Micro-ROS: ROS2 on microcontrollers

Audience Check

- Disciplines:
  - Computer science
  - Electrical engineering
  - Mechanical engineering
  - Other?

- Who has used an Arduino or similar maker board?
- Who has written hardware drivers for ROS?
Micro-ROS: ROS2 on microcontrollers

Microcontrollers are everywhere

- Typical applications
  - Motor control
  - Sensor interfaces (AD, post-processing)
    - Incl. sensor fusion
  - Driving displays, LEDs, etc.
  - Low-latency real-time control

- Characteristics
  - Low power usage (up to battery operation for years)
  - Very predictable execution times
  - Hardware integration
  - Many integrated I/Os (I²C, SPI, CAN, etc.)
  - Sophisticated safety-rated versions available
BUT: Development totally disconnected from ROS-based development
**Micro-ROS: ROS2 on microcontrollers**

### Goals

- Run nodes on microcontroller *seamlessly*
  - Publish/subscribe/services just work
  - Parameters/lifecycle/… just work
- Take advantage of hardware features
  - Power saving
  - Hard real-time scheduling
  - Easy hardware access
- Developer Experience
  - Build using ROS tools
  - Same codebase and APIs wherever possible
- Challenges
  - Resource use (RAM, CPU, Disk)
  - Different build-systems, OS, community expectation

### Typical topology

```
<table>
<thead>
<tr>
<th></th>
<th>ROS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux/Windows/Mac OS</td>
<td></td>
</tr>
<tr>
<td>MCP</td>
<td></td>
</tr>
<tr>
<td>Firmware</td>
<td></td>
</tr>
<tr>
<td>MCU</td>
<td></td>
</tr>
<tr>
<td>Device (Sensor/Actuator)</td>
<td></td>
</tr>
</tbody>
</table>
```

- Optional

---

© Robert Bosch GmbH 2019. All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.
# Micro-ROS: ROS2 on microcontrollers

## Device Classes

<table>
<thead>
<tr>
<th></th>
<th>ROS2</th>
<th>Micro-ROS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware</strong></td>
<td>X86, ARM Cortex-A, ...</td>
<td>ARM Cortex-M, ...</td>
</tr>
<tr>
<td><strong>Resources</strong></td>
<td>&gt;512 MB RAM, &gt;8 GB Disk</td>
<td>&gt;100 KB RAM, &gt;1 MB Flash</td>
</tr>
<tr>
<td><strong>Communications</strong></td>
<td>Ethernet, 802.11 WiFi</td>
<td>Serial, WPAN – 250 KBit - 1 MBit/s</td>
</tr>
<tr>
<td><strong>Operating System</strong></td>
<td>Linux, Windows, MacOS</td>
<td>RTOS (NuttX by default)</td>
</tr>
<tr>
<td><strong>Middleware</strong></td>
<td>DDS variant (by default)</td>
<td>XRCE-DDS (by default)</td>
</tr>
<tr>
<td><strong>Middleware Abstraction</strong></td>
<td>RMW</td>
<td>RMW</td>
</tr>
<tr>
<td><strong>Client Support Library</strong></td>
<td>RCL</td>
<td>RCL</td>
</tr>
<tr>
<td><strong>Execution Layer</strong></td>
<td>RCLCPP / RCLPY / ...</td>
<td>RCL + RCLCPP</td>
</tr>
<tr>
<td><strong>Executors</strong></td>
<td>Generic</td>
<td>Micro-ROS custom</td>
</tr>
</tbody>
</table>
## Micro-ROS: ROS2 on microcontrollers

### Ingredients

<table>
<thead>
<tr>
<th>Ease of use</th>
<th>MCU-targeted capabilities</th>
<th>Performance &amp; Predictability</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Build system integration</td>
<td>- Middleware XRCE-DDS</td>
<td>- Executor performance</td>
</tr>
<tr>
<td>- Default configurations</td>
<td>- Custom executors</td>
<td>- Deterministic execution</td>
</tr>
<tr>
<td>- Default hardware</td>
<td>- E.g., static ordering</td>
<td>- System Modes</td>
</tr>
<tr>
<td>- Looking for collaborators!</td>
<td>- MCU tracing and debugging</td>
<td>- Benchmarking tools</td>
</tr>
<tr>
<td>- Tutorials</td>
<td>- Portability</td>
<td></td>
</tr>
<tr>
<td>- Community demos</td>
<td>- Transports extensible</td>
<td></td>
</tr>
<tr>
<td>- Slack channel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Ready-to-use docker containers</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### MCU-targeted capabilities
- Middleware XRCE-DDS
- Custom executors
  - E.g., static ordering
- MCU tracing and debugging
- Portability
  - Transports extensible
## Differences between ROS 2 and Micro-ROS

<table>
<thead>
<tr>
<th>ROS 2</th>
<th>Micro-ROS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application component</strong></td>
<td><strong>Application component</strong></td>
</tr>
<tr>
<td>C++ API (rcpp)</td>
<td>C++ API (rcpp)</td>
</tr>
<tr>
<td>Python API (rdpy)</td>
<td>System modes</td>
</tr>
<tr>
<td>Java API (rdjava)</td>
<td>Predictable execution</td>
</tr>
<tr>
<td><strong>C API</strong></td>
<td><strong>C API</strong></td>
</tr>
<tr>
<td>ROS Client Support Lib (rcl)</td>
<td>ROS Client Support Lib (rcl)</td>
</tr>
<tr>
<td>** ROS Middleware Interface (rmw)**</td>
<td>** ROS Middleware Interface (rmw)**</td>
</tr>
<tr>
<td>FastRTPS</td>
<td>Micro XRCE-DDS Middleware</td>
</tr>
<tr>
<td>RTI Connext</td>
<td></td>
</tr>
<tr>
<td>freeRTPS</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td><strong>O.S. (Linux, Windows, Mac, RTOS)</strong></td>
<td><strong>POSIX</strong></td>
</tr>
<tr>
<td></td>
<td>+ Aditional abstractions</td>
</tr>
<tr>
<td></td>
<td>NuttX RTOS</td>
</tr>
<tr>
<td><strong>microProcessor</strong></td>
<td><strong>microController</strong></td>
</tr>
</tbody>
</table>
micro-ROS: ROS 2 on microcontrollers

