Architecture and Implementation of Micro-ROS with OpenAMP on an Heterogeneous Multi-core Processor

Vincent Conus
Dpt. of Mechanical Engineering and System Control
Nanzan University, Nagoya, JP
vincent.conus@pm.me

Shinya Honda
Dpt. of Mechanical Engineering and System Control
Nanzan University, Nagoya, JP
shonda@nanzan-u.ac.jp

Shinkichi Inagaki
Dpt. of Mechanical Engineering and System Control
Nanzan University, Nagoya, JP
inag@nanzan-u.ac.jp

Abstract— Integration of a variety of systems on a chip has become possible in recent years, making heterogeneous multi-core processors (HMP) available as development targets. In this article, the implementation and deployment of the Robot Operating System (ROS) and micro-ROS on an HMP is presented, as these are very popular choices as middleware in robotics, automotive and beyond. We are focusing on the architecture of the system and the use of OpenAMP shared-memory system as a mean of communication as well as on the early result in data transfer speed improvement compared to communication using a serial bus.

I. Introduction

The integration of components and functions for embedded system brings notable advantages in performance and energy consumption[1]. As Heterogeneous Multi-core Processors (HMP) devices are becoming available, both a general purpose application processing unit (APU) and a real-time capable micro-controller unit (MCU) can be used on a single System on Chip (SoC). The gain in performance and reduced parts number comes with an increased cost in complexity for the firmware deployment as the MCU is no longer a separated device, but rather a component that needs to be setup and accessed from within the main chip.

In the context of robotics development, running the Robot Operating System (ROS) on the APU of an HMP is possible. ROS and its sequel ROS2 are popular middleware, its network system being particularly powerful, allowing for robust discovery and data transmission from a Linux system.

Micro-ROS, on the other hand, is an implementation of the ROS2 communication mechanisms that is meant to be deployed on a MCU. This new firmware makes real-time devices able to join the ROS network using system such as UART, Ethernet or CAN to communicate. Micro-ROS needs a node called "Agent" in the ROS2 side in order to communicate with it.

Running micro-ROS on an HMP is the natural next step in the described context. As ROS2 is capable of running on any APU, it would make sense to run micro-ROS on an MCU on the very same chip. The challenge, however, is the data transport: we need to find a channel through which ROS2 and micro-ROS can communicate. Propositions exist but they all have trades-off. Virtual Ethernet requires TCP/IP which has a large overhead, in particular for the MCU.

UART requires external controllers, a cable and will cancel out the speed advantage of using intra-chip communication. For this publication we focused on using the Open Asymmetric Multi-Processing (OpenAMP) framework, communicating through shared-memory, potentially at the expense of flexibility but with the promise of great data throughput.

Compared to the other systems, OpenAMP is integrated in the board operating system, making it potentially portable to many HMPs. The communication passes through the shared memory with a system called Remote Processor Messaging (RPMsg). Using Remote Processor Framework (`remoteproc`) [2], the MCU can be accessed as a device from the host Linux.

OpenAMP used as a channel between ROS2 and micro-ROS has been tried in another project [3] but only on a custom architecture. With very little details on the exact implementation as well as no information at all regarding the performances of their system, it was difficult to get anything useful for our own project from this implementation.

In this project, a way for ROS2 and micro-ROS to be deployed together on an HMP is proposed. While running on the same silicon chip, the goal is to make it possible to exchange data over OpenAMP's RPMsg system. The architecture presented here should be the first of its kind to be deployed on a commercially available platform (Xilinx KRIA boards), with a fully described architecture as well as some communication speed testing. In the implementation detailed here, a basic, proof-of-concept communication was archive, showing the feasibility of the imagined architecture, acknowledging the general requirements and giving the first real-life data regarding transfer performances.

Most critically, this projects main contributions are:

- An **architecture proposition** for running micro-ROS on an HMP with OpenAMP as the transport layer.
- The distribution of a **deployable template** that can be utilize to run ROS2 and a micro-ROS instance as a pair on an HMP.

Implemented in robotics, this system allows to have a more direct access to the peripheral devices without scarifying the real time capability of a true Real Time Operating System (RTOS).

