



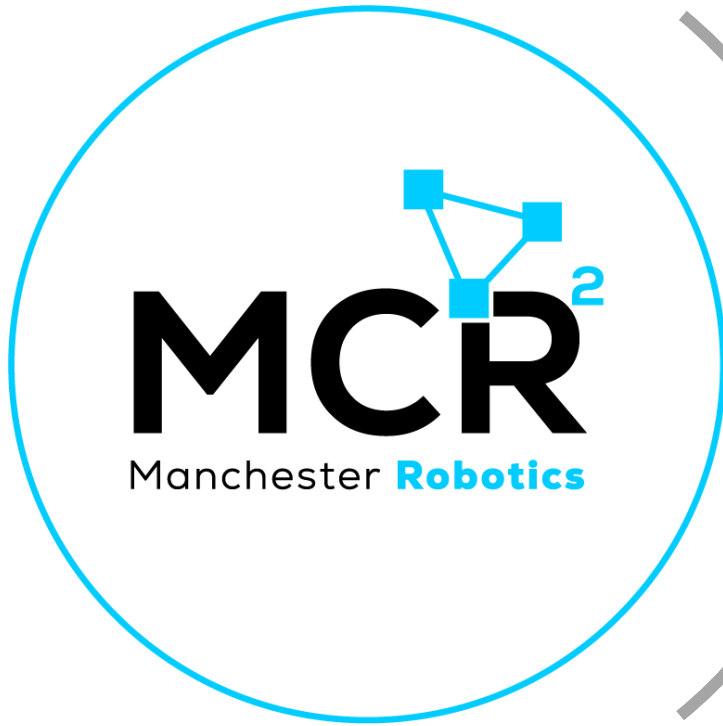
{Learn, Create, Innovate};

ROS serial communication

*Interfacing a
microcontroller and
ROS*



Table of contents



- 1 Micro-ROS: Introduction
- 2 Micro-ROS: Architecture
- 3 MCU Programming
- 4 Micro-ROS Program Structure
- 5 Activity 1 and Activity 1.2: Publisher
- 6 Activity 2 and Activity 2.2: Subscriber
- 7 Activity 3: Publisher and subscriber
- 8 Activity 4: Wi-Fi Publisher



Micro-ROS: Introduction



Introduction

- Micro-ROS (μROS) is an extension of ROS 2 designed for microcontrollers.
- It enables embedded systems to communicate within ROS2 ecosystems.
- Built on ROS 2, DDS (Data Distribution Service), and FreeRTOS, Zephyr, NuttX, or bare-metal systems.
 - **Lightweight:** Optimised for resource-constrained microcontrollers.
 - **Real-time support:** Meets real-time requirements for embedded robotics.
 - **Interoperability:** Seamlessly integrates with ROS2.
 - **Standardised communication:** Uses XRCE-DDS for efficient data transmission.



[micro-ROS | ROS 2 for microcontrollers](#)



Micro-ROS: Introduction

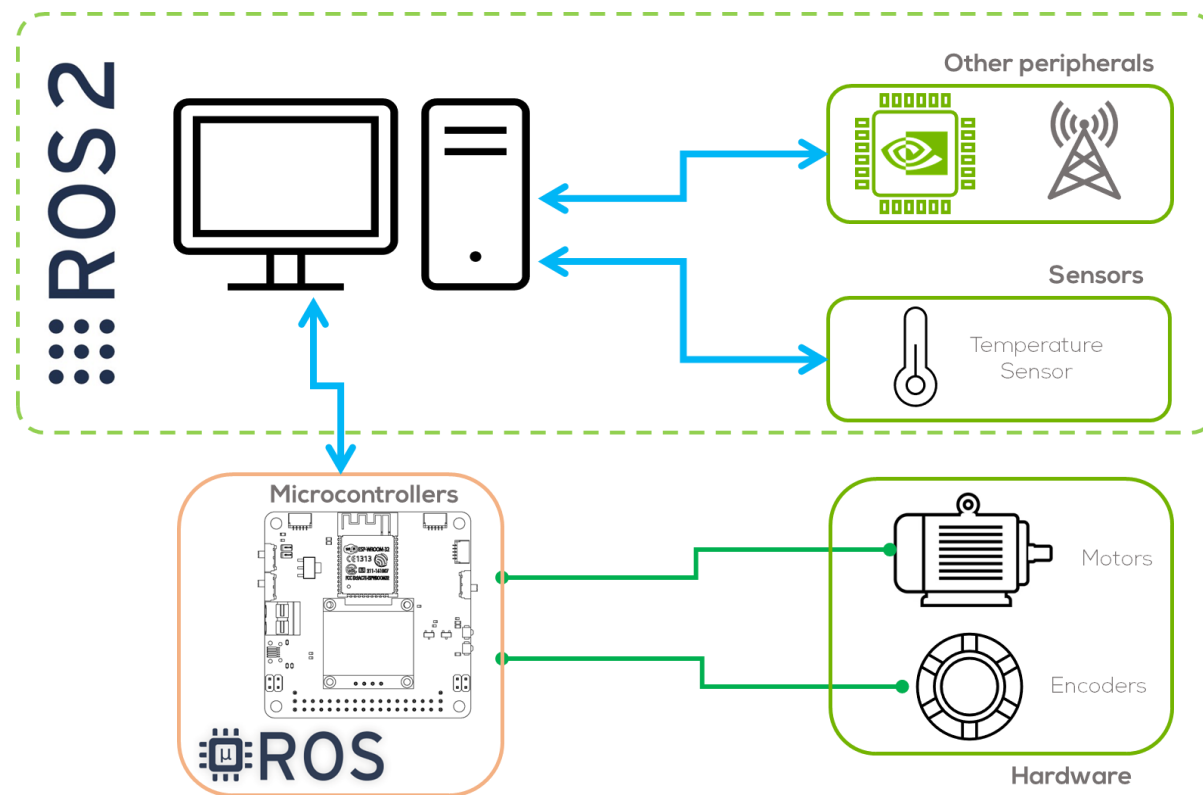


Why Use Micro-ROS?

- Bridges microcontrollers and ROS 2 for enhanced functionality.
- Reduces power consumption compared to full ROS 2 nodes.
- Enables real-time control of actuators and sensors.
- Compatible with various embedded platforms, including STM32, ESP32, and Raspberry Pi Pico.

Common Applications:

- Embedded robotic controllers.
- Autonomous vehicles and drones.
- IoT-based robotic systems.
- Industrial automation and smart sensors.





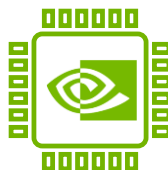
Micro-ROS: Introduction



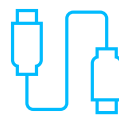
External Computer



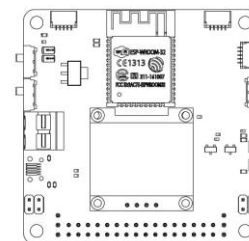
Wi-Fi



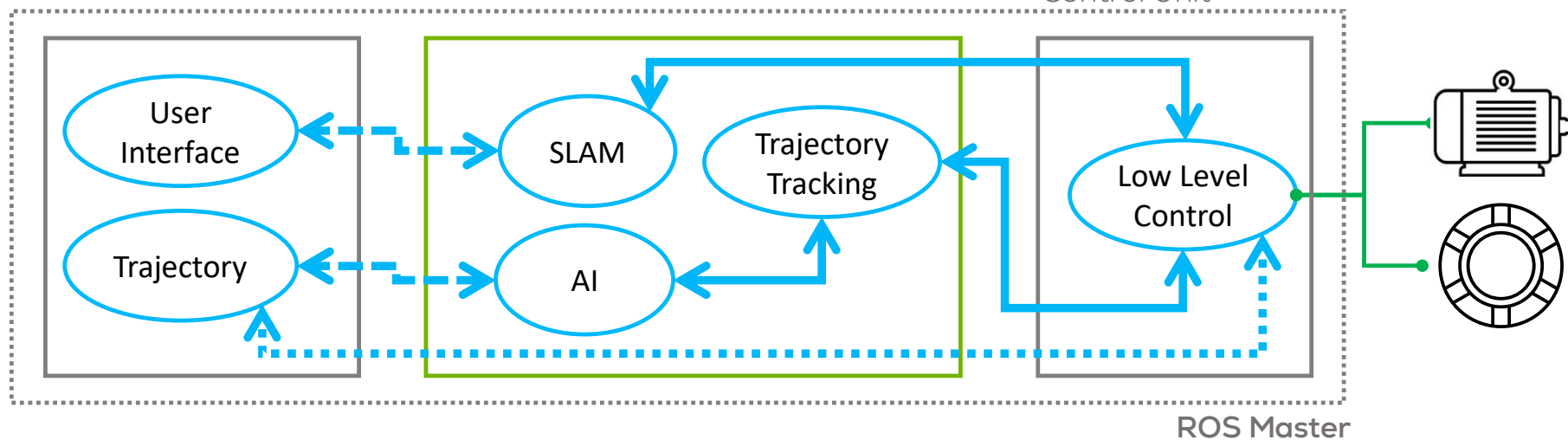
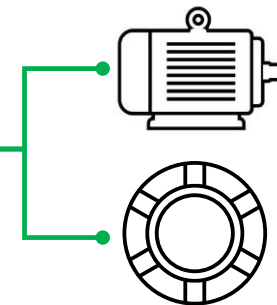
NVIDIA Jetson Nano



