CompROS: A composable ROS2 based architecture for real-time embedded robotic development

Saeid Dehnavi, Martijn Koedam, Andrew Nelson, Dip Goswami, Kees Goossens
Department of Electrical Engineering, Eindhoven University of Technology, The Netherlands
{S.Dehnavi, M.L.P.J.Koedam, A.T.Nelson, D.Goswami, K.G.W.Goossens}@tue.nl

Abstract—**Robot Operating System (ROS) is a de-facto standard robot middleware in many academic and industrial use cases. However, utilizing ROS/ROS2 in safety-critical embedded applications with real-time requirement is challenging because of C1) Non-real-time underlying hardware, C2) No control on the host OS scheduler, C3) Unpredictable dynamic memory allocation, C4) High resource requirement, and C5) Unpredictable execution model for ROS nodes. In this paper, we address these limiting factors by proposing a hardware-software architecture -CompROS- for ROS2 based robotic development in a Multi-Processor System on Chip (MPSoC) platform that. The proposed hardware architecture consists of a Hard Real-Time (HRT) RISC-V based subsystem implemented in the Programmable Logic (PL) part of the MPSoC platform, a Soft Real-Time (SRT) ARM-based subsystem in the Processing System (PS) part of the MPSoC platform, and a Non-Real-Time (NRT) PC. While the proposed hardware architecture along with a partitioning layer overcomes the first two limiting factors, the rest are managed by the proposed multi-layer software architecture. We make a bare-metal implementation of XRCE-DDS standard for PL-PS communication, while peer-to-peer PL-PL communication is done through a proposed real-time publish-subscribe approach. The reliable communication for PS-PLL communication is done through utilizing C-HEAP protocol. Further, we integrate ROS2 software layers on top of the proposed hardware and software layers. Finally, with respect to C5, we present a real-time execution model of ROS2 nodes by a mapping of ROS2 entities to CompROS entities, which is validated through experimental results. We run ROS2 middleware with an executable size of less than 200 KB on an MPSoC platform.**

Keywords-**Robotics; ROS2; Real-time; Embedded System;**

I. Introduction

Predictability: the temporal behaviour of every instruction is bounded by the Worst Case Response Time (WCRT), and **Composability:** there is no interference between different applications, are considered as two important factors for the analyzablity of real-time robotic applications in safety-critical domains such as medical robotics and space exploration [1]. Moreover, most of the robots in the mentioned domains are Cyber-Physical Systems (CPS) [2] equipped with a resource constrained embedded platform that suffers from limited memory capacity, limited Operating System (OS), etc. A common misconception is to consider a system real-time if it is fast enough. Following e.g. [3], a system is called real-time if it is predictable, i.e every operation has a WCRT.

Robot Operating System (ROS) [4] is a de-facto standard middleware in robotic development. In the ROS2 community, it is claimed that ROS2 embraces real-time features without formal analysis of the system, and no performance guarantee is provided in the current ROS2 versions. The challenges to utilize ROS2 in the mentioned real-time safety-critical applications are: **C1)** Targeted hardware platforms are mostly non-embedded architectures with unpredictable strategies like caching. **C2)** Since ROS2 runs on top of a host OS like Linux, it does not have control of the underlying scheduler. **C3)** Dynamic memory allocation in the ROS2 source code makes the execution time unpredictable. **C4)** ROS/ROS2 executable normally needs more than 100 MB RAM which is a challenging demand for small embedded systems that often have less than 1 MB of RAM. **C5)** There is no formal model to compute the WCRT of a ROS2 node.

Contribution: Although some of the above mentioned challenges have been investigated in the literature, to the best of our knowledge, there is no integrated work to consider all of the challenges in the context of ROS2. In this paper, we propose an integrated hardware-software architecture for robotic development in MPSoC platforms. More specifically, our contributions are:

1) A hardware architecture that includes a HRT subsystem based on CompSOC [5] concepts for real-time control tasks, a SRT ARM-based subsystem for supervisory control tasks, and a NRT Personal Computer (PC) for monitoring tasks. The HRT subsystem includes a partitioning kernel to partition the resources into isolated Virtual Execution Platforms (VEP) with the clock cycle-level precision. This contribution provides a performance analysis at the instruction level (addressing C1 and C2).
2) A software architecture that includes a Local Real-Time Publish-Subscribe (LRTPS) communication approach for HRT PL-PL communication, a bare-metal implementation of the XRCE-DDS standard for SRT PL-PS communication, and a lightweight predictable implementation of ROS2 layers on the proposed hardware architecture. Each of these these provide a performance analysis at the task level (addressing C3 and C4).
3) An execution model of ROS2 nodes to map the ROS2 entities into CompROS VEPs. This contribution provides a performance analysis at the application level (addressing C5).

