors, a higher range of requirements is covered such as trade-offs between energy efficiency and performance, system safety, BOM reduction or EMC interference reduction [3].

Multiple Operating Systems Running on Heterogeneous MPSoCs

A combination of operating systems can be used on heterogeneous MPSoCs. They are subdivided into Symmetric Multiprocessing (SMP) and Asymmetric Multiprocessing (AMP). Both options are examined in more detail in the following sections.

Symmetric multiprocessing (SMP):

With SMP, exactly one operating system is executed on all available processor cores. All cores are subject to the same processor architecture as well as the same Instruction Set Architecture (ISA), share the same memory area and in part the same cache levels, and have common access to I/O devices. The operating system manages all resources and distributes the processes to be executed to the available cores, which are all treated in the same way. Any process can be executed on any core without any restrictions or the need for any changes. If the operating system is overloaded, all processes to be executed are equally affected, which means that deterministic behavior cannot be achieved on SMP systems. SMP systems use all cores of the given multicore architecture to achieve maximum performance. As a rule, a general-purpose operating system (GPOS) such as Linux is used, which is SMP-capable [4].

Asymmetric multiprocessing (AMP):

With AMP, more than one operating system runs independently on one or more processors with different ISAs. This is precisely where the use of heterogeneous MPSoCs becomes interesting, since they provide several different processors for specific tasks such as multimedia, real-time or image processing. For example, Linux can be used on all Application Processing Unit (APU) cores, while a Real Time Operating System (RTOS) runs on the ARM Cortex-M4 (Fig. 1). All operating systems share the publicly accessible memory, but at the same time typically still have their own private memory. All operating systems also share access to resources such as I/O devices. Mechanisms such as IPC, message passing, and shared memory enable the operating systems to communicate with each other and exchange data. Access and synchronization must be regulated in this case. In practice, SMP and AMP are often combined on an MPSoC to optimize energy consumption and achieve better performance. [4].

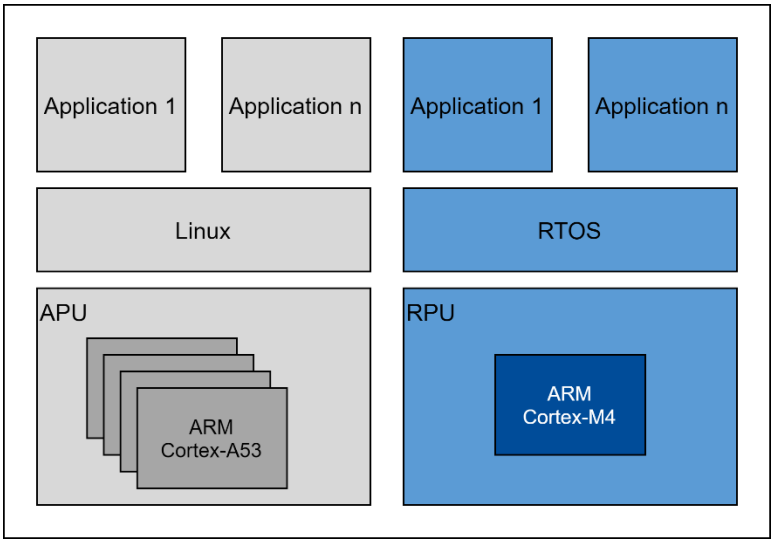


Fig. 1: possible AMP system structure. As shown in the figure above, a Linux-based system manages all APU cores (gray areas) with integrated ARM Cortex-A53s. An RTOS runs on the RPU with an integrated ARM Cortex-M4 (blue areas). In addition, AMP systems are divided into the two categories unsupervised and supervised AMP [5].

Challenges on AMP Systems

On heterogeneous MPSoCs, the use of multiple operating systems on different processors and the resulting synchronization effort leads to a degree of complexity that should not be underestimated. The challenges arising from this are examined in more detail in the following sections.

Memory management:

Memory management, i.e., the provision, allocation and separation of memory areas for subsystems (APU, RPU) must be taken into account. It must be ensured that the individual subsystems do not have access to the memory areas from the other subsystems. This could result in unstable system states or, in the worst case, cause the complete system to crash.

Resource management:

In addition, structured resource management must define which subsystems have access to which resources in which time window. If both subsystems access the same IO device at the same time, undefined behavior occurs.