Serial

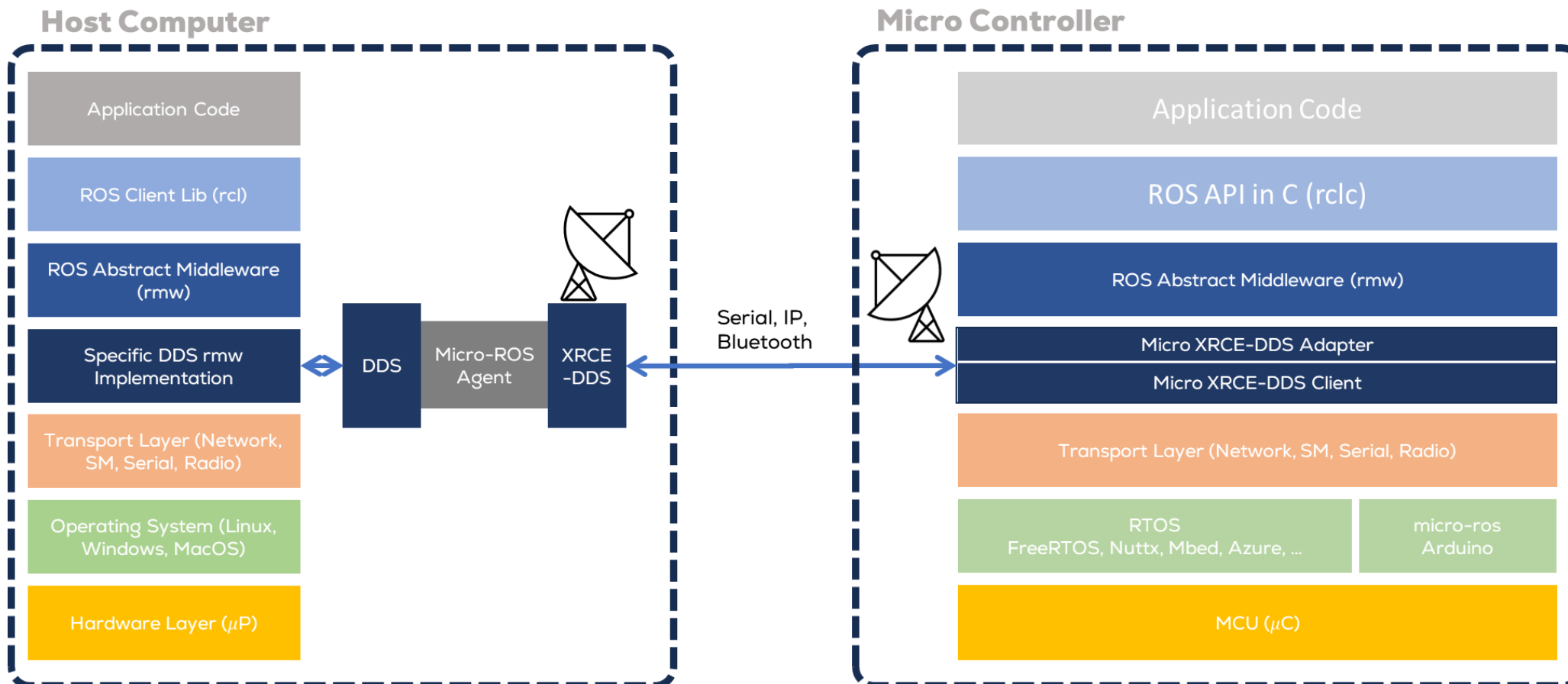


Puzzlebot
Control Unit





Micro-ROS: Architecture



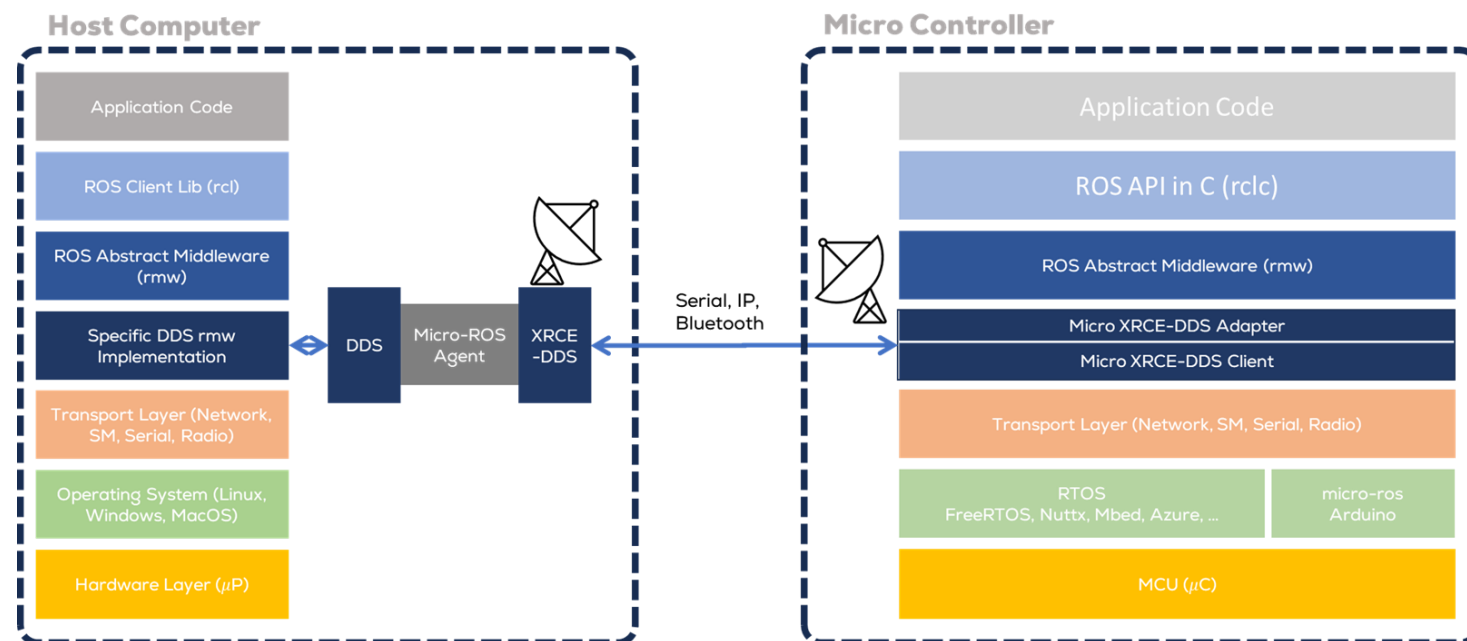


Micro-ROS: Architecture



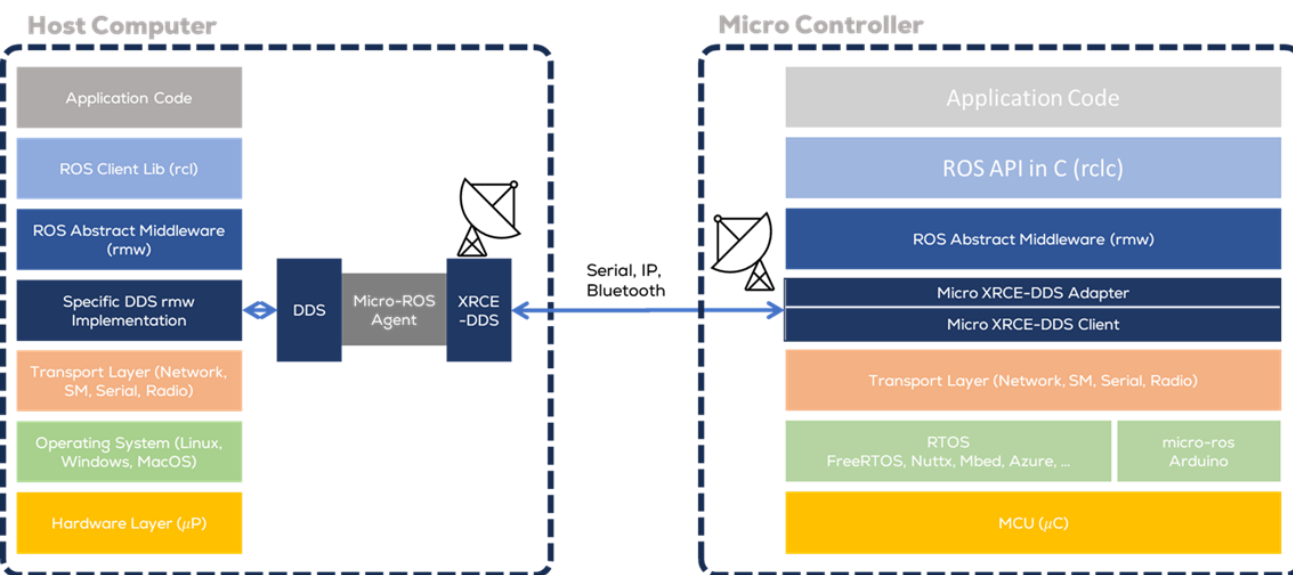
ROS 2 Agent (Left Side)

- The ROS 2 Agent acts as a gateway between the **Micro-ROS Client** (running on a microcontroller) and the **full ROS 2 stack**.
- It runs on a standard operating system (e.g., Ubuntu) and handles **communication** with Micro-ROS clients.
- Communication happens via **Ethernet**, **Bluetooth**, or **Serial** connections.





Micro-ROS: Architecture



Micro-ROS Client (Right Side)

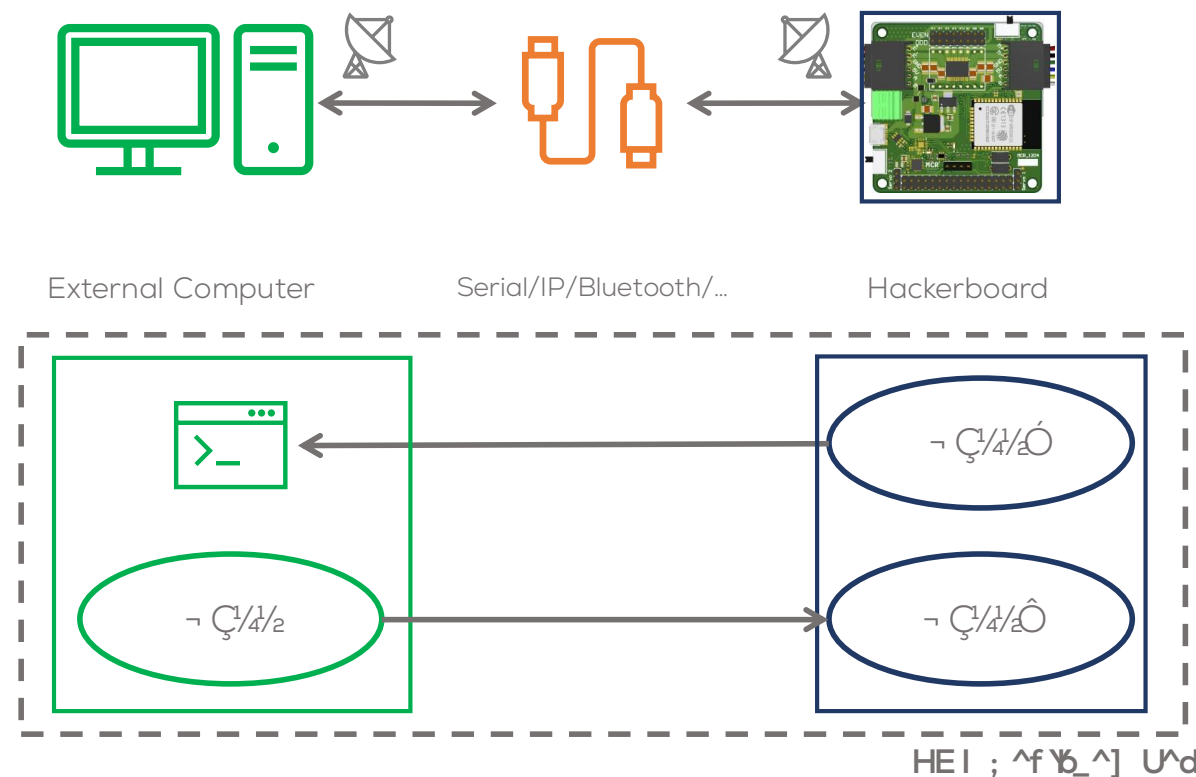
- Runs directly on a microcontroller (μ C) and consists of multiple layers:
- Client Library:
 - ROS API in C (rcl, rcl): Implements ROS functions, including node management, message handling, and execution.
 - Middleware Interface (rmw): Bridges ROS API with communication layers.
 - Micro XRCE-DDS Adapter: Adapts standard DDS (Data Distribution Service) for low-power devices.
- Middleware:
 - Micro XRCE-DDS Client: Implements lightweight DDS communication, allowing microcontrollers to publish and subscribe to ROS 2 topics with minimal overhead.
- RTOS (Real-Time Operating System) Support:
 - Zephyr, FreeRTOS, and NuttX provide real-time capabilities needed for embedded applications.
 - Micro-ROS Arduino: Arduino Bare Metal implementation.



Micro-ROS: Architecture



- Unlike a computer running with ROS, the dedicated OS of the microcontrollers, allows the user to have more control over the timing functions required for certain hardware and control algorithms.
- Micro-ros allows the board to become part of the ROS2 environment (“creating nodes”) that can directly publish and subscribe to ROS2 messages, publish TF transforms, and get the ROS2 system time.