II. Related Work

The works that fall into the scope of this paper are categorized into the following three different classes:

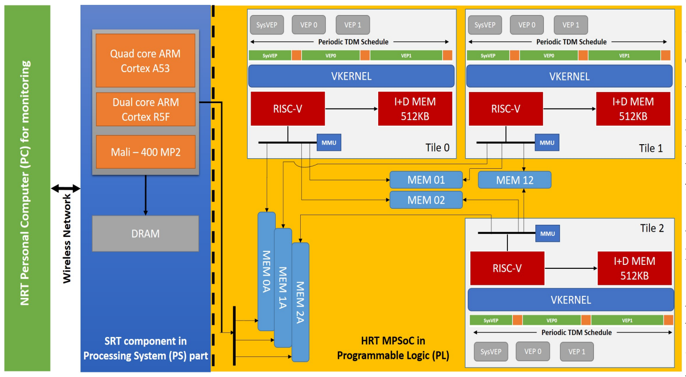


Fig. 1: CompROS Hardware Architecture

Hard real-time embedded hardware platforms: *Predictability* and *composability* in the hardware level have been heavily studied in the literature. In terms of predictability, Time-Triggered Architecture (TTA) [3] focused on the design and implementation of dependable real-time embedded systems. In line with the PRET machines [6], FlexPRET [7] is a configurable RISC-V [8] based architecture that executes both HRT tasks and NRT performance-sensitive tasks on the same processor. However, lack of composability in the right place, i.e. clock cycle level of the processor, means that these approaches are not fully predictable and composable. In a composable platform, the behaviour of an application is not affected by other applications. This is mainly achieved by temporal and spatial partitioning techniques. Time Division Multiplexing (TDM) [9] is a static and non-work-conserving partitioning technique to provide composability on a shared resource such as processor or memory. It has been used by some well-known platforms such as CompSOC [5] and MERASA [10]. In conclusion, to the best of our knowledge, CompSOC is the only platform that considers both cycle-accurate predictability and composability. We use CompSOC concepts as the underlying HRT subsystem.

ROS/ROS2 in real-time embedded systems: Considering the executable size and the sources of unpredictability such as dynamic memory allocation in ROS/ROS2, there has been some research on making a light-weight and real-time implementation for the embedded systems. Maruyama et al. [11] investigated real-time potentials and constraints of ROS2 and its communication layer (DDS). The research topics on embedded ROS are discussed in [12]. Medeiros et al. [13] proposed FreeRTPS as a light-weight implementation of Real-Time Publish Subscribe (RTPS) [14] protocol to run ROS2 applications on FreeRTOS [15]. However, ROS2 Quality of Services (QoS) are ignored in their implementation, and their approach is limited to FreeRTOS as the underlying OS. In [16], ROS2 was ported to the Nuttex RTOS. However, the project was limited to the single processor Nuttex based systems. Saito et al. [17] introduced ROSCH as an OS level real-time DAG scheduler. By using RESCH [18] as a loadable Linux scheduler module and a proposed synchronization technique, they showed the applicability of ROS for real-time control systems. However, their approach is based on ROS1, limited to Linux scheduling modules, and does not provide a guarantee on HRT control tasks. In conclusion, the proposed approaches for ROS/ROS2 in real-time embedded systems do not offer timing guarantee and/or ignore QoS.

Integrated architectures for robotic development in real-time embedded systems: Chishiro et al [19] discussed the general basis to propose a robot development framework in many-core systems. In the same direction, Azumi et a. [20] proposed ROS-Lite as a ROS based framework for NoC-based embedded many-core platforms. By integrating a proposed software layer, that runs on top of eMCOS real-time operating system, with the Kalray MPPA-256 many-core embedded platform they run a ROS based autonomous driving application with low resource and energy consumption. However, compared to the work presented in this paper, ROS-Lite is based on ROS (not ROS2) that ignores DDS based communication as a reliable and accepted communication standard in distributed real-time systems. Moreover, off-chip communication through an uniform communication protocol is not considered in their proposed approach. This makes the framework less useful for multi-robot ROS/ ROS2 based applications. Furthermore, their definition of real-time (fast enough to meet the deadline), is less strict than our definition (requiring formal analysis) in this paper.

Wei et al. [21] proposed RT-ROS, a hybrid architecture to run real-time and NRT ROS nodes on the same multi-processor system that hosts two operating systems (RTOS and NRT Linux). However, RT-ROS is based on ROS-1, and it is not a light-weight embedded platform. H-ROS [22] is a hardware-software architecture to design standardized ROS2 based modular robotic hardware components such as sensors and actuators. However, predictability and timing aspects are ignored in this approach. The MicroROS [23] project, replaced the default communication layer (Fast-DDS [24]) by ePromisa XRCE-DDS [25], for a light-weight implementation of ROS2 for small embedded devices with limited resources. Moreover, since all the dynamic memory allocations have been converted to a static allocation, a source of unpredictability has been removed. However, the main focus is on the software architecture and the real-time aspects of the hardware are ignored. The project is validated on a single-processor STM micro-controller with Nuttex RTOS. In conclusion, the works presented in this research direction do not offer real-time guarantee at the hardware level, have large memory footprint, and are focused on single processor.

III. CompROS Hardware architecture

As it can be seen in Fig. 1, the hardware architecture of CompROS consists of three subsystems including a HRT subsystem for real-time control, a SRT ARM subsystem for supervisory control, and a NRT PC for system monitoring. As the HRT subsystem, we implement a 3-tile CompSOC platform on a Xilinx ZCU102 board that hosts a Zynq UltraScale+ as the FPGA (PL) region. However, given the modularity of the platform, it is easily scalable to more tiles.