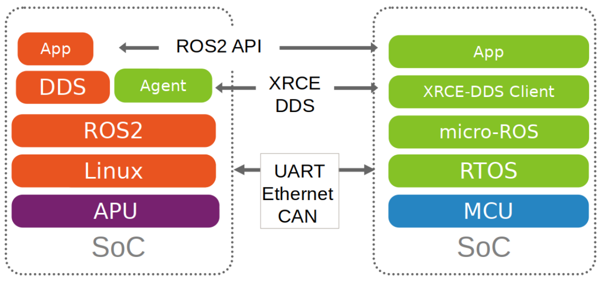


Fig. 1. General architecture for a ROS2 and micro-ROS system.

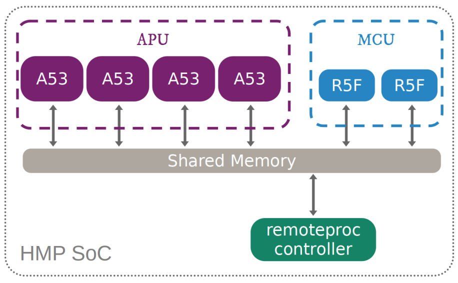


Fig. 2. Xilinx's ZynqMP HMP architecture used in this project.

II. Background

A. *ROS2, DDS & micro-ROS*

The robot operating system (ROS or ROS2) is a commonly used middleware in the world of robotics. It allows for a convenient access to development tools, libraries, simulation utilities and API as well as to a topic-based communication protocol between the components of the ROS system (nodes). This Data Distribution Service (DDS) protocol[4] is a central piece for our proposed deployment of this paper, in particular regarding its great discoverability and communication reliability with nodes.

While ROS and ROS2 systems are typically deployable only on top of a general purpose operating system (Linux), the recent micro-ROS project gives ROS2 communication capabilities for micro controllers, allowing real-time capable devices to utilize the messaging system and be part of a ROS2 environment.

Fig. 1 shows an architecture for a typical deployment of a ROS2 and a micro-ROS system. In this configuration, the computationally heavy control and vision tasks can be deployed on ROS2 on the Linux side (APU). The real-time sensitive applications can be run on top of a RTOS on the micro-ROS side (MCU).

On the left of the figure, the ROS2 stack can be seen, with a node called the "Agent". It's purpose is to make the communication with the micro-ROS system possible. The agent makes the micro-ROS entity a part of the global DDS network.

On the right, we have the micro-ROS system running on top of RTOS on a separated device. Real-time systems can be deployed here as RTOS tasks are running in pseudo-parallel with the micro-ROS system.

These tasks are now able to communicate with the ROS world with an API. Both devices communicate using typically UART or other serial processes, CAN or Ethernet.

B. *HMP*

An HMP, as viewed in example schematic at Fig. 2, has the particular property of having multiple cores of different types, architecture and capabilities within the same chip. This is by opposition to the "traditional" CPUs, with a single type of core.

This gives more options to the developers to use the best suited cores for their applications (APU, MCU, DSP or even FPGA), as well as making the communication between the components potentially much faster.

This type of SoC are becoming increasingly popular, with a wider range of manufacturer producing them for industrial application, such as i.MX, R-CAR or ZynqMP.

C. *OpenAMP & RPMsg*

OpenAMP is an open-source framework meant to simplify the communication for components in an HMP.

It is integrated into the Linux kernel since at least the version 4.11. This makes OpenAMP available for every HMP boards running Linux, and an ideal choice for an intra-chip communication framework.

Using this frameworks RPMsg, receiving messages between the cores of supported devices becomes possible through shared-memory.

This messaging system allows for the Linux user-space to gain access to low-level memory for communication through a device [5].

III. Implementation

A. *Motivations & requirements*

As for the motivation on pursuing the deployment of micro-ROS on a HMP, we want to show an example and provide a way to use them in a ROS environment. The hardware is available; we might as well use it.

The promise of having intra-chip memory being used for data transfer between nodes is exiting. It gives a high sealing on what the data transfer performance can be.

Hard real-time capability is a critical component for some types of autonomous system and we think that proposing a full, separated RTOS stack available for ROS on the same SoC will power a lot of development.