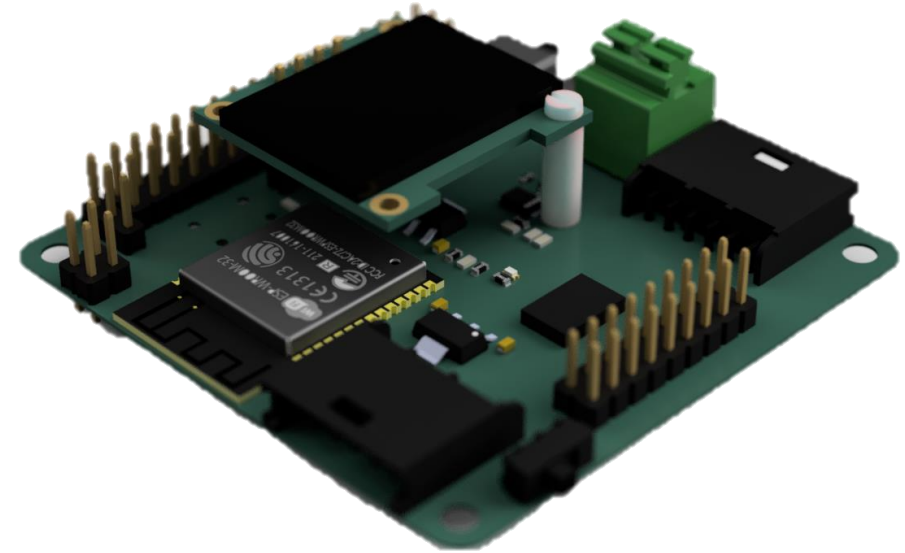




Micro-ROS Compatibility



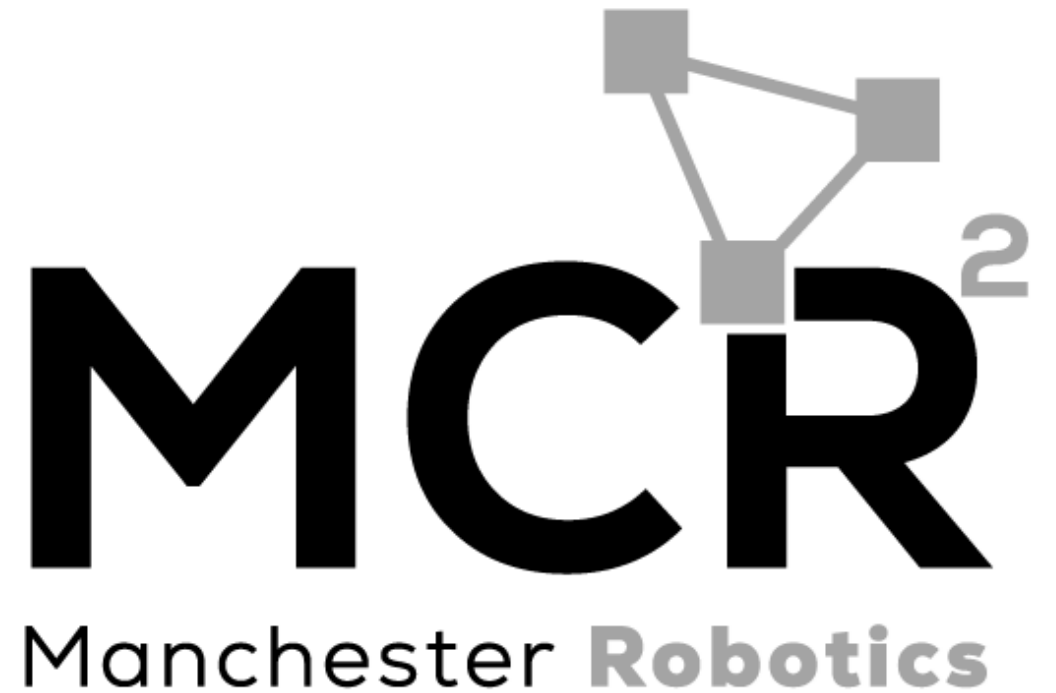
- Not all the microcontrollers can be used with micro-rose.
- Renesas EK RA6M5, ESP32, STM and Arduino are some microcontrollers that micro-ROS support.
- No support for Arduino UNO or Arduino MEGA2560



Micro-ROS Communication

MCU Programming

{Learn, Create, Innovate};



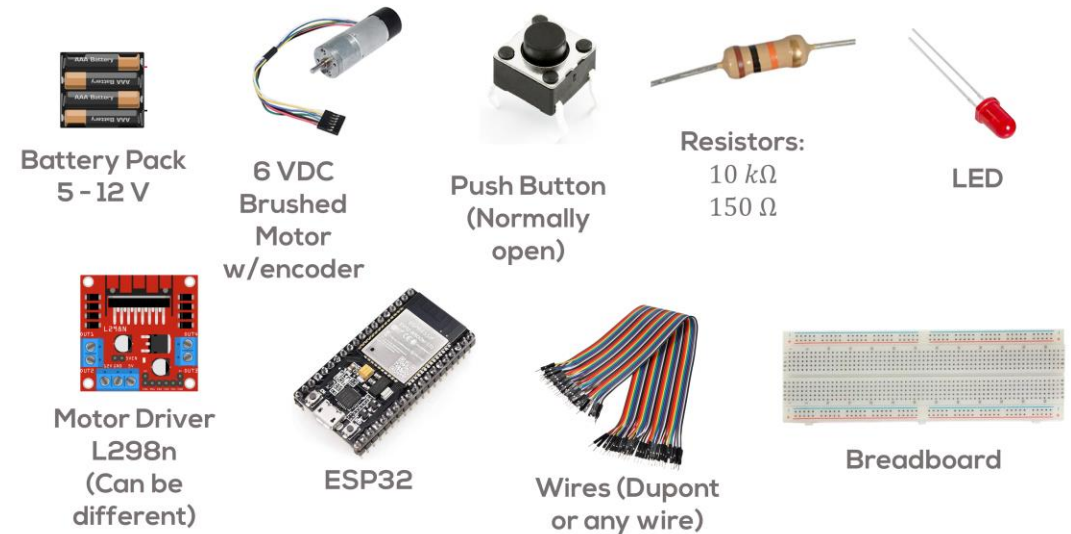


MCU Programming



General information

- As stated before, STM32, Arduino and ESP32 are some of the most used development platforms because of their ease of use.
- Arduino and ESP32 boards can be programmed using the Arduino IDE.
- For all the activities and challenges in this session, the Arduino IDE will be used for programming (Other IDE's can be used such as Platform IO).
- The activities and challenges shown in this presentation will be performed using a ESP32 WROOM or a MCR2 Hackerboard.
- Please refer to the prerequisites of this session for the complete list of required components.





MCU Programming



Arduino IDE

- An IDE, or Integrated Development Environment, helps programmers' productivity by combining common activities of writing software into a single application: editing source code, building executables, and debugging.
- Arduino IDE supports C and C++ programming languages.
- A sketch is a program written with the Arduino IDE.
- Sketches are saved on the development computer as text files with the file extension .ino.

A screenshot of the Arduino IDE 2.1.1 interface. The window title is 'nlights_1Way_arduino | Arduino IDE 2.1.1'. The menu bar includes 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for opening files, saving, compiling, and uploading. A dropdown menu shows 'Arduino Mega or Mega 2560'. The main editor area displays a sketch named 'nlights_1Way_arduino.ino' with the following code:

```
1 // Traffic Light Basic Code
2 /* Kat Nelms, Manchester Robotics, Feb 19 2022
3 // adapted from Arduino Neopixel Library "RGBWstrandtest"
4 // "Nlight_oneway" = choose between one or two TLBs, only one-way traffic
5 // one MCU board (MCUB) and up to two traffic lights (TLB)
6 // next level of complexity up from "1int_1way" script
7 // should initialize TLB1 to green and, if TLB2, TLB2 initialized to red
8 // then change green > yellow > red > yellow > green according to hard coded timers
9 */
10
11 /* SETUP */
12 #define LED_PIN 52 // Define the data out pin on the ESP32
13 #define greenTimer 2 //how long each color lasts, units are sec
14 #define redTimer 2
15 #define yellowTimer 1
16 /* END SETUP */
17
18
```

The bottom of the window shows the 'Output' and 'Serial Monitor' tabs. The status bar at the bottom indicates 'Ln 195, Col 36' and 'Arduino Mega or Mega 2560 [not connected]'.



MCU Programming



Sketch

- The simplest syntax for writing a sketch consists of only two functions:
- `setup()`: This function is called once when a sketch starts after power-up or reset. It is used to initialize variables, input and output pin modes, and other libraries needed in the sketch. It is analogous to the function `main()`.
- `loop()`: The `loop()` function is executed repeatedly in the main program after the `setup()` function. It controls the board until the board is powered off or is reset.

Sketch Structure

```
// Variable declaration section
```

```
// the setup function runs once when you press reset or power the board
```

```
void setup() {  
  // initialize digital pin LED_BUILTIN as an output.  
  pinMode(LED_BUILTIN, OUTPUT);  
}
```

```
// the loop function runs over and over again forever
```

```
void loop() {  
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)  
  delay(1000); // wait for a second  
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW  
  delay(1000); // wait for a second  
}
```

Variable Declaration:
Libraries, Components,
Variables, constants,
Definitions, etc.

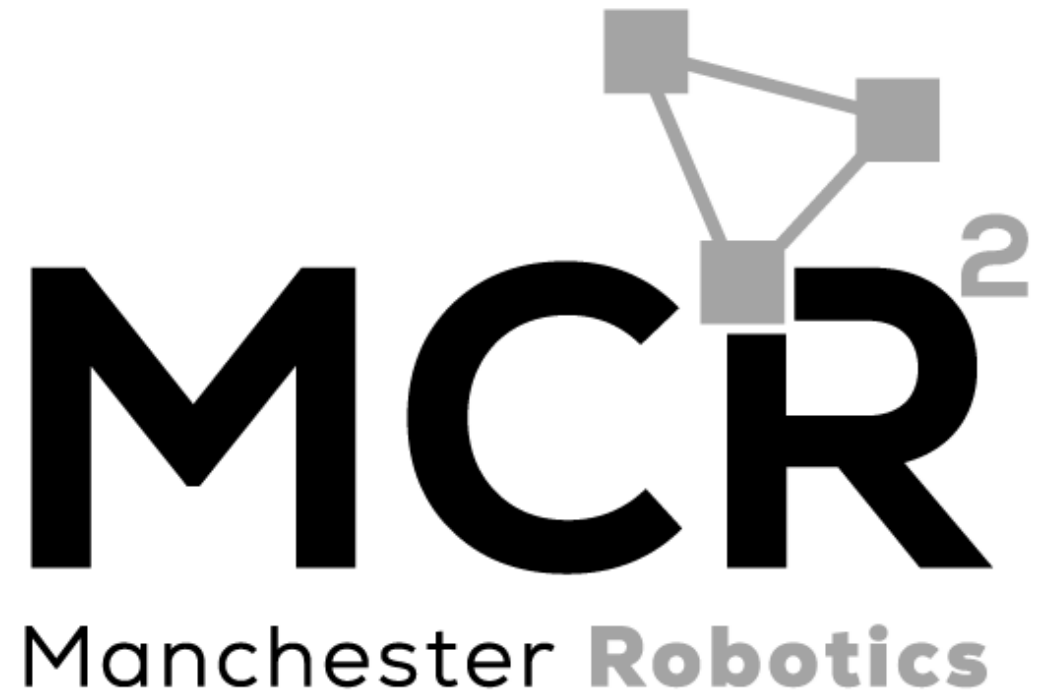
Setup Section:
Set up sensors,
variables, Ports,
Functions, Serial comms.