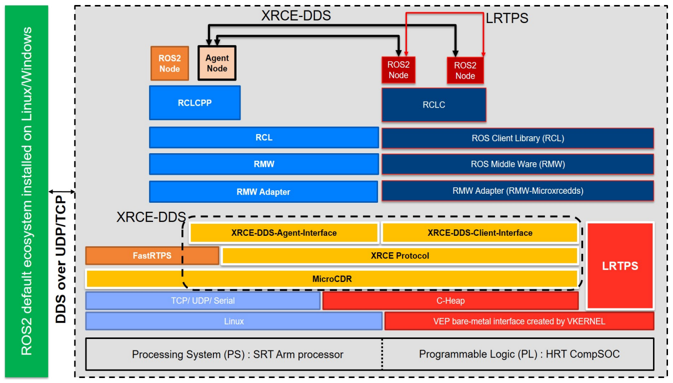


Fig. 2: CompROS Software Architecture

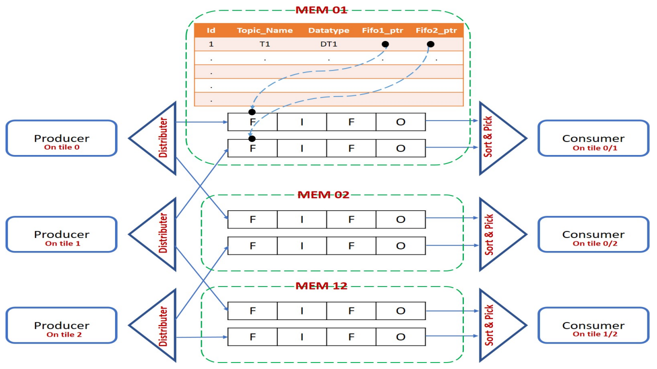


Fig. 3: Local Real-Time Publish Subscribe (LRTPS) approach

CompSOC Tiles: A 32-bit RISC-V processor that runs at 100 MHz is embedded inside each tile. Each tile includes 512 KB of Instruction+Data (I+D) memory that is implemented through the block memories of the PL. The VKERNEL partitions the tile resources into fully isolated VEPs by multiplexing different VEPs on the same RISC-V processor using a TDM preemptive scheduler. The I+D memory is also partitioned spatially among the VEPs that run on the same tile. Memory Management Units (MMUs) translate virtual addresses physical addresses and restrict access to allowed regions on all memories. In this version, we partitions the tile resources into 2 isolated VEPs (each with 256 KB of RAM). It should be mentioned that the context switch time between TDM slots (assigned to VEPs) is fixed and the first slot of each period is occupied by the system application (SysVEP). The platform is globally synchronous with a global timer, while each VEP has also its own local VEP timer that counts only when the processor is assigned to the VEP.

PL-PL and PL-PS communication: As it can be seen in Fig. 1, there is a real-time dual-ported shared memory with the capacity of 64 KB between any two tiles of the platform that is used for inter/intra tile communication of the VEPs. Moreover, for RISC-V to/from ARM (PL-PS) communication, and to dynamically load programs, there is a 32 KB dual-ported shared memory between each tile and the SRT ARM processor in the PS side. These memories (specified with MEMxA) are implemented using the block memories of the PL part and each memory is segmented between the VEPs on the tile (16 KB per VEP, 8 KB for user in/out, 8 KB for stdin/stdout). Although the access time from a tile to these shared memories is predictable, the access time from the SRT subsystem is not predictable.

SRT and NRT subsystems: The second subsystem of the CompROS architecture is a SRT subsystem that consists of a quad-core ARM Cortex-A53, dual-core Cortex-R5F real-time processors, and a Mali-400 MP2 graphics processing unit. This subsystem is used for supervisory control (like robot navigation), programming HRT subsystem, and connecting the robot to the world outside. Moreover, as the last subsystem, we have a NRT PC node that is used to monitor multiple robots each of which hosts its HRT+SRT subsystems. The communication from the monitoring system to the supervisory control system is done through wireless network infrastructures.

IV. CompROS Software architecture

The VKERNEL creates predictable and composable VEPs on the RISC-V cores through spatial and cycle-accurate temporal partitioning. Each partitioned HRT application runs bare metal (See Fig. 2). In the SRT (PS) subsystem, a Linux distribution (Ubuntu) is installed as the host OS. The communication inside the SRT, and with the world outside (NRT node) is done through TCP/UDP (the supported protocols by Fast-DDS), while the low-level PL-PS interaction is done through the C-HEAP FIFO protocol [26]. The performance of C-HEAP is modelled using dataflow in [27]. The default ROS2 layers (RCLCPP, RCL, RMW, and RMW Adapter) are used to develop ROS2 applications on the SRT subsystem. On the other hand, the ROS2 layers utilized in the HRT side must be free of unpredictability sources such as dynamic memory allocation. Therefore, we have made a bare-metal version of ROS2 layers on CompROS platform as a fork of microROS [23] by eliminating the OS dependencies and integrating it with our communication layers. Development of ROS2 applications on the HRT subsystem of the platform is done in C using the RCLC APIs.