In addition to the presented motivations, those requirements were determined:

- **Using a standard communication layer**. We want to keep the system portable to other targets, other HMP boards; without the need to maintain a new system over the evolution and new versions of the underlying systems such as Linux or ROS2.
- **No modification of the software stack** (ROS2, Linux modules, micro-ROS agent, micro-ROS library) except for the transport layer. The reasoning here is the same as for the first point: the use of the new communication layer should be transparent for the other components of the overall ROS2 system.

B. *General architecture*

The usual setup for using both ROS2 and micro-ROS for robots provides number of benefits, from unified communication to access to a standardized development platform. A proposed setup for the deployment on an HMP can be seen in Fig. 3, which can be compared with Fig. 1, running the same system on two devices.

Multiple adaptations are required to make this integration possible.

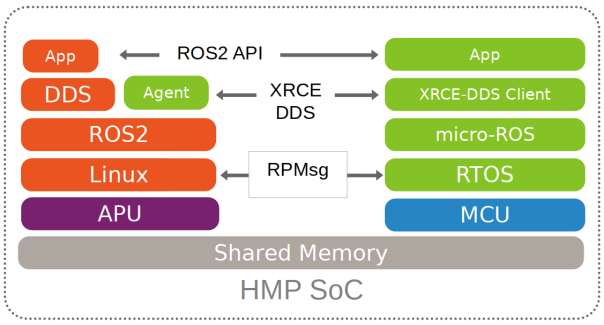


Fig. 3. Top-level view of the proposed architecture for this project.

While a functional RPMsg communication is obtainable without major difficulties, as shown in the official documentation, modifications of both the ROS2 Agent and the micro-ROS Client are needed in order for the systems to know and use RPMsg as a protocol to transfer DDS messages in both directions.

Similarly to what is seen on Fig. 1 in the previous section, we have a ROS2 stack (in orange) and a micro-ROS system (in green), but running on the same SoC. In this setup, the ROS2 Agent on the left needs to be accommodated to support RPMsg as a transport layer.

C. *Agent*

In a ROS2 setup meant to communicate with a micro-ROS system, the Agent is a key component that allows to transmit back and forth the data packets in a way that is compatible with DDS, making the micro-ROS device able to subscribe and publish topics to the overall ROS2 ecosystem.

Fig. 3 shows the position of this critical element in the global structure. As for the naming system, this component will often be referred to as "micro-ROS Agent", but it must be clear that the Agent is a C++ ROS2 node deployed on Linux, not a part of the RTOS micro-ROS deployment.

The current implemented architecture of this system was made by modifying the code of the default serial agent and is visible in Fig. 4.

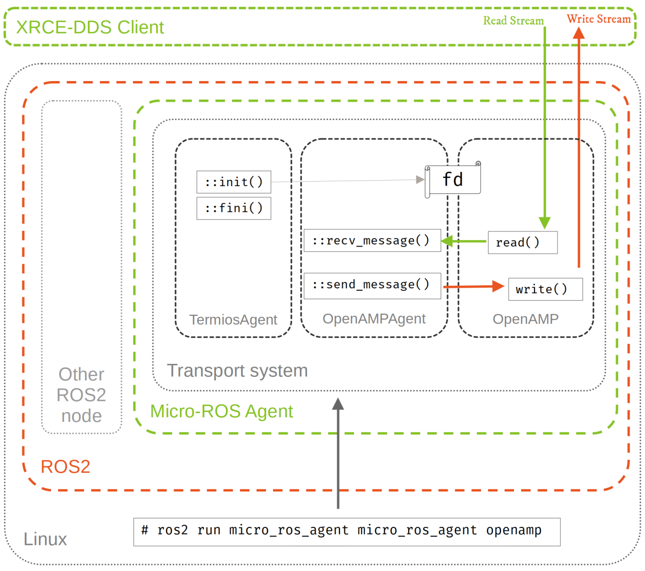
A key aspect was to keep the change as reduced as possible, thus only the transport system was modified to be able to use OpenAMP.