Loop Section:
Loops and repeats
actions.

Micro-ROS Communication

*Micro-ros Program
Structure*

{Learn, Create, Innovate};





Micro-ros Program Structure



Micro-ros program architecture

- Unlike traditional Arduino sketches, Micro-ROS follows a modular and event-driven architecture.
- Instead of a simple setup() and loop(), micro-ros uses an event driven approach, where an **executor** dynamically **schedules tasks** based on the user or ROS2 events.
- Micro-ros does not rely on an OS-based scheduler (like Linux), so the executor provides structured task execution.
- It guarantees that each event (message, timer, service request) is processed in a controlled manner.

Normal Arduino Program Structure

Libraries to be used.

Setup:
Instantiate objects,
Configure the variables and
libraries to be used.

Loop:
Run the program
sequentially and in a loop

Micro ROS Program Structure

Libraries to be used.

Setup:

- Instantiate objects (timers, subscribers, etc.)
- Define callbacks
- Configure the variables
- Config executor

Loop:
Executor(checkCallbacks)
if (callback ready || new data || trigger)
Execute(callbacks)



Micro-ros Program Structure



What is the Micro-ROS Executor?

- The **executor** is a key component in Micro-ROS that **manages callbacks asynchronously**, ensuring **efficient execution of multiple tasks**.
- It is responsible for handling:
 - **Timers** (periodic function execution).
 - **Subscriptions** (processing incoming ROS 2 messages).
 - **Services** (handling requests and responses).
 - **Publishers** (sending messages to the ROS 2 network).

Why an Executor?

- Ensuring predictable execution under real-time constraints is essential for many robotic applications.
- The service-based paradigm of ROS lacks fine-grained control over execution management, i.e., there are no built-in mechanisms to enforce a specific execution order for callbacks within a node.

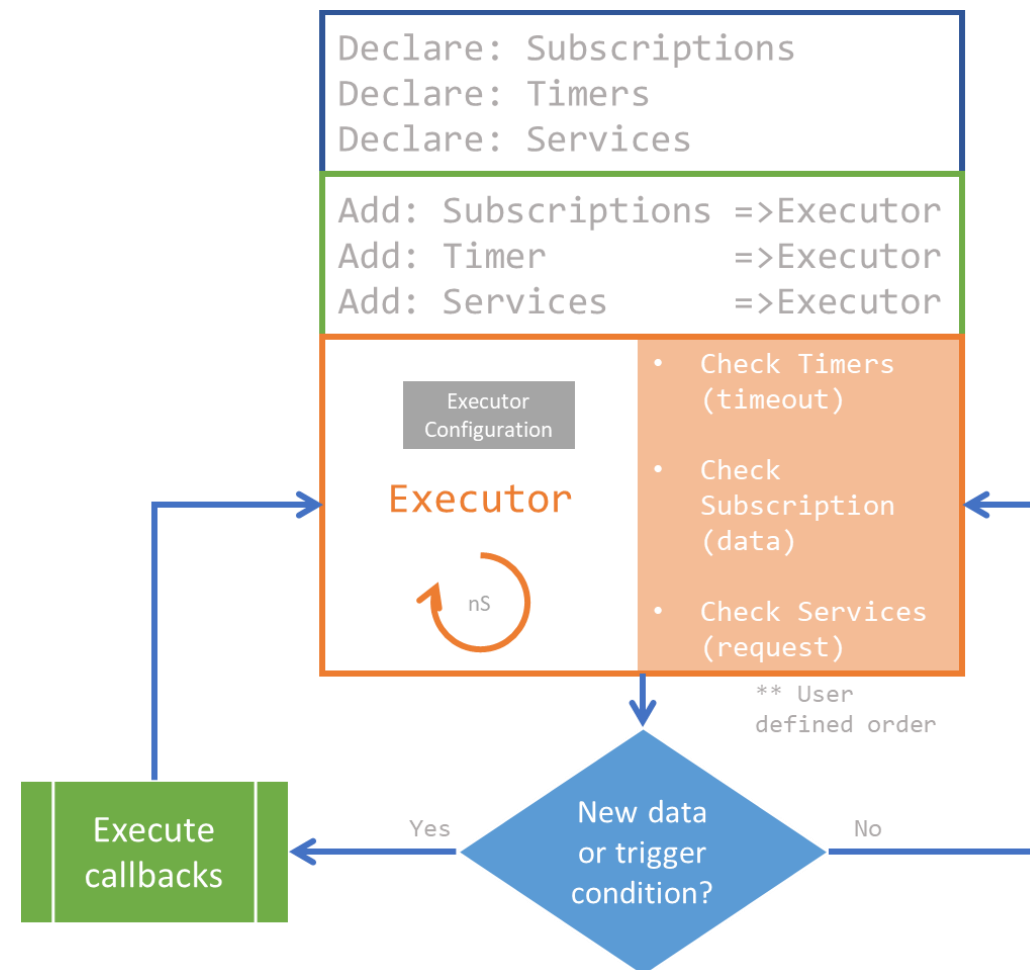


Micro-ros Program Structure



How the Executor Works in Micro-ROS

- Micro-ROS Initializes the Handlers (timers, subscribers, etc.) and Executors.
 - The executor is created and linked to a ROS 2 node.
 - Timers, subscriptions, publishers are created.
- Callbacks Are Added to the Executor
 - The developer registers subscriptions, timers, and services to the executor.
 - Each callback function is assigned to a specific event (e.g., new message received).
- Executor Runs in loop()
 - The executor “spins” (This checks for incoming messages or scheduled events)
 - Executes the corresponding callbacks.





ROS2 Arduino Sketch Structure



Init Section

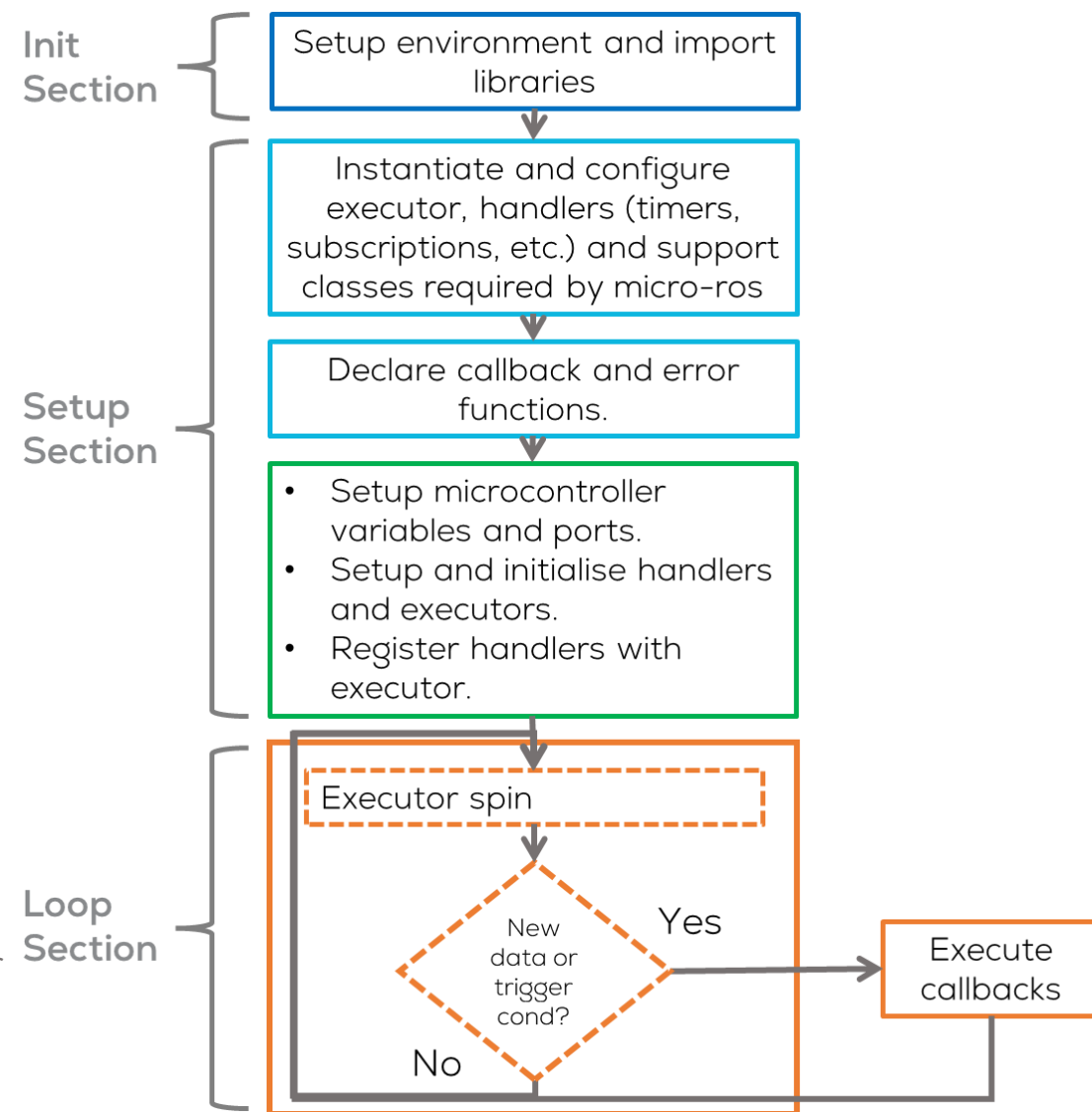
- Setup Environment
- Import ROS and user libraries