System boot management:

Due to the many subsystems on heterogeneous MPSoCs, there is a large number of boot stages, which leads to greater system complexity. Therefore, it is important to clarify which subsystem is responsible for powering and configuring the necessary subsystems, including the clocks. In addition, a boot image is required that contains all the necessary firmware to start all the desired subsystems in the desired boot order at system startup. Generally, a dedicated subsystem is provided for this by the respective MPSoC, which takes over the responsibility for extracting the boot image, loading the firmware and starting the

processors. Within the subsystems, such as the APU, it must be clarified which boot loader stages are used. The loading of the device tree or Linux kernel must also be carried out. Furthermore, it must be ensured during system boot that the parallel boot processes of the individual subsystems do not block each other.

Life cycle management (LCM):

In addition to system boot management at system startup, LCM during runtime is also an important topic. It enables secure updates during runtime on individual subsystems such as the RPU without having to update the entire system. It is important here that different firmware can be loaded onto the subsystems or coprocessors without affecting other subsystems by this process. Accordingly, it must be possible to stop the coprocessor first before it can be restarted with new firmware. It is necessary to define which processor takes on the role of primary and which processor takes on the role of coprocessor.

Inter-processor communication (IPC):

As soon as several processors run within an MPSoC, an IPC is essential to exchange data between them. This is usually done via shared memory as well as inter-processor interrupts (IPI). Via an IPI, one processor can notify another processor of new information in shared memory. On an MPSoC, messaging units (MU) are used for this purpose. The IPI used must be registered and enabled with the shared interrupt controller, which is integrated as a hardware component on the MPSoC.

Selecting a Framework for LCM and IPC

For the heterogeneous MPSoCs from the manufacturers NXP, Xilinx or STM, no standardized solutions are currently offered for the challenges listed. As a result, the challenges must be considered individually by platform and solved with the solution approaches provided by the platform. However, several frameworks have been developed in recent years that at least attempt to solve the LCM and IPC across platforms. In the following sections, therefore, these frameworks are presented and one of them selected for closer consideration.

OpenCL, Cilk, OpenMP and MCAPI:

The frameworks OpenCL, Cilk, openMP and MCAPI can be found in specialist publications when looking for a solution for LCM and IPC. However, these frameworks focus on the parallelization of tasks. The tasks are executed in a distributed manner on multiple cores on one or more MPSoCs. This results in high-performance and efficient processing of the tasks. This is also referred to as massively parallel processing (MPP) [6–9].

OpenAMP:

The Open Asymmetric Multi-Processing (OpenAMP) framework offers functionalities for the LCM and for the IPC on AMP systems and can be used on different operating systems such as Linux, RTOS or bare metal. For Linux the existing kernel components Remote Processor Framework (remoteproc) and Remote Processor Messaging (RPMsg) are used, which are contained in the mainline kernel (version 3.4.x or later). Remoteproc takes over the tasks of the LCM and RPMsg is responsible for the IPC. On RTOS and bare-metal, the framework consists of a hardware-dependent porting layer that provides the necessary hardware interfaces and operating system environments (Fig. 2) [10–12].

