micro-ROS on ESP32 [tutorial]

4 Comments / By Geibinger / 2023-06-14

What is ROS and why would you want to run a micro version of it on an ESP32?

ROS is the Robot Operating System. The naming is a bit misleading, as this is not really an operating system, but an environment and sweet of programs and tools that help to develop and run robotics applications. In this post, I want to showcase how I got to run a basic ROS2 publisher on an ESP32, especially as the existing guides on how to do this seem a bit lacking.

So what do you need for following along?

- Ubuntu 22.04 (native or on a VM) with access to a serial port
- an ESP32-dev-kit V1
- a micro USB cable

As the ROS2 node on the ESP32 on its own is not really useful, it makes sense to have ROS2 installed on your host pc. For the installation I recommend you to follow the official instructions and, if you do not have much experience with ROS, do the basic tutorials. In my case, I use ROS2 Humble, as this is the latest version that is supported by micro-ROS right now.

For the firmware build, check out the basics of PlatformIO if you're not familiar with it. All the relevant code samples can be found here.

Once you have ROS2 installed, we can start with the actual topic of this post:

1. Create the micro-ROS Agent

The micro-ROS agent is used to receive the messages from the micro-ROS nodes and make them available in the host system. To build the agent, the following steps are

required (here is the official documentation):

```
# Source the ROS 2 installation
source /opt/ros/$ROS_DISTRO/setup.bash

# Create a workspace and download the micro-ROS tools
mkdir microros_ws
cd microros_ws
git clone -b $ROS_DISTRO https://github.com/micro-ROS/micro_ros_setup.git src

# Update dependencies using rosdep
sudo apt update && rosdep update
rosdep install --from-paths src --ignore-src -y

# Build micro-ROS tools and source them
colcon build
source install/local_setup.bash

# Download micro-ROS-Agent packages
ros2 run micro_ros_setup create_agent_ws.sh

# Build step
ros2 run micro_ros_setup build_agent.sh
source install/local_setup.bash
```

If you want to dive deeper into the topic, I recommend you to check out the official tutorials on the micro-ROS webpage.

2. Upload the Firmware to the ESP32

Below you can see the main part of the example code for the ESP32 firmware. By using the micro_ros_platformio library, the PlatformIO building system can be used, which simplifies the development of the firmware a lot.

```
// We include the Arduino library for general Arduino functionality
#include <Arduino.h>
// The micro_ros_platformio library provides the functions to communicate w
```

```
#include <micro_ros_platformio.h>

// These are core ROS2 libraries for creating nodes, publishers, and executor
#include <rcl/rcl.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>

// This is a standard ROS2 message type, an integer
#include <std_msgs/msg/int32.h>

// Ensure that the transport layer being used is Arduino Serial.
// If it's not, compilation is stopped and error is printed.
#if !defined(MICRO_ROS_TRANSPORT_ARDUINO_SERIAL)
#error This example is only available for Arduino framework with serial trans
#endif

// Define ROS2 objects for a publisher, a message, an executor, support objec
rcl_publisher_t publisher;
std_msgs__msg__Int32 msg;

rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t timer;

// Macros for checking return of ROS2 functions and entering an infinite erro
#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){err
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK))

// Infinite error loop function. If something fails, the device will get stuc
void error_loop() {
  while(1) {
    delay(100);
  }
}

// This is the function that will be called every time the timer expires
void timer_callback(rcl_timer_t * timer, int64_t last_call_time) {
  RCLC_UNUSED(last_call_time);
  if (timer != NULL) {
    // We publish our message here
    RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
```

```
    // The message contains a single integer that we increment each time
    msg.data++;
  }
}

void setup() {
  // Start serial communication with a baud rate of 115200
  Serial.begin(115200);
  // Configure Micro-ROS library to use Arduino serial
  set_microros_serial_transports(Serial);
  // Allow some time for everything to start properly
  delay(2000);

  // Get the default memory allocator provided by rcl
  allocator = rcl_get_default_allocator();

  // Initialize rclc_support with default allocator
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

  // Initialize a ROS node with the name "micro_ros_platformio_node"
  RCCHECK(rclc_node_init_default(&node, "micro_ros_platformio_node", "", &sup

  // Initialize a ROS publisher with the name "micro_ros_platformio_node_publ
  RCCHECK(rclc_publisher_init_default(
    &publisher,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "micro_ros_platformio_node_publisher"));

  // Initialize a timer with a period of 1 second which calls the function ti
  const unsigned int timer_timeout = 1000;
  RCCHECK(rclc_timer_init_default(
    &timer,
    &support,
    RCL_MS_TO_NS(timer_timeout),
    timer_callback));

  // Initialize an executor that will manage the execution of all the ROS ent
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
  // Add our timer to the executor
  RCCHECK(rclc_executor_add_timer(&executor, &timer));

  // Initialize our message data to 0
```

```
  msg.data = 0;
}

void loop() {
  // Wait a little bit
  delay(100);
  // Execute pending tasks in the executor. This will handle all ROS communi
  RCSOFTCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100)));
}
```