Setup section

- Instantiate the executor, handlers and support objects required by micro-ROS.
- Declare callback and error functions (if required).
- Initialise variables, ports, functions, etc.
- Register declared handlers with executor.
- Configure Executor.

Loop section

- Spin Executor.
- Execute Callback, according to executor configuration (New data or trigger conditions).





Executor Configuration



Steps to Configure an Executor in Micro-ROS

1. Initialize Memory Allocator: Required for memory allocation in Micro-ROS.
2. Initialize Support and Create a Node:
 - Support: manages the execution context of Micro-ROS, including its communication state, memory management, and initialisation data
 - Node: acts as a ROS 2 processing unit.
3. Create and Configure the Executor:
4. Initializes an executor.
 - 1 in `rcl_executor_init()` defines the number of handles (e.g., timers, subscribers).
5. Add Callbacks (e.g., Timer) to the Executor
6. Process Executor in `loop()`

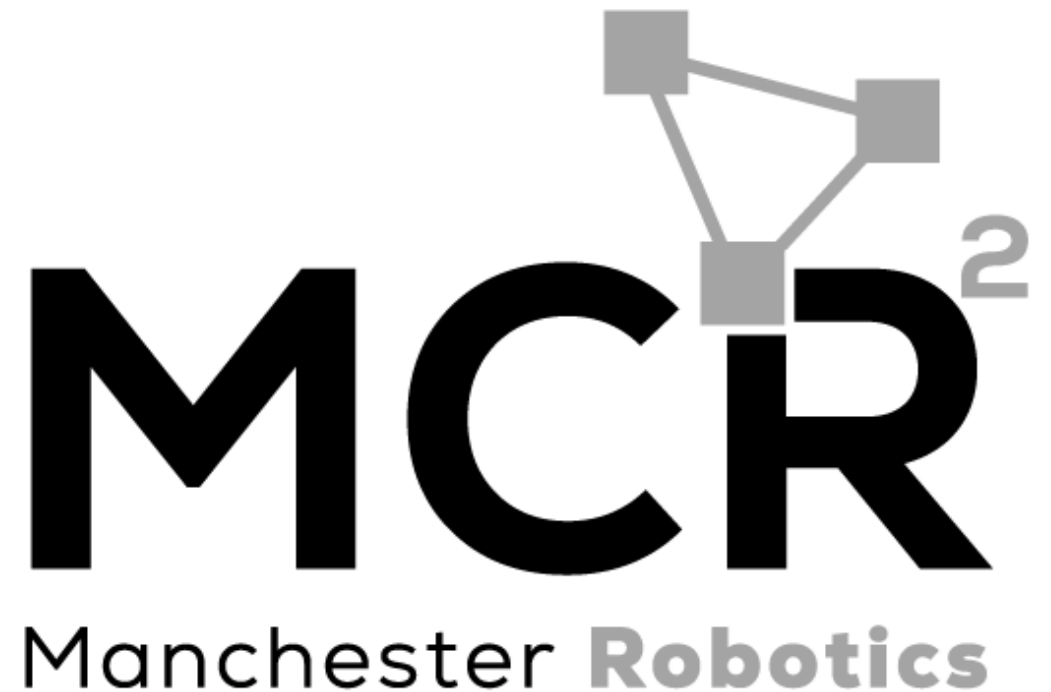
```
// Include Libraries to be used
#include <micro_ros_arduino.h> //micro-ros-arduino library
#include <rcl/rcl.h> //Core ROS 2 Client Library (RCL) for node management.
#include <rcl/error_handling.h> //Error handling utilities for Micro-ROS.
#include <rcl/rcl.h> //Micro-ROS Client library for embedded devices.
#include <rcl/rcl.h> //Micro-ROS Executor to manage callbacks
//Instantiate executor and its support classes
rcl_executor_t executor; //Manages task execution (timers, callbacks, etc.).
rcl_support_t support; //Data structure that holds the execution context of Micro-ROS,
//including its communication state, memory management, and initialization data.
rcl_allocator_t allocator; //Manages memory allocation.
//Setup
void setup() {
...
//Initializes memory allocation for Micro-ROS operations.
allocator = rcl_get_default_allocator();
//Creates a ROS 2 support structure to manage the execution context.
RCL_CHECK(rcl_support_init(&support, 0, NULL, &allocator));
// create node
RCL_CHECK(rcl_node_init_default(&node, "micro_ros_sub_node", "", &support));
// create executor
RCL_CHECK(rcl_executor_init(&executor, &support.context, 1, &allocator));
// Register subscription with the executor
RCL_CHECK(rcl_executor_add_subscription(&executor, &subscriber, &msg,
&subscription_callback, ON_NEW_DATA));
// Register timer with executor
RCL_CHECK(rcl_executor_add_timer(&executor, &timer));
}

void loop() {
delay(100);
RCL_CHECK(rcl_executor_spin_some(&executor, RCL_MS_TO_NS(100))); //Executor Spin
}
```

Micro-ROS Serial Communication

*Activity 1: Simple
Publisher*

{Learn, Create, Innovate};





Requirements



- The following activity is based on the example tutorial found in the provided micro-ROS libraries.
- This activity requires Arduino IDE to be installed and configured as shown in “MCR2_Micro_ROS_Installation”.
- Requirements:
 - Microcontroller
 - Computer
 - micro-usb to USB cable (Data)
 - Breadboard
 - LED
 - 150 Ohm Resistor