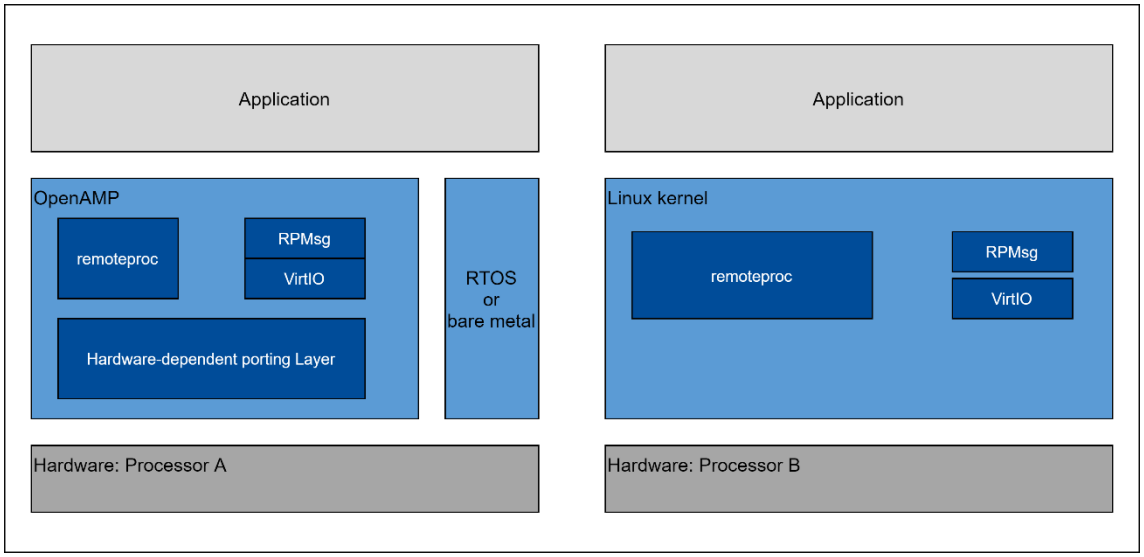


Fig. 2: illustration of the OpenAMP framework on Linux, RTOS and bare metal.

The boxes in the figure above are divided from top to bottom into the following layers: application (light gray), middleware and operating systems (blue) and hardware (dark gray). The left side illustrates the hierarchical structure of OpenAMP on an RTOS/bare metal. It consists of the platform-independent components remoteproc and RPMsg and the hardware-dependent porting layer. The structure for Linux is illustrated on the right. It consists of the same remoteproc and RPMsg components (redrawn from [13]).

In addition to the OpenAMP open-source solution approach, the company Mentor Graphics offers the commercial solution Mentor Embedded Multicore Framework (MEMF) for OpenAMP. However, since an open-source approach is followed, the Mentor Graphics framework is not considered for use on the i.MX8X.

RPMsg-lite:

The RPMsg-lite approach developed by NXP focuses on the IPC and uses RPMsg, like OpenAMP. LCM is not supported in this approach. The special feature is that the extensive API of RPMsg has been simplified to reduce the required memory and complexity. As a result, RPMsg-lite focuses primarily on processors that have little memory integrated, such as the Cortex M0+. In addition, the user has the option to set RPMsg-lite as a static or dynamic library and to use a zero-copy mechanism for copying the data between the application and the RPMsg device. RPMsg-lite minimizes the latency that occurs when copying the messages. The required flash memory with RPMsg-lite is 2926 bytes when using the static library, the required RAM memory is 352 bytes. In comparison, OpenAMP requires 5547 bytes of flash and 456 bytes of RAM, plus additional allocation in dynamic memory. [14].

RPMsg-lite or OpenAMP?

The advantage of RPMsg-lite is that it is ready for use on the deployed i.MX8X MPSoC and does not involve any increased implementation effort. NXP already offers the necessary components for Linux to communicate via RPMsg with the M4, which uses RPMsg-lite. There are also demo applications on the M4 side that include the RPMsg-lite approach. The disadvantage of this solution is that it does not include the

component for the LCM, as mentioned above. It also lacks functionality for providing a proxy infrastructure. For these reasons, OpenAMP is used.

Introducing the MPSoC i.MX8X

System architecture i.MX8X

The MPSoC i.MX8X integrates different subsystems that communicate with each other via the AMBA bus. Each integrates a quad-core ARM Cortex-A35 for high-performance applications and a single-core ARM Cortex-M4 for real-time critical applications. The integrated GPU or VPU processors are available for graphical applications. Since this paper deals with the AMP between the A35 and M4 processors, the subsystems for the graphical applications are not considered in detail here. The memory system is composed of a 256 KB on-chip OCRAM and a 4 GB off-chip LPDDR4. In addition, a memory area with up to 64 GB is provided by an eMMC interface or by a MicroSD card. For external communication, a range of interfaces including for JTAG, USB, GPIO, Bluetooth and CAN-FD are available. The System Controller Unit (SCU) handles, among other things, the management of access to peripheral devices [15].

The SCU consists of an ARM Cortex-M4 and an ARM Cortex-M0+ for the integrated security controller (SECO). It introduces an additional abstraction layer for access to the underlying hardware on the i.MX8X platform. The SCU has exclusive direct access to all subcomponents of the entire platform. If subsystems like the APU or RPU directly access a subsystem such as the DDR area without the necessary rights, access is denied and an error message is generated. The SCU is responsible for the following tasks: system initialization and boot process, system controller communication, power management service, resource management service, pad configuration, timer, interrupts and security [16].

The following section defines the terms “resource” and “partition”, which are part of the resource management service and are relevant for the subsequent sections. The other tasks are described in more detail in [16]. Note that due to their interdependency, it is necessary to read the definitions for both terms to fully understand the usage in the context of this paper.