Following platformio.ini file is used:

```
[env:esp32doit-devkit-v1]
platform = espressif32
board = esp32dev
framework = arduino
monitor_speed = 115200
lib_deps =
    https://github.com/micro-ROS/micro_ros_platformio
board_microros_distro = humble
```

Now, the ESP32 only needs to obe connected to the serial port of your machine and the firmware uploaded. Note that if you see the error "*Cannot open /dev/ttyUSB0: Permission denied*" check out this solution.

Once the upload is complete, press the EN pin on the ESP32 to restart the code. Now the ROS2 node should be up and running.

3. Testing

In order to recceive the ROS messages on your host PC, you need to start the corresponding micro-ROS agent we created earlier. In our case, the example tries to publish the messages over the serial port. You need to declare this when launching the agent as follows (change the port name if you use another one):

```
ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyUSB0
```

Now, the published messages are accessible in the */micro_ros_platformio_node_publisher* topic, as can be seen by the ***ros2 topic list*** command.

Conclusion

Although the provided example is not really useful as it is (it only publishes an increasing integer value), using the code as a base, more complex publishers and subscribers can be made. These can be used (and will be in the Roboost project) to send velocity commands to the motor drivers over the network, or transfer sensor data from the sensor microcontroller to the board computer.

Note that I have not yet found a good solution to use the *micro_ros_platformio* library for UDP message communication. I will look into the *micro_ros_arduino* library and see if it has better support for that and make a separate post about it.

If you have any questions or think something is missing in this tutorial, please let me know in the comments! 🙂 Thanks for sticking around.

← Previous Post Next Post →

Author

Geibinger 6 months ago

Update:

Since playing with the code for a while, I discovered that the library does not provide support for UDP communication. As this is a really useful feature, especially when using an ESP32, I will resolve to use the *ros2arduino* library, which supports UDP. If in the future, the same functionality is provided by *micro_ros_platformio* (or *micro_ros_arduino*), I will make an update here.

Reply

Author

Geibinger 6 months ago

Reply to Geibinger

Also, *micro_ros_arduino* might support UDP. However, I'm getting compile errors so far.

Reply

Author

Geibinger 6 months ago

Reply to Geibinger

So, update to the update:
I managed to get UDP message transfer working. It is indeed supported by the micro_ros_platformio library, the documentation was really lacking though. An important thing to note is to add

```
board_microros_transport = wifi
```

to the platform.ini file, so that the compiler flags are set correct for the wifi communication

All the examples can be found here.

Reply

Roboost – Primary Motor Cortex – Technologie Hub Wien 4 months ago

[…] micro-ROS specific and handles the ROS communication. More information on micro-ROS can be found on the dedicated post, or the official […]

Reply