PL-PS communication with XRCE-DDS protocol: Data Distribution Service (DDS) [28] is the default publish-subscribe communication layer of ROS2 to provide reliable machine to machine communication. Despite some lightweight implementations for DDS such as Fast-DDS by eProsima, the memory footprint is still too large to fit in resource-constrained embedded systems. XRCE-DDS [29] is a Client-Agent based pub-sub standard to enable resource constrained small embedded systems to connect to the DDS global data space. The Agent acts as a bridge between the Clients and other nodes in the DDS network. At the lowest layer of the protocol, MicroCDR is used for serialization/deserialization of DDS messages. At the top layer, there is XRCE-DDS Client Interface for development of Client applications on the HRT subsystem, and XRCE-DDS Agent Interface is used to create Agent nodes on the SRT subsystem. In the middle layer, the Client is only dependent on the XRCE Protocol layer, while the Agent is dependent on both the XRCE Protocol and Fast-DDS using standard DDS. This is because the Agent should provide the connection between XRCE-DDS based ROS2 nodes (on the HRT subsystem) and the nodes using the standard ROS2 architecture over the DDS network. Mpre details of the implementation and probabilistic performance analysis can be found in [30].

LRTPS (PL-PL real-time node-to-node communication): When both publisher and subscriber both reside in the HRT subsystem, communication via SRT subsystem using XRCE is slow, and worse, SRT. To provide HRT node-to-node communication there are some challenges. P1) Providing ROS2 QoS policies [31] is challenging, P2) Existing concurrent FIFO solutions, e.g. C-HEAP [26], fail to be a wait-free concurrent approach. P3) Every single publisher/subscriber that runs on a specific tile can only access to two of the communication memories (*MEM01*, *MEM02*, and *MEM12*). Because of P3, data may have to be duplicated across multiple communication memories, and the data consistency should be considered. To address these challenges, we propose the LRTPS protocol by S1) a single wait-free pub-sub FIFO channel, presented in [32], that allows token overwriting by the producer. Our FIFO buffer has a WCRT even though it uses a lock, and the QoS policies are preserved. S2) For any topic *Ti* with datatype *DTi*, there is one FIFO for each publisher-subscriber pair in the communication memory that both publisher and subscriber can access.

Algorithm 1 safe and consistent write on LRTPS FIFOs

```
1 while ((wc+1) % n) == rc /*FIFO full*/ do
2     if cclaim==0 then
3         pclaim=1
4         if cclaim==0 then
5             rc=(rc+1) % n
6         pclaim=0
7 Write token
8 wc = (wc + 1) % n
```

Algorithm 2 safe and consistent read on LRTPS FIFOs

```
0 if wc==rc /*FIFO empty*/ then return Null;
1 cclaim=1
2 while (pclaim==1);
3 Read token
4 rc = (rc+1) % n
5 cclaim=0
```

In the example of Fig. 3, the producer node publishes its message on topic *Ti* to the assigned FIFO channels in both memories connected to the tile that hosts the producer. On the consumer side, multiple FIFO channels per topic (2 channels in each shared memory) are received. Therefore, the consumer picks and consumes the received tokens based on their timestamps to respect consistency when there is more than one publisher on a topic. Since the message communication in our approach is done at the HRT subsystem, the *Deadline* QoS is supported for ROS2 nodes running in the system. According to the *History* QoS policy in ROS2, the message processing queue has a maximum size equal to the *Depth* (n) value. If the queue is full, the oldest messages are dropped to make room for newer ones. We address this by Alg. 1 and Alg. 2 to write/read on the FIFO channels. In the write/read algorithms, we use producer/consumer claim locks (pclaim and cclaim) for safe concurrent access. The WCRT analysis is found in [32].

V. Execution model

The HRT subsystem of CompROS is a multi core subsystem in which the tile resources are spatially and temporally partitioned into v isolated VEPs (v=2 in this paper). Scheduling the created VEPs on each RISC-V processor is done through a preemptive static TDM schedule. As it is shown in Fig. 4, the TDM period for processor p is shown by π_p, the first slot of the TDM schedule ($s_{0,p}$) is assigned to the system application (*SysVEP*), and each VEP$_{i,p}$ is assigned to slot $s_{i,p}$. The slot length for the system application ($s_{0,p}$) and the context switch time (c) are 5K and 2K cycles respectively. In our task model, for control and dataflow applications where the computation and communication times are independent, we compute their WCRT separately. In this model, a node first reads its input messages, computes the output messages, and send them. The CompSOC hardware architecture allows us to compute the WCET of computation and communication [5] from which we then drive the WCRT. For a specific task that runs in VEP$_{i,p}$, given the WCET of the computation part (e^{cmp}), the computation WCRT (r^{cmp}) is calculated by Eq. 1. Note that Eqs. 2,3 are independent. However, the publisher WCRT depends on the WCRT of the code segments of publisher and subscriber, and similarly for the subscriber.