Resource Management Service:

“Resource”: as previously mentioned, read and write access to memory and peripheral units must be managed correctly to avoid unexpected behavior. The concept of the resource is introduced for this purpose. A resource is an Intellectual Property (IP) function block, which can be memory or a peripheral unit. So, a resource can be a GPIO, an MU or a memory area. Each resource has exactly one owner, provided by one partition. A partition can provide other partitions with access to the resources, or pass ownership of the resource to another partition. At system startup, all resources are assigned to the permanent partition of the SCU.

“Partition”: In the context of the i.MX8X platform, a partition is not defined as the division of a hard disk into different areas, but rather the allocation and division of resources to individual partitions. Partitions can be created or deleted at boot time and during runtime. Each partition has a parent partition when it is created, which is capable of booting, rebooting or shutting down the child partition. This principle provides important functionality for the LCM. During system runtime, access to resources can be changed. This feature allows

multiple partitions to have access to the same resource, for example, to define a shared memory area. It is also possible to transfer a resource to another partition during runtime.

The MMAMP System Design from OpenAMP on the i.MX8X

The system design for the AMP system in development by Mixed Mode, Mixed Mode AMP (MMAMP), is derived from the deployed OpenAMP framework and the architecture of the i.MX8X platform described. In addition, the existing software components are used in the Linux kernel provided by the MPSoC manufacturer NXP in order to reduce the implementation effort and to ensure reusability of the system design.

The system design is composed of three main components (white, Fig. 3), which in turn are built from several components. The first two main components are located in the Linux system which runs on the APU and is divided into user space and kernel space. The third component is on the RPU and is the firmware for the M4.

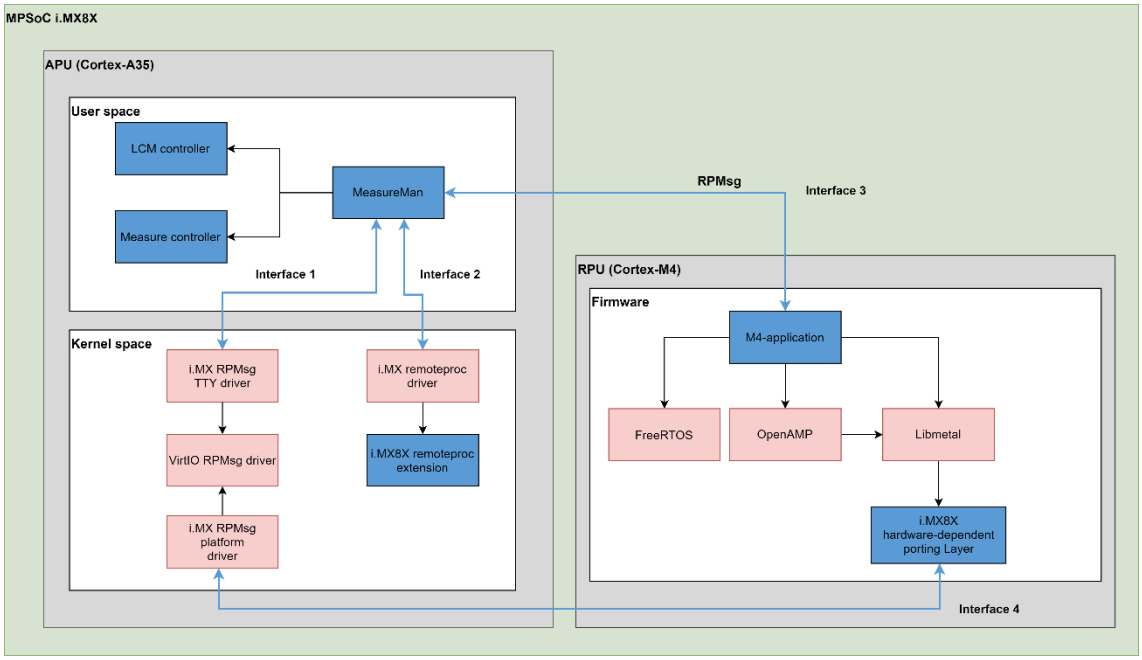


Fig. 3: MMAMP: AMP system design.