In the current state of the system, the `TermiosAgent` and the `OpenAMPAgent` are both classes whose methods can be used by the general micro-ROS Agent as the transport layer. The `TermiosAgent` class includes the `::init` and `::fini` methods that will, respectively, create and destroy the file descriptor (`fd`) used to access the RPMsg device.

The setup of this file descriptor is not trivial as it requires extensive configuration for the RPMsg communication. This includes an initial handshake with the MCU before the initialization phase of ROS.

The `OpenAMPAgent` contains the `read` and `write` methods themselves, allowing `RPMsg` to be used as a ROS transport. Little modification was required in this class as most of the setup is happening during the init phase.

The code for the agent is freely available [6], as a fork of eProsima's own source.



```
# ros2 run micro_ros_agent micro_ros_agent openamp
```

Fig. 4. Software architecture of the micro-ROS Agent.

D. *Client*

Fig. 5 shows the data flow for both incoming and outcoming data packages for the micro-ROS client. The various read and write functions needed to be implemented within the two main FreeRTOS tasks to communicate with ROS are also visible.

The data reader stream can be seen in orange at the top of the figure, with the stream buffer making the link between the `RPMsg` callback and the micro-ROS functions. The data writer function is in green at the center. The reason of being for these two tasks is the fundamental incompatibility between what micro-ROS expects a 'read' function to be and the way RPMsg manages the same functionality.

In micro-ROS, we need to have four (`init`, `read`, `write` and `close`) function pointers that can be passed to the various ROS2 methods.

In the other hand, the `read` function in RPMsg can only be a callback with a specific format, including the necessity to deal with the interrupt routine service (IRS).

This lead to the creation of a message buffer that bridges both worlds: the buffer can store one or more message from the callback function, waiting for the micro-ROS `read` function to claim it.

It is to be noted that this only applies to the `read` function, as the `write` function from the micro-ROS setup can directly use the `RPMsg` function.

Fig. 6 shows the pseudo-parallel aspect of the micro-ROS firmware tasks.

- The black `default task` is simply the `main()` function starting the `micro-ROS` task.
- The red `micro-ROS task` will then make further setup and create the `RPMsg task`.
- The green `RPMsg task` will then setup the shared memory configuration, before both tasks wait for something to be received from another ROS node.

To implement this Client, the example firmware for running OpenAMP on the ZynqMP SoC was used as a base, on top of which a statically compiled micro-ROS library was deployed.

The source for the client is freely available [7].

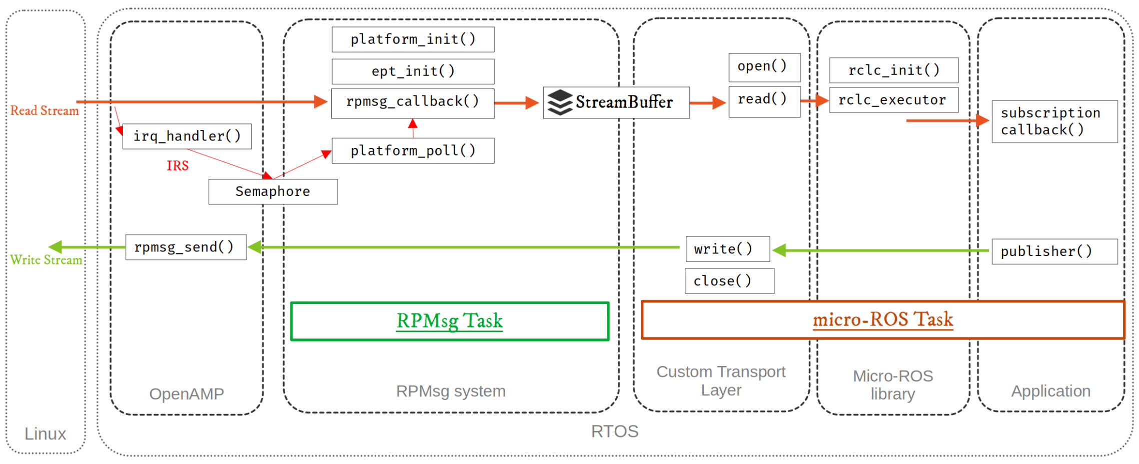


Fig. 5. Software architecture for the micro-ROS Client firmware and FreeRTOS Tasks.