$$r^{cmp} \leqslant \left(\lceil \frac{e^{cmp}}{s_{i,p}} \rceil + 1 \right) \times \pi_p \tag{1}$$

$$r^{p2p}_{pub} \leqslant 2 \times \left(\lceil \frac{e_{p1-8}}{s_{i,w}} \rceil + 1 \right) \times \pi_w + \left(\lceil \frac{e_{c2-5}}{s_{j,z}} \rceil + 1 \right) \times \pi_z \tag{2}$$

$$r^{p2p}_{sub} \leqslant \left(\lceil \frac{e_{c0-1}}{s_{j,z}} \rceil + \lceil \frac{e_{c3-5}}{s_{j,z}} \rceil + 2 \right) \times \pi_z + \left(\lceil \frac{e_{p5-6}}{s_{i,w}} \rceil + 1 \right) \times \pi_w \tag{3}$$

A. PL-PS/PL-PL communication WCRT: Since the PL-PS communication involves the SRT subsystem, it is not possible to guarantee the communication time. However, this problem is addressed in the XRCE-DDS standard by defining a *Timeout* parameter as the maximum waiting time for the *Agent* response. In other words, the *Client* (publisher/subscriber at the HRT subsystem) sends its request to the *Agent* (at the SRT subsystem) and waits to receive the pub/sub response. According to the standard, if the defined *Timeout* happens, the *Client* continues to the next round. In this approach, we may loose some messages but the WCRT of pub/sub on a topic is bounded by the defined

Timeout. Moreover, as presented in [32], the WCRT of publish/subscribe functions on a specific topic in PL-PL real-time communication (r^{p2p}_{pub} and r^{p2p}_{sub}) are guaranteed through Eq. 2 and Eq. 3 respectively. In these equations, the publisher runs in VEP$_{i,w}$ with slot length $s_{i,w}$, and the subscriber runs in VEP$_{j,z}$ with slot length $s_{j,z}$. Moreover, e_{p1-8} is the WCET of the lines 1-8 of the producer (Alg. 1), e_{c2-5} represents the WCET of the lines 2-5 of the consumer (Alg. 2), etc.

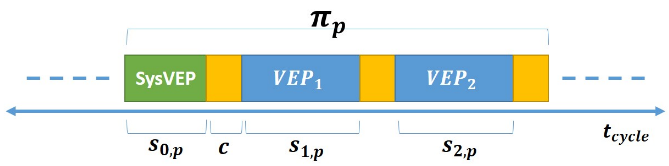


Fig. 4: TDM Schedule of isolated VEPs on the processor p

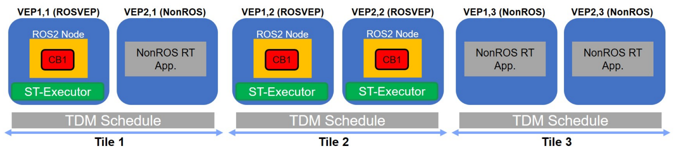


Fig. 5: Mapping ROS2 entities to CompROS entities

B. Application Response Time: It is important to mention that the pub-sub paradigm is a transparent communication model (whiteboard model) in which the publisher sends its messages to a topic regardless of who subscribes on the topic and vice-versa. Therefore, the WCRT of a publisher/subscriber task (r^{task}) depends only on the computation time of the task (e^{cmp}) and the pub/sub time to/from the interested topic. Therefore, we can find r^{task} for a publisher task by the equation $r^{task} \leqslant r^{cmp} + r^{p2p}_{pub} + Timeout$, and the WCRT of a subscriber task by replacing r^{p2p}_{pub} with r^{p2p}_{sub} in the same equation. It is worth mentioning that in the publication/subscription time on a topic, the message is published/subscribed to/from both PL-PL/PL-PS channels to be visible to all nodes on the DDS network.

C. ROS2 execution model on CompROS: In general, there are three entities in the default execution model of the ROS2 ecosystem. **Node:** a modular computation process to perform a specific task in the robot. **Callback:** The functionality of a ROS2 node is comprised of different callback functions. **Executor:** In every single time of the processor the executor decides about which callback from which node should be executed. In other words, scheduling of the executors on the processor is done by the OS scheduler, while internal scheduling of the executor over the assigned nodes and their callbacks is done by the scheduler inside the executor.

Casini et al. [33] showed that the default executor of ROS2 is not real-time and it suffers from priority inversion, unpredictable update, one message per callback, and no custom ordering. In [34], a static Single Threaded (*ST*) executor was proposed to address the mentioned problems. As it can be seen in Fig. 5, we assign each ST-Executor with one node and single callback function to a VEP on CompROS HRT side. Therefore, the timing isolation is guaranteed by the VEP concept, and there is no priority assignment, and no priority inversion in the used executor. Since our HRT subsystem is a multi core subsystem hosting isolated VEPs on each tile, we can run ROS2 applications (in ROSVEP) and non-ROS real-time applications (in NonROS VEPs) in parallel on the same system. In this model, the WCRT of a ROS2 node with a callback function on a topic can be calculated by the discussed equations in Sec. V.*B*.