In Fig. 3, the integrated processors Cortex-A35 and Cortex-M4 (represented by the gray boxes) on the MPSoC i.MX8X (green area) run the software systems that are built on them (represented by the white boxes). The Cortex-A35 runs a Linux setup that is divided into user space and kernel space (white boxes). The Cortex-M4 runs firmware, which again consists of several components (white boxes). Within the white boxes, components are shown hierarchically in blue and red. Red components are existing components and blue components are Mixed Mode's own contributions. The black arrows represent the dependencies between components. Blue arrows present the interfaces between the components.

MeasureMan component in user space:

The MeasureMan component is located in user space and has three central tasks. Its first main task is to control the LCM and it therefore takes over the role of the master processor. It provides the i.MX remoteproc driver with the firmware to be loaded for the Cortex-M4 in the file system and provides information about a newly downloaded firmware via the virtual file system Sysfs (interface 2). The second main task is the IPC via RPMsg between user space and the coprocessor using interface 3, which ensures the communication flow between the two processors. The data flow of the exchanged messages is provided by the i.MX RPMsg Teletypewriter (TTY) driver using a TTY RPMsg interface (interface 1). The third main task is the control and execution of measurements, which are necessary for evaluating the latency times.

Remoteproc and RPMsg components in the kernel space:

The components in the kernel space are divided into two areas. The first area provides the functionalities for the IPC via RPMsg and consists of the following components:

1. The i.MX RPMsg TTY driver is responsible for communication between the user and kernel space and is included in the kernel as a loadable kernel module. It provides a serial TTY RPMsg interface which is used by MeasureMan to send messages to and receive messages from the M4 (interface 1).
2. The VirtIO RPMsg driver takes over the management of the VirtIO devices, which are built from Virtqueues and VRings [11].
3. The i.MX RPMsg platform driver creates the necessary VirtIO devices and registers them with the VirtIO RPMsg driver. In addition, the IPC runs on this driver via IPIs and shared memory using two MUs (interface 4).

The second area provides the functionalities for the LCM:

1. The i.MX remoteproc driver provides the functionalities for the LCM. After a more detailed analysis of this component, it was found that it only supports the MPSoC i.MX6SX and i.MX7D. As a result, this driver cannot be used for the i.MX8X platform because it does not support mechanisms for partitions and resources. For this reason, this driver must be extended so that it can be used for the i.MX8X platform.
2. The i.MX8X remoteproc extension implements and respects the i.MX8X concept of partitions and resources. This allows the LCM to be used on the i.MX8X via the i.MX remoteproc driver. The extension is implemented within the i.MX remoteproc driver.

The precise implementation of the presented system design with the porting of OpenAMP can be found in the MSc thesis [17].

Evaluating Latency Times

Having presented the system design for the MMAMP system, an overview of the measurement system used to measure latency on the i.MX8X between the Cortex-A35 and Cortex-M4 processors follows.

Overview of the measurement system:

The tests carried out refer to the measuring sections 1-4 shown in the system overview (Fig. 4):
IPI latency from the A35 (i.MX RPMsg platform driver) to the called ISR in the M4 (measurement section 3 to 4).
IPI latency from the M4 to the A35 (i.MX RPMsg platform driver) to the called ISR (measurement section 4 to 3).
Latency from the MeasureMan application in user space to the triggering of the IPI 3 i.MX RPMsg platform driver via the TTY-Write interface (measurement section 1 to 3).
Latency from the i.MX RPMsg platform driver to the MeasureMan application via the TTY-Read interface (measurement section 3 to 1). It is particularly important to note that this measurement includes M4 processing time and IPI latency.
The processing time on the M4 caused by the OpenAMP (measurement section 4).

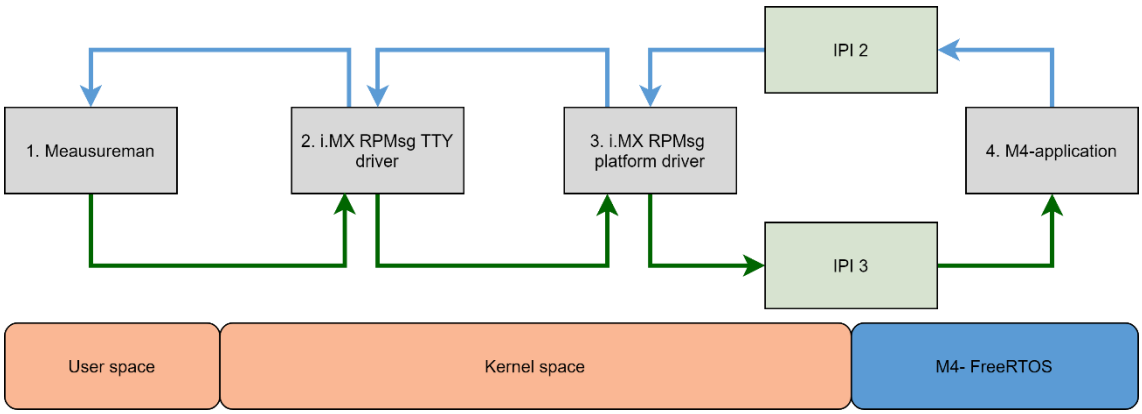


Fig. 4: system overview of the tests to be performed. This shows the affected components for the individual tests.