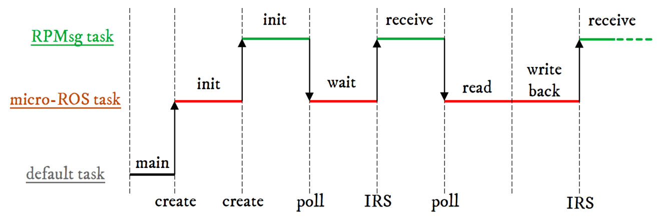


Fig. 6. Mains function tasks running the micro-ROS Client.

TABLE I
Deployment environment

SoC	Zynq UltraScale+ MPSoC ARM Cortex-A53, 4 cores ARM Cortex-R5F, 2 cores RAM DDR4	 1.5GHz 600MHz 4GB
Cortex-A53	Ubuntu 22.04 Linux v.5.15.0-1023 ROS2 v. Iron Irwini	
Cortex-R5F	FreeRTOS v.10 Build in Vitis IDE 2023.1 micro-ROS v. Iron Irwini	

IV. Evaluation

A. *Environment*

The proposed system was deployed and tested on a Xilinx KRIA KV260 single-board computer. See Table I for details on the various hardware and software used. The general HMP architecture of this ZynqMP processor was already visible in Fig. 2, but here we have some details on the software running on it as well as the exact core names.

B. *Challenges & difficulties*

The main challenges of this projects implementation were the deployment of micro-ROS on the MCU part of our HMP boards SoC and the adaptation of micro-ROS transport Agent to understand and use OpenAMP.

The first part required to adapt and port micro-ROS to an non-supported device. The separated cross-compilation of the library for the specific ARM core was needed before being able to merge it into a Xilinx firmware development pipeline.

The second part involved dealing with custom transport for micro-ROS Agent. Eventually, a fork and modification of an existing transport (serial) was done, requiring to fit the RPMsg initialization and functions into the Agent class.

C. *Performances*

Running a communication system over shared memory instead of serial wires provides an obvious potential benefit: data transfer speed. With the described system deployed, it is possible to make early data transfer speed measurements.

A comparative test was run between our new `RPMsg` DDS transport layer and a more conventional one, utilizing the Nucleo F446ZE board as a micro-ROS target and communicating with it using serial (UART).

Fig. 7 and 8 show a set of measurements of packages transmitted back and forth to the Client from the point of view of a ROS2 node behind the Agent, and the probability density of the transmission time for a single byte in each case.

Here are the characteristics of the transfer:

- The raw package transfer time and probability density functions for each case were computed, as well as the values for the average and the standard deviations, as seen in Table II.
- Two package sizes were chosen for this first test: 16 and 32 bytes. The size was supposed to be under 50 bytes since this is the default limit for example micro-ROS firmware running on the Nucleo board.
- Each measurement was made by sending only ASCII characters, as to ensure that the Python node will transmit a fixed, known number of bytes, without conversions.
- Each measurement was made by sending 10,000 packages of data, leading to a total of `160KB` or `320KB` being transmitted in each case.
- The `ping` demonstration function of the firmware was disabled in order to accelerate the transmission and avoid variations in the data transfer speed both for the Nucleo board used for serial tests and the Cortex-R5F on the HMP.

TABLE II
Data transfers for each measured case.

	Average [ms/B]	Average [KB/s]	Standard deviation [ms/B]
16B RPMsg	0.614	1.6	0.0133
32B RPMsg	0.305	3.3	0.0067
16B UART	0.789	1.3	0.028
32B UART	0.480	2.1	0.0158

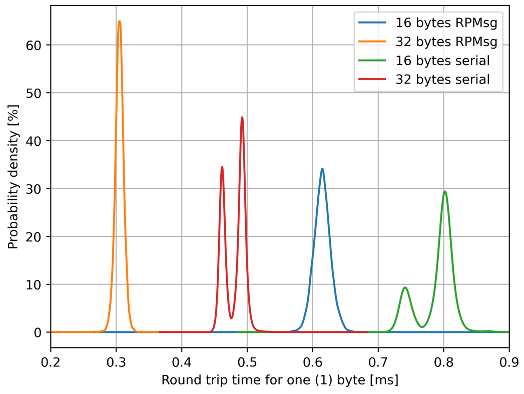


Fig. 7. Probability density for the data of the figure 8.