VI. Experimental evaluation

As the experimental setup, we made our implementation on a Xilinx ZCU102 board that includes a Zynq UltraScale+ XCZU9EG with 600K system logic cells, 32.1 MB memory, and 2520 DSP slices in the FPGA region. On the PS side, the board includes a 64 bit quad-core Arm Cortex-A53, a dual-core Cortex-R5F real-time processor, and a Mali-400 MP2 graphics processing unit with 4 GB of DRAM. The PS side hosts the SRT subsystem of our architecture. Connection to the board is done through SSH from a NRT PC. In the configuration time of the platform, we assign 100K clock cycles to each VEP$_{i,p}$ on a tile as the TDM slot size $s_{i,p}$ (introduced in Sec. V) assigned to the VEP.

A. Real-time software development on the HRT MPSoC

To validate the hardware predictability, we need to make sure that the WCRT of a fully compute-intensive real-time task with a given WCET is guaranteed on the platform. We developed 6 dummy real-time tasks with different workload (WCET), and assign each task to a specific VEP (6 VEPs on the platform). As it was mentioned in Sec. III, we measure the ET and RT of each task respectively with the VEP timer and global timer. Moreover, we calculate the WCRT of each task by Eq. 1. As it can be seen in Table I, the measured RT for all real-time tasks are less than the calculated WCRT. It is worth mentioning that increasing the TDM slot size of a specific VEP leads to a decrease in the RT and WCRT of the task that is hosted by the VEP, but increases the RT and WCRT of other VEPs on the tile. Therefore, it can be used to prioritise the tasks on the same tile. Considering the composability of the platform, the reported values for a specific task are the same even if we remove the other tasks on a tile.

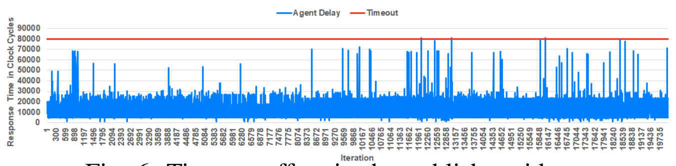


Fig. 6: Timeout effect in the publisher side

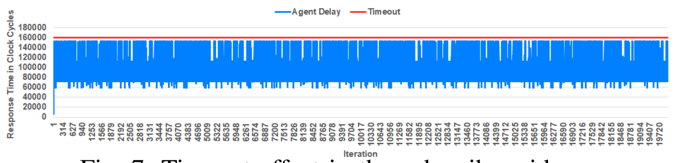


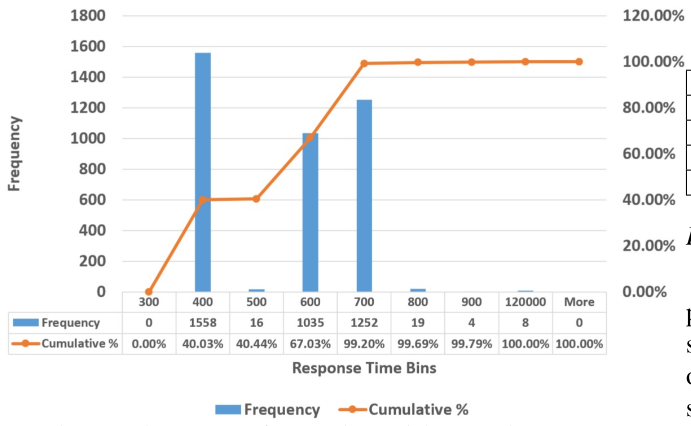
Fig. 7: Timeout effect in the subscriber side

TABLE I: HRT SOFTWARE DEVELOPMENT ON THE PLATFORM

Task (tile,vep)	**Task.1 (0, 1)**	**Task.2 (0, 2)**	**Task.3 (1, 1)**	**Task.4 (1, 2)**	**Task.5 (2, 1)**	**Task.6 (2, 2)**
ET	7019	15426	28031	57449	217126	420226
RT	14019	55426	77031	181449	665126	1321226
WCRT	62000	93000	124000	217000	713000	1364000

B. PL-PS communication through XRCE-DDS

To measure the performance of our implementation of XRCE-DDS for PL-PS communication, we conducted experiments on the pure XRCE-DDS communication (ignoring ROS2 layers). In these experiments, the publisher and subscriber run on the HRT subsystem as the XRCE-DDS Clients, and the XRCE-DDS Agent runs on the SRT subsystem of the platform. However, all the messages sent/received in the HRT subsystem, are also accessible over DDS network. The measured throughput and RT over different message size is presented in Table II. Moreover, to show the effect of *Timeout* parameter on bounding the WCRT of the publisher, we added random bad requests in the publisher side to make the Agent take long time. To check the *Timeout* effect on the subscriber side, we made the published messages failed to keep the subscriber waiting for the longest time. As it is shown in Fig. 6 and Fig. 7, the *Timeout* effect on the publisher and subscriber side are validated respectively.