Target Devices

- Reference HW platform
  - Cortex-M4 devices with ~100KB RAM
    - Olimex STM32-E407
  - Cortex-M0 investigated but no longer pursued

- 3rd Party platforms
  - Renesas is on track to support GR-ROSE boards
  - Sony has expressed interest in supporting their SPRESENSE board

- RTOS
  - Default RTOS is NuttX
  - Intel has expressed interest in working on Zephyr support
micro-ROS: ROS 2 on microcontrollers

Build System: Background

- RTOS’s are complete packages including
  - Scheduling (of course)
  - Networking
  - Standard libraries (libc, libm, libstdc++) etc
  - Tools, and many more things

- RTOS’s are highly configurable
  - Most things are turned off by default to save resources
  - Every change can affect system headers

→ RTOS’s have relatively sophisticated, diverse and complex build systems

- Microcontrollers often build operating system and application into a single firmware image
  - This includes all dependencies, e.g. ROS 2
  - Everything is cross-compiled
micro-ROS: ROS 2 on microcontrollers
Micro-ROS Build Support


▶ Features
  ▶ Creates the firmware workspace for you
    ▶ RTOS
    ▶ Apps
    ▶ Necessary ROS 2 packages
    ▶ Cross-compilation setup that avoids interference from already source ROS 2 host workspace
  ▶ Creates agent workspace for you
  ▶ Example: „ros2 run micro_ros_setup build_firmware.sh“

▶ Upcoming work: Integrate as CMake macros for ease of use
micro-ROS: ROS 2 on microcontrollers

Executor Performance

- The current SingleThreadedExecutor adds measurable overhead
  - For some use-cases, just polling the middleware already consumes 20% of CPU
- Nobleo has addresses this in rclcpp PR 873
- We’ve also identified more overhead in the rmw implementations, this is current work.
DDS meets MCUs: DDS-XRCE

- OMG’s DDS-XRCE (DDS for eXtremely Resource Constrained Environments) brings DDS on MCUs
- Based on Client-Server architecture
  - Power-Saving
  - Stateless
- Agent acts on behalf of Clients (Low resource devices) on the DDS global data space.
micro-ROS: ROS 2 on microcontrollers

Micro XRCE-DDS

- eProsima C99 (Client) / C++11 (Agent) implementation of XRCE protocol
- Multiple and extensible transport support: UDP/IP, TCP/IP, Serial ... or create your own!
- Low memory usage (Client library):
  - Stack: ~2 KB
  - Heap: 0 KB (**only static memory**)
  - .Text (code in Flash): core: 64 KB +/- TCP profile: 2 KB +/- UDP profile: 1 KB +/- Serial profile: 5.5 KB ...
- Agent library API: micro-ROS-Agent
- ROS 2
  - Several success stories: Robotis, Renesas.
  - Crystal and Dashing enabled.

Open-source: [https://github.com/eProsima/Micro-XRCE-DDS/](https://github.com/eProsima/Micro-XRCE-DDS/)

micro-ROS: ROS 2 on microcontrollers

Micro XRCE-DDS Client portability

A) Clock dependency.
- Relative clock measurement. Crazyflie timer registers e.g.

B) Platform transports.
- Simple pairs of functions required:
  - Init/Close.
  - Write/Read.
- Common platforms implementation provided.
Drone demo

micro-ROS: ROS 2 on microcontrollers
<table>
<thead>
<tr>
<th>Status</th>
<th>Crystal</th>
<th>Dashing</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No executor</td>
<td>LET executor</td>
<td>Complete RMW Implementation.</td>
</tr>
<tr>
<td></td>
<td>RMW Simple communication mechanisms.</td>
<td>RMW configuration.</td>
<td>Incorporate new platforms to build system. e.g. FreeRTOS</td>
</tr>
<tr>
<td></td>
<td>Basic and nested type support, no arrays.</td>
<td>Full type support.</td>
<td>CPP API support (RCLPP) in some platforms.</td>
</tr>
<tr>
<td></td>
<td>NuttX firmware incorporated in build system.</td>
<td></td>
<td>Demos and more tutorials.</td>
</tr>
<tr>
<td></td>
<td>Plain C API support (RCL).</td>
<td></td>
<td>ROS2 Packages.</td>
</tr>
<tr>
<td></td>
<td>Demos.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ready to use dockers.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
micro-ROS: ROS 2 on microcontrollers

Further information

- All open-sourced code at GitHub
  - [https://github.com/micro-ROS](https://github.com/micro-ROS)
- Web-site: [micro-ros.github.io](https://micro-ros.github.io)
- Slack [micro-ros.slack.com](https://micro-ros.slack.com)
- ROS 2 Embedded Working Group
- ROS Discourse in Embedded category
- ROS 2 Embedded Design Page
  - [https://github.com/ros2/design/pull/197](https://github.com/ros2/design/pull/197)