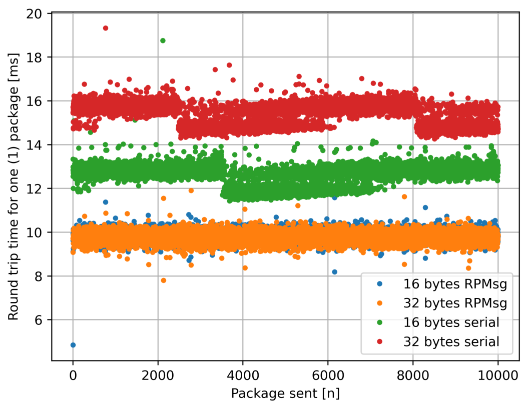


Fig. 8. Measurements of the time for 10'000 packages to go back and forth to the real-time core, measured at the Agent.

Two main takeaways are clear with Fig. 7 and Fig. 8:

- The general data speed was improved, but not as dramatically as expected. DDR memory being typically a thousand time faster that serial for raw data transfer speed, we would expect a more significant difference. This shows that the overhead of the OpenAMP and/or RPMsg system must be significant.
- However, the transfer time for RPMsg packages seems to not depend on its size, which appears as an net advantage of shared-memory over serial. This leaves room for further improvements in the future implementations.

D. *Contribution*

Overall, this project contributed to the ROS and robotic community by providing a new way to deploy the micro-ROS firmware.

We think that the modularity of ROS is particularly adapted at the nature of robotics development, with a variety of hardware being access. In that context, our solution proposed yet another way to access and communicate with devices, existing alongside other deployment solutions and options. Deploying micro-ROS on an HMP is, to our best knowledge, novel to this paper.

V. Future work & developments

A. *Performance enhancements*

The first move that will be made from this early version of the deployment is to try and enhance the transfer speed.

As the package throughput seems to be fairly size-agnostic, implementing larger package transfer capability should lead to significant performance gains. Different type of package type might also improve the raw bite transfer rate of the system by having less overhead.

Within the Client firmware implementation, copies of the data are currently performed. Getting rid of it in a new implementation should also improve responsiveness and data transfer speed.

Eventually, there is the option of working with the device's `virtio` system directly instead of `RPMsg`. This might require more adaptations since it is a less standard communication channel than `OpenAMP`.

Combining these points, we hope to have a significant gain of performance for this system of DDS messaging within a chip, taking full advantage of the shared-memory.

B. *Ease of use*

The presented architecture is currently in a very prototype-y state.

In order for this framework to be useful to other people, some further effort will be made to improve its ease of use. In particular we will be creating and publishing more documentation as well as making the sources more generally useful.

C. *Deployment on other HMPs*

As presented in the requirements, the goal of this project is to make the software abstraction layers as transparent as possible. In that regard, the deployment of the system on a different SoC is something to be tested.

We have good hope that the OpenAMP Agent will be re-deployable without any issue as a ROS2 node.

The micro-ROS Client, however, might need extra attention for new hardware, as it was already a challenge for this project.

D. *Implementation in robotics*

The use of ROS2 in combination with micro-ROS in the context of a walking robot has been archived before [8] but typically on separated hardware.

As the sensors, motors and actuators access must be real-time compliant, the connection to a micro-controller is required. With the usage of an HMP, it becomes possible to have a more direct access to these elements from the Linux devices.

With this advantage of using an HMP to deploy a robot firmware, a goal of our project is to be used for a hexapod robot [9] in order to improve the gait and walking control [10] as well as the vision and sensing capabilities.

Furthermore, the modular nature of ROS should allow the system to scale up seamlessly. Integrating other devices, nodes or even other HMPs is also something we want to work on for our robotic deployments.

E. *Tracing & in-depth monitoring*

A key application of the described system would be to enable tracing and debugging for ROS2 as discussed in a previous research [11]. The authors propose a low-overhead tracing mechanism for ROS2 and we would like to extend it to the micro-ROS devices. This effort should improve comfort for development and debugging on micro-ROS MCUs, including on an HMP.