	300	400	500	600	700	800	900	120000	More
Frequency	0	1558	16	1035	1252	19	4	8	0
Cumulative %	0.00%	40.03%	40.44%	67.03%	99.20%	99.69%	99.79%	100.00%	100.00%

Fig. 8: Histogram of Actual publisher RT in LRTPS

C. LRTPS real-time PL-PL communication

We conducted some experiments on the LRTPS layer (ignoring ROS2 layers) to validate the system predictability in terms of node-to-node PL-PL communication. The experiments were done in various configurations with multiple publishers/subscribers to make sure that the proposed approach is consistent with multiple numbers of nodes. To evaluate the effect of Alg. 1 and Alg. 2 in non-blocking pub-sub communication, our consumer node in these experiments is delayed by 1000 cycles after each subscription. This is to validate that the producer is not blocked even if the buffer is full (*History* QoS). We first measure the WCET of the publisher by running the publisher algorithm on the platform for 20K iterations. The ET of the publisher varies between 300 to 700 clock cycles. The variation in the publisher ET is related to the scenarios where the buffer is full and the *rc* variable should be shifted. The same experiment on the read function (Alg. 2) results the actual WCET of the consumer (subscriber) node. We also measure the actual RT of the publisher using the global timer of the HRT subsystem. As it is presented in Fig. 8, the actual RT of the producer node is bounded by the calculated WCRT (Sec. V.A). It should be mentioned that the variation in the actual RT is related to the context switch time. The variation is within the bound and it itself is quite predictable. Moreover, the throughput (TP) of message publication in both PL-PL and PL-PS communication is compared in Fig. 9 over different message sizes. As it was expected, the TP in LRTPS is at least 6 times more than PL-PS XRCE-DDS communication. It is also showed that by increasing the message size, the TP in LRTPS is not affected heavily since the read/write time to the shared memory is fast in the HRT subsystem. It should be noted that in these experiment, the computation power of each HRT tile is shared between two isolated VEPs. Therefore, the reported TP numbers can be almost doubled when the whole processor is assigned to a single VEP.

TABLE II: TP (MSG PER SEC) IN PL-PS COMMUNICATION

Message	**Publisher TP**	**Subscriber TB**	**App. (end-2-end TP)**
10B	1281	1056	1173
25B	1028	893	961
50B	860	794	763
100B	602	539	513

D. ROS2 application development:

As a proof to run ROS2 applications on CompROS platform, we consider a simple usecase with one ROS2 sensor node on tile 1, one ROS2 actuator node on tile 2, one ROS2 real-time control node on tile 3, and a ROS2 supervisory control node on the SRT part. In this usecase, the supervisory control node sends set points to the actuator node based on the data that it receives from the sensor node. Moreover, the real-time control node receives and processes the data from the sensor node through the real-time PL-PL communication. There is also a monitoring node on the NRT PC to monitor the whole nodes. In this experiment, we measured the actual ET and the actual RT of each ROS2 node that runs on the HRT subsystem over 500K iterations. As it is shown in Fig. 10, the measured values for the control node is significantly less than other nodes. This is due to node-to-node PL-PL communication since the other sensor and actuator nodes have interaction with the supervisory control node that runs on the SRT subsystem through the XRCE-DDS Agent node. Since the supervisory control node and the monitoring node run on the SRT and NRT subsystems, measuring their timing behaviour does not give useful information in terms of predictability. It should be mentioned that the final executable size for the whole

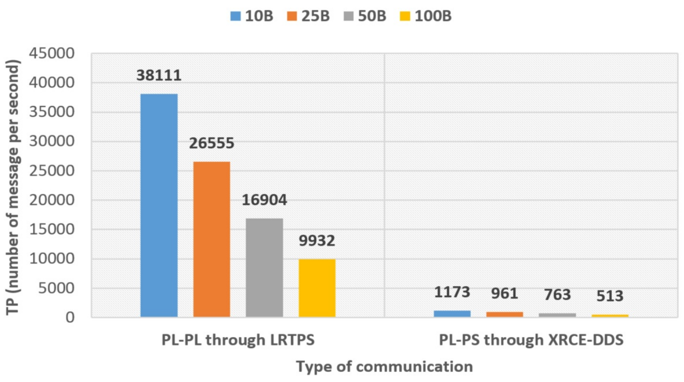


Fig. 9: Compare Throughput of LRTPS and XRCE-DDS

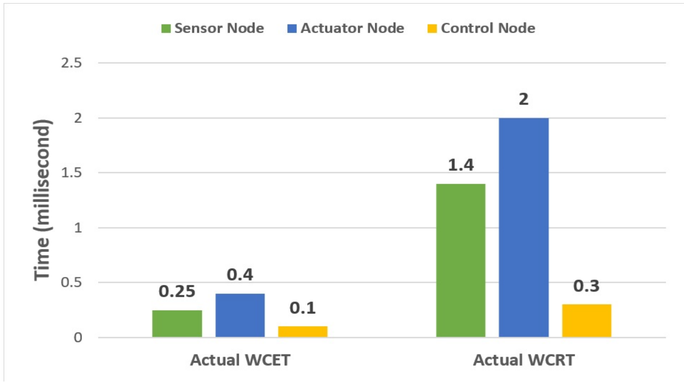


Fig. 10: Actual WCET and WCRT on ROS2 nodes

software layers on the proposed HRT subsystem is less than 200 KB which is significantly less than the default ROS2 ecosystem that requires more than 100 MB of memory.