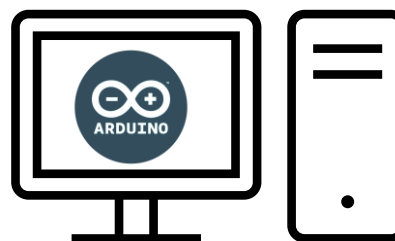


Or

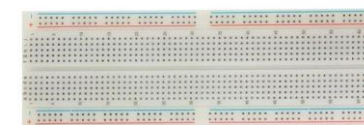


Hackerboard

ESP32 board



Computer



Breadboard



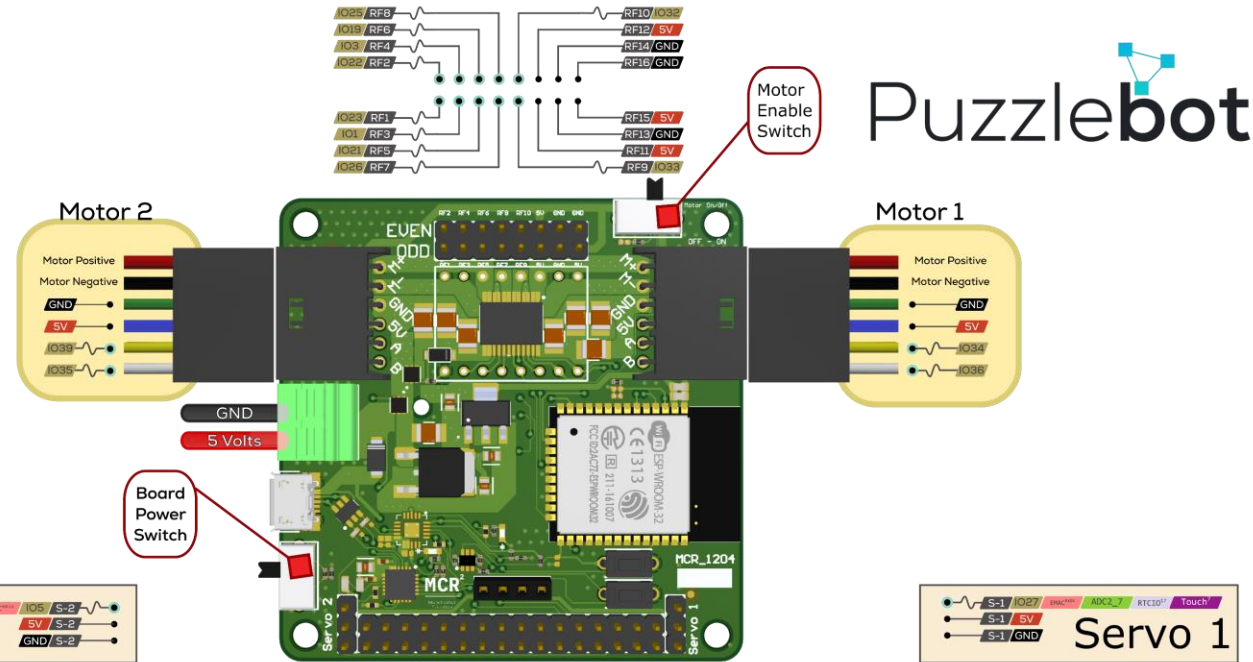
LED



150 Ω Resistor



Hackerboard Pinout



- 5V comes from LDO regulated supply, max current draw is 800mA
- Absolute maximum current draw per GPIO pin is 40mA
- Input voltage is 5-13.5V via terminal block
- Current draw via microUSB socket is 1.8A max

- Puzzlebot Devices
- Power
- GND
- Serial Pin
- Analogue Pin
- Control Pin
- Physical Pin Number
- GPIO Pin
- Touch Pin
- DAC Pin
- Port Pin
- PWM Pin

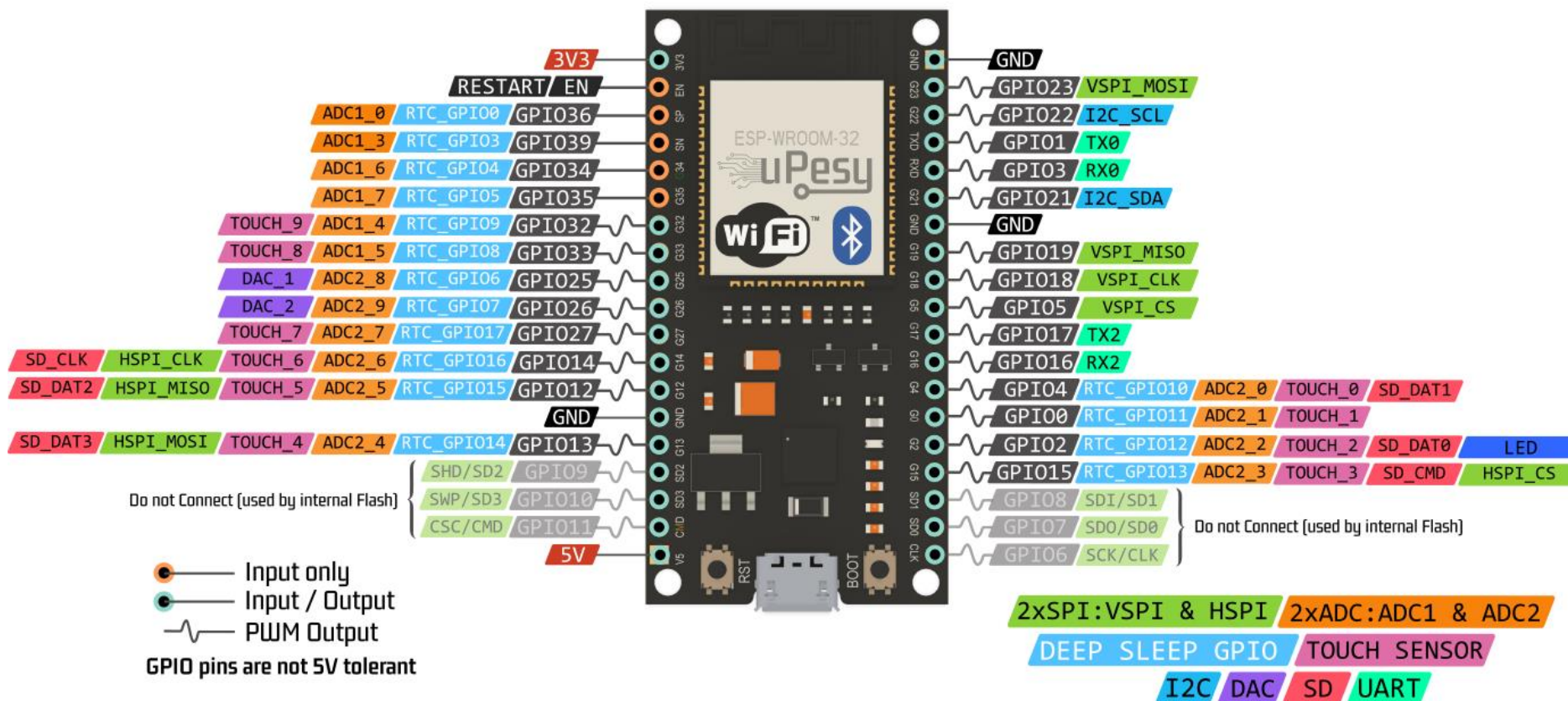
- Notes:**
- These pins are **input only**. They cannot output signals
 - These pins are integrated to the Microcontroller's flash memory, and are only present for debugging purposes. They should not be used for general purpose
 - Outputs PWM signal at boot time
 - Boot fails if pulled high
 - Boot fails if pulled low