VI. Summary & Conclusions

The successful integration of the ROS ecosystem into a device with heterogeneous cores, as discussed in this paper, offers numerous advantages for embedded system development, robotics, and beyond.

The novelty of the implementation presented in this publication resides in the fact that this is the first time a micro-ROS deployment was made on an commercial HMP. The purpose is to demonstrate the possibility of an intra-chip communication for ROS, showing what the challenges are but also the potential performance capabilities.

Fast communication between the real-time and the general purpose processors within the same chip gives the Linux-based control side a more direct access to the data gathered by sensor. The real-time system can be accessed while reducing the number of physical components and cables being used for a similar functionality.

Testing and evaluating such system is a continuous process, and since this technology is only as valuable as the practical benefits it can provide, further improvement and newer implementation versions will be worked on. The use of HMP for robots, especially in an academic context will provide a greater degree of integration for a cost way lower than other custom solutions such as FPGA or custom designed chip. The processors within the HMP are standard, but their setup onto a single silicon should be beneficial. As ROS2 and micro-ROS implementation are used as a base system for robotic application development and research, we hope these result to be meaningful for the ROS and embedded systems community as a whole.

Acknowledgements

This paper is based on results obtained from a project, JPNP23015, commissioned by the New Energy and Industrial Technology Development Organization (NEDO). The research is supported by the "Joint Research Promotion Program" of Aichi Science and Technology Foundation. Financial support for this project came from from Nanzan University as well. Finally, we also would like to thank Alexandra Elbakyan.

References

[1] M. Goraczko, J. Liu, D. Lymberopoulos, S. Matic, B. Priyantha, and F. Zhao, "Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems," *DAC '08: Proceedings of the 45th annual Design Automation Conference*, pp. 191–196, 2008.

[2] "Remote processor framework." `https://www.kernel.org/doc/html//v6.4/staging/remoteproc.html`. Accessed: 2023-11-03.

[3] L. Cuomo, C. Scordino, A. Ottaviano, N. Wistoff, R. Balas, L. Benini, E. Guidieri, and I. M. Savino, "Towards a risc-v open platform for next-generation automotive ecus," in *2023 12th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1–8, 2023.

[4] "Fast dds - the most complete open source dds middleware." `https://www.eprosima.com/index.php/products-all/eprosima-fast-dds`. Accessed: 2023-11-03.

[5] "Linux rpmsg framework overview." `https://wiki.st.com/stm32mpu/wiki/Linux_RPMsg_framework_overview`. Accessed: 2023-11-03.

[6] "Github - sunoc - micro-xrce-dds-agent." `https://github.com/sunoc/Micro-XRCE-DDS-Agent/tree/develop`. Accessed: 2023-11-03.

[7] "Gitlab - libmicroros kv260." `https://gitlab.com/sunoc/libmicroros_kv260`. Accessed: 2023-11-03.

[8] N. Weerakkodi, I. Zhura, I. Babataev, N. Elena, A. Fedoseev, and D. Tsetserukou, "Hyperdog: An open-source quadruped robot platform based on ros2 and micro-ros," pp. 436–441, 10 2022.

[9] K. Tanada, S. Inagaki, Y. Murata, R. Kato, and T. Suzuki, "Semi-autonomous walking control of a hexapod robot based on contact point planning and follow-the-contact-point gait control," in *Robotics in Natural Settings* (J. M. Cascalho, M. O. Tokhi, M. F. Silva, A. Mendes, K. Goher, and M. Funk, eds.), (Cham), pp. 289–300, Springer International Publishing, 2023.

[10] H. Hosogaya, S. Inagaki, and T. Suzuki, "Fcp gait control for hexapod robot capable of decreasing/increasing number of walking legs," *Transactions of the Society of Instrument and Control Engineers*, vol. 58, pp. 304–313, 06 2022.

[11] C. Bedard, I. Lutkebohle, and M. Dagenais, "ros2 tracing: Multipurpose low-overhead framework for real-time tracing of ros 2," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6511–6518, 2022.