VII. Conclusion and future work

In this paper we presented CompROS, a composable ROS2 based hardware-software architecture for predictable and composable robotic development in real-time embedded systems. CompROS includes a HRT multi core subsystem for real-time control, a SRT subsystem for supervisory control, and a NRT PC for system monitoring. The HRT subsystem has a formal model at the instruction level, and its integration with the proposed software architecture has a formal model at the task level. Based on the proposed execution model, we define the WCRT of a single ROS2 node that runs on the HRT subsystem of the platform. ECSEL JU grant agreement No 826610 (COMP4DRONES) supported this work.

References

[1] J. Guiochet *et al.*, “Safety-critical advanced robots: A survey,” *Robotics and Autonomous Systems*, 2017.
[2] E. A. Lee, “Cyber physical systems: Design challenges,” in *ISORC*, 2008.
[3] H. Kopetz, *”Real-time systems: design principles for distributed embedded applications”*. Springer, 2011.
[4] M. Quigley *et al.*, “ROS: an open-source Robot Operating System,” in *ICRA*, 2009.
[5] K. Goossens *et al.*, “NoC-Based Multiprocessor Architecture for Mixed-Time-Criticality Applications,” in *Handbook of Hardware-Software Codesign*, Springer, 2017.
[6] S. A. Edwards and E. A. Lee, “The case for the precision timed (PRET) machine,” in *DAC*, 2007.
[7] M. Zimmer *et al.*, “Flexpret: A processor platform for mixed-criticality systems,” in *RTAS*, 2014.
[8] A. Waterman *et al.*, “The RISC-V instruction set manual, volume i: Base user-level isa,” *UC Berkeley, Tech. Rep.*, 2011.
[9] S. Goossens *et al.*, “A reconfigurable real-time SDRAM controller for mixed time-criticality systems,” in *CODES+ ISSS*, 2013.
[10] T. Ungerer *et al.*, “Merasa: Multicore execution of hard real-time applications supporting analyzability,” *IEEE Micro*, 2010.
[11] Y. Maruyama *et al.*, “Exploring the performance of ROS2,” in *EMSOFT*, 2016.
[12] P. Bouchier, “Embedded ROS [ROS topics],” *IEEE Robotics & Automation Magazine*, 2013.
[13] L. da Silva Medeiros *et al.*, “Enabling Real-Time Processing for ROS2 Embedded Systems,” in *Robot Operating System (ROS)*, Springer, 2019.
[14] Object Management Group, “Real-Time Publish Subscribe (RTPS),” in *https://www.omg.org/spec/DDSI-RTPS*, 2018.
[15] [Online]. Available: http://www.freertos.org/.
[16] [Online]. Available: https : / / github . com / ros2 / ros2_embedded_nuttx.
[17] Y. Saito *et al.*, “ROSCH: Real-Time Scheduling Framework for ROS.,” in *RTCSA*, 2018.
[18] S. Kato *et al.*, “A loadable real-time scheduler suite for multicore platforms,” *Tech. Report CMU*, 2009.
[19] H. Chishiro *et al.*, “Towards heterogeneous computing platforms for autonomous driving,” in *ICESS*, 2019.
[20] T. Azumi *et al.*, “ROS-lite: ROS Framework for NoC-Based Embedded Many-Core Platform,” in *IROS*, 2020.
[21] H. Wei *et al.*, “RT-ROS: A real-time ROS architecture on multi-core processors,” *Future Generation Computer Systems*, 2016.
[22] V. Mayoral *et al.*, “The shift in the robotics paradigm—The Hardware ROS (H-ROS);” in *AHS*, 2017.
[23] [Online]. Available: https://github.com/microROS/.
[24] [Online]. Available: https://www.eprosima.com/index.php/products-all/eprosima-fast-dds/.
[25] [Online]. Available: https://github.com/eProsima/Micro-XRCE-DDS/.
[26] A. Nieuwland *et al.*, “C-HEAP: A heterogeneous multi-processor architecture template and scalable and flexible protocol for the design of embedded signal processing systems,” *DAES*, 2002.
[27] A. Nelson *et al.*, “Dataflow formalisation of real-time streaming applications on a composable and predictable multi-processor SOC,” *Journal of Systems Architecture*, 2015.
[28] G. Pardo-Castellote, “OMG Data Distribution Service: Architectural overview,” in *ICDC*, 2003.
[29] Object Management Group (OMG), *”DDS for Extremely Resource Constrained Environments”*. [Online]. Available: https://www.omg.org/spec/DDS-XRCE/1.0/Beta2.
[30] S. Dehnavi *et al.*, “Modeling, implementation, and analysis of XRCE-DDS applications in distributed multi-processor real-time embedded systems,” in *DATE*, 2021.
[31] [Online]. Available: https://design.ros2.org/articles/qos.html.
[32] S. Dehnavi *et al.*, “A predictable single producer single consumer wait-free fifo,” *Electronic systems group tech. note, 2021-12*, 2021.
[33] D. Casini *et al.*, “Response-Time Analysis of ROS 2 Processing Chains Under Reservation-Based Scheduling,” in *ECRTS*, 2019.
[34] [Online]. Available: http : / / www . ofera . eu / index . php / publications.