The overview is divided into four components: the MeasureMan, the i.MX RPMsg TTY driver and the i.MX RPMsg platform driver running on the A35, and the M4 application running on the M4 (gray). Both processors communicate with each other via the two IPIs 2 and 3 (green). The green arrows represent the communication direction from Linux, which is divided into the user and kernel space (orange), to the FreeRTOS on the M4 (blue). The blue arrows represent communication in the opposite direction.

Details on the measurement methodology and the performance of the individual tests can be found in [17].

Summarizing the Measurement Results - Is Real Time Possible?

Table 1 summarizes the most important results for a message size of 496 bytes (maximum message size of an RPMsg) and a sample number of 4,000,000 measurement points. The tests performed show an overview of the latency times between the two processors. Here, the kernel used was already extended with the RT patch, since outliers of up to 125,472 µs occurred in the TTY-read latency without the RT patch. By using the RT patch, the maximum latency times for the TTY-read measurement could be reduced to 55.291 µs.

Message size: 496 bytes	**TTY-write**	**TTY-read**	**IPI- latency to M4**	**IPI- latency to A35**	**OpenAMP processing time**
Max. [µs]	**628**	**55291**	**809**	**581**	**365**
Min. [µs]	10	4	3	4	190
Arith. mean [µs]	15	278	4	5	198
Standard deviation [µs]	5	108	1	2	1
99.9 % quantile [µs]	50	942	4	11	200

Table 1: Summary of the latency and processing times measured for a message size of 496 bytes.

The high number of outliers is due to the non-deterministic Linux system. The Cortex-M4 can be excluded as the reason for this since it guarantees a maximum interrupt latency of 12 clock cycles.

Instead, the statistical outliers could possibly be attributed to a hardware problem. For example, this could be due to the system bus of the MPSoC. Another cause could be the memory alignment, meaning that to access data it may be necessary to access memory several times and therefore constant access to memory cannot be guaranteed.

The tools TraceCompass or Traceshark can be useful to perform an analysis and identify the cause of the outliers.

The IPC between the processors via OpenAMP is not suitable for the use of hard real-time. The use for soft real-time, on the other hand, is plausible if the outliers shown (Table 1) are within the tolerance range.

Discussion

When porting OpenAMP to the ARM Cortex-M4, the separation of the platform-dependent part by the hardware-dependent porting layer within the Libmetal library has proven to be a great advantage.
A ready-to-use RPMsg implementation is already available for the running embedded Linux on the ARM Cortex-A35. It was only necessary to implement the missing platform-dependent remoteproc driver for the LCM in the kernel.

However, only one use-case was implemented for this paper and the full potential of OpenAMP was not exploited. Therefore, the use of RPMsg-lite would have been sufficient for the evaluation of the IPC. However, due to the lack of functionality for the LCM, OpenAMP was used rather than RPMsg-lite. For applications that only require IPC between processors without LCM, RPMsg-lite is the better choice. However, it must be noted that this solution can only be used for the MPSoCs provided by NXP.

In addition, the high complexity of the i.MX8X MPSoC leads to a high learning curve due to the many different integrated subsystems. For example, the existing SCU brings hardware-dependent functionalities for controlling system boot, memory and resource management, which are based on resources and partitions. This also requires considerable time and effort for familiarization with the solution, irrespective of the MPSoC used, since each platform provides its own special features and the manufacturers do not follow a common approach.

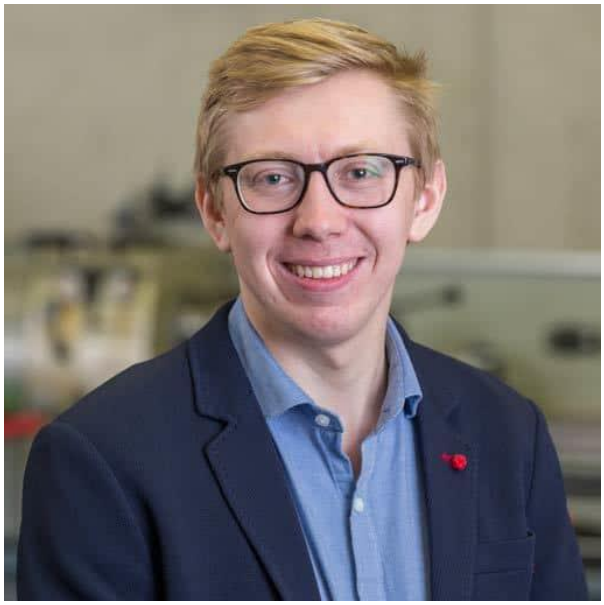
The evaluation results for OpenAMP on the i.MX8X show high latency outliers for which only hypotheses for the causes were derived. The causes were not considered in more detail in the context of this paper and only approaches for investigation were suggested. Further investigations should be carried out for this.

Bibliography and Sources

[1] Michael Hübner and Jürgen Becker, *Multiprocessor System-on-Chip*. Springer Science & Business Media, 2011.
[2] Mpsoc Technology et al., *Multiprocessor System-on-Chip*. In: IEEE Transactions on Multi-Scale Computing Systems 27.10 (2008), page 1701–1713.
[3] Alexandra Fedorova et al., *Maximizing power efficiency with asymmetric multicore systems*. In: Communications of the ACM 52.12 (2009), page 48–57.
[4] Bryon Moyer, *Real World Multicore Embedded Systems*. Newnes, 2013
[5] Louise Crockett et al., *Exploring Zynq MPSoC: With PYNQ and Machine Learning Applications*. Strathclyde Academic Media, 2019.
[6] Suyang Zhu et al., *Exploring task parallelism for heterogeneous systems using multicore task management API Paper*. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 10104 LNCS (2017), page 697–708.
[7] Lauri Matilainen et al., *Multicore Communications API (MCAPI) implementation on an FPGA multiprocessor*. In: Proceedings - 2011 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, IC-SAMOS 2011 July (2011), pages 286–293.
[8] Peng Sun, Sunita Chandrasekaran and Barbara Chapman, *OpenMP-MCA: Leveraging Multiprocessor Embedded Systems Using Industry Standards*. In: Proceedings - 2015 IEEE 29th International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2015 May (2015), page 679–688.
[9] Jerry L Potter, *The Massively Parallel Processor*. Cambridge, MA, USA: MIT Press, 1985.
[10] Multicore Framework - Mentor Graphics. url: https://www.mentor.com/embedded-software/multicore
[11] OpenAMP/open-amp. url: https://github.com/OpenAMP/open-amp
[12] Linux 3.4 - Linux Kernel Newbies. url: https://kernelnewbies.org/Linux_3.4
[13] Warren Kurisu, *Implementierung eines Multicore-Frameworks* — All-Electronics. 2017. url: https://www.all-electronics.de/multicore-framework/
[14] NXP, *NXPmicro/rpmsg-lite: RPMsg implementation for small MCUs*. url: https://github.com/NXPmicro/rpmsg-lite
[15] NXP Semiconductors, *i.MX 8DualX/8DualXPlus/8QuadXPlus Applications Processor Reference Manual*. 2019.
[16] NXP Semiconductors, *System Controller Firmware Porting Guide*. 2019
[17] David Kauschke, MSc thesis. url: https://github.com/kauschked/ma_openamp

Author

David Kauschke works as a software developer at Mixed Mode GmbH in Gräfelfing near Munich. He completed his MSc degree in computer science with a focus on embedded systems at the University of Applied Sciences in Munich. His work focuses on the development and implementation of applications on heterogeneous multiprocessors with an emphasis on the Linux side. While completing his MSc thesis at Mixed Mode, he ported the OpenAMP framework to the heterogeneous MPSoC i.MX8 from NXP.



Contact

Email: david.kauschke@mixed-mode.de
LinkedIn: https://www.linkedin.com/in/david-kauschke-a530bb89/

Variscite Ltd. Headquarters

4, Hamelacha Street
Lod
P.O.B 1121
Airport City, 70100
ISRAEL

Contact Us

+972 (9) 9562910
+972 (9) 9589477
www.variscite.com

Sales: sales@variscite.com