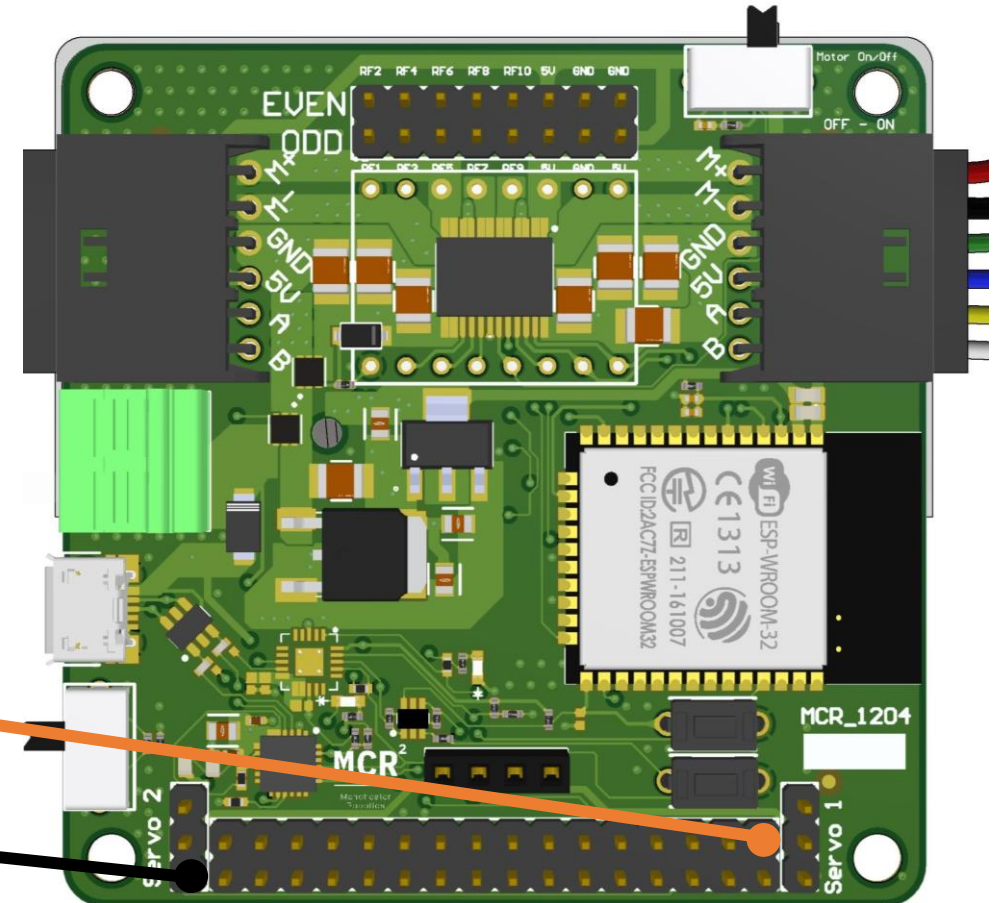
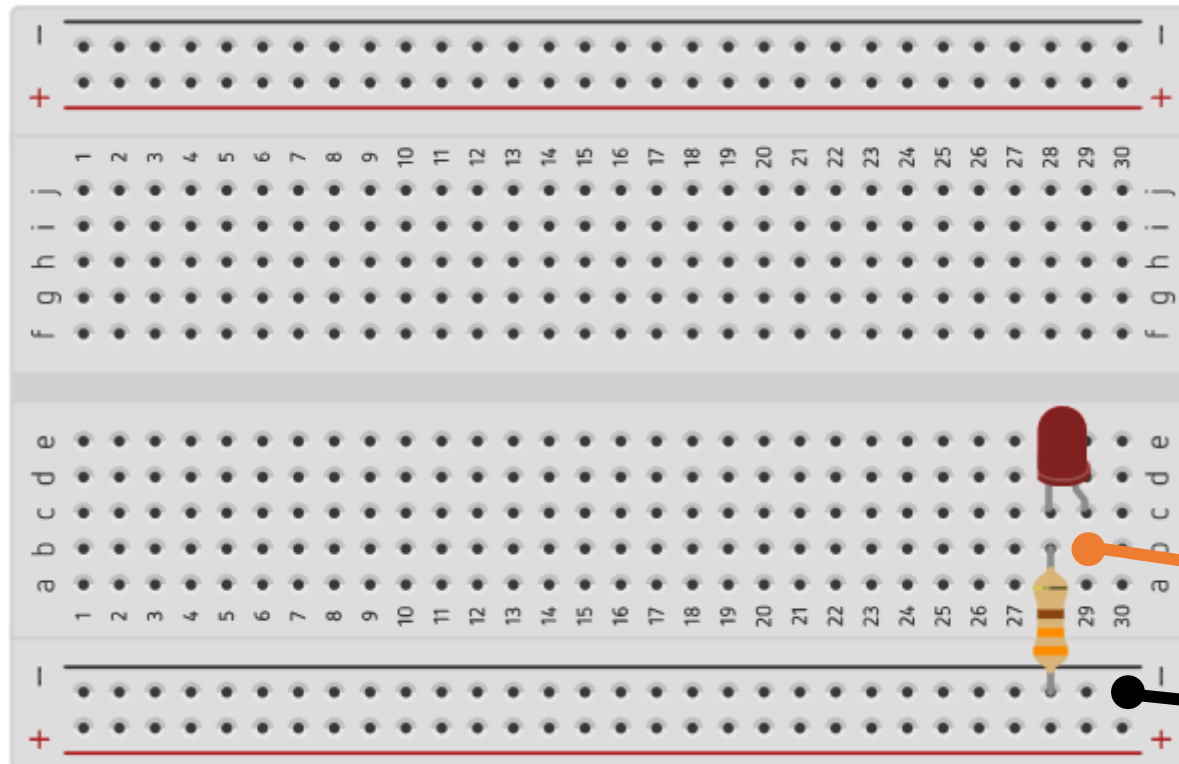
<https://www.manchester-robotics.com>

01 FEB 2022
v1

Derivative of "Puzzlebot Template" by AthakronOrg, used under CC BY SA.
MCR_1204 Pinout is licensed under CC BY SA by MCR

ESP32 Wroom DevKit Full Pinout





GND

Pin 22

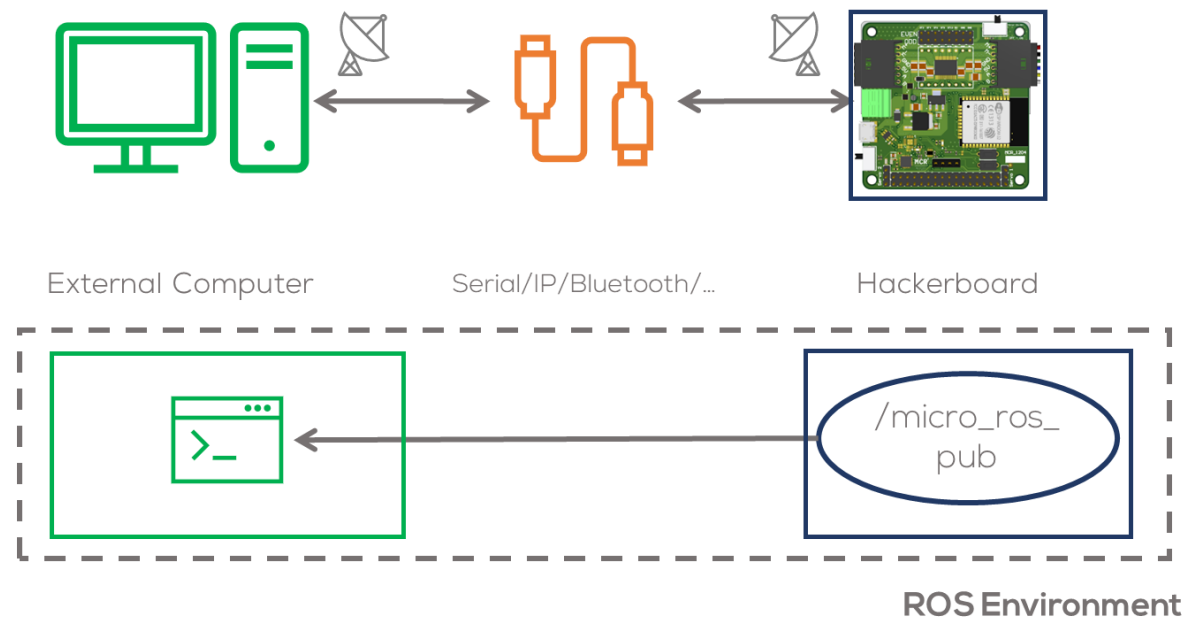


Introduction



Introduction

- In this activity, a node running a simple publisher inside the microcontroller will be declared.
- This node will run inside the microcontroller and will communicate with the computer via UART.
- The node will publish a simple int32 message.
- This activity will be divided into two parts. The first part involves the Arduino IDE to program the MCU.
- The second part involving the commands required to connect to the board to the computer.





Publisher set up



Configuring a Publisher in Micro-ROS

What is a Publisher?

- A publisher sends data to a ROS 2 topic, allowing other nodes to receive it.
- In Micro-ROS, publishers work by sending messages periodically or on-demand.

1. Define the Message Type
2. Initialize the Publisher
3. Publish Messages Periodically
4. Use a Timer or loop to Publish at Intervals

```
#include <std_msgs/msg/int32.h> //Predefined ROS 2 message type

//Declare Messages to be used
std_msgs__msg__Int32 msg; //Defines a message of type int32.
//Declare Publishers to be used
rcl_publisher_t publisher; //Declares a ROS 2 publisher for sending
messages.

...
void setup() {
    // create publisher
    RCCHECK(rclc_publisher_init_default(
        &publisher,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
        "micro_ros_counter"));
}
void loop() {
    ...
    //Fill Message
    msg.data = 2;
    //Publishes msg to the ROS 2 topic.
    RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
    delay(100);
}
```



Timer set up



Configuring a Timer in Micro-ROS

What is a Timer in Micro-ROS?

- A timer executes a function at fixed intervals without blocking the system.
- Useful for periodic publishing, sensor readings, and state updates. Define the Message Type

1. Define a timer and a callback function
2. Initialize the Timer inside the setup and define a timeout.
3. Add Timer to the Executor
4. Process the Executor in loop()

```
//Declare timers to be used
rcl_timer_t timer;           //Creates a timer

void timer_callback(rcl_timer_t * timer, int64_t last_call_time) {
    (void) last_call_time;
    if (timer != NULL) {
        Serial.println("Timer triggered!");
    }
}

void setup() {
    const unsigned int timer_timeout = 1000; // 1 second
    rclc_timer_init_default(
        &timer,
        &support,
        RCL_MS_TO_NS(timer_timeout),
        timer_callback);

    // Initializes the Micro-ROS Executor, which manages tasks and callbacks.
    RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
    // Register timer with executor
    RCCHECK(rclc_executor_add_timer(&executor, &timer));
}

void loop() {
    //Executor Spin
    delay(100);
    RCSOFTCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100)));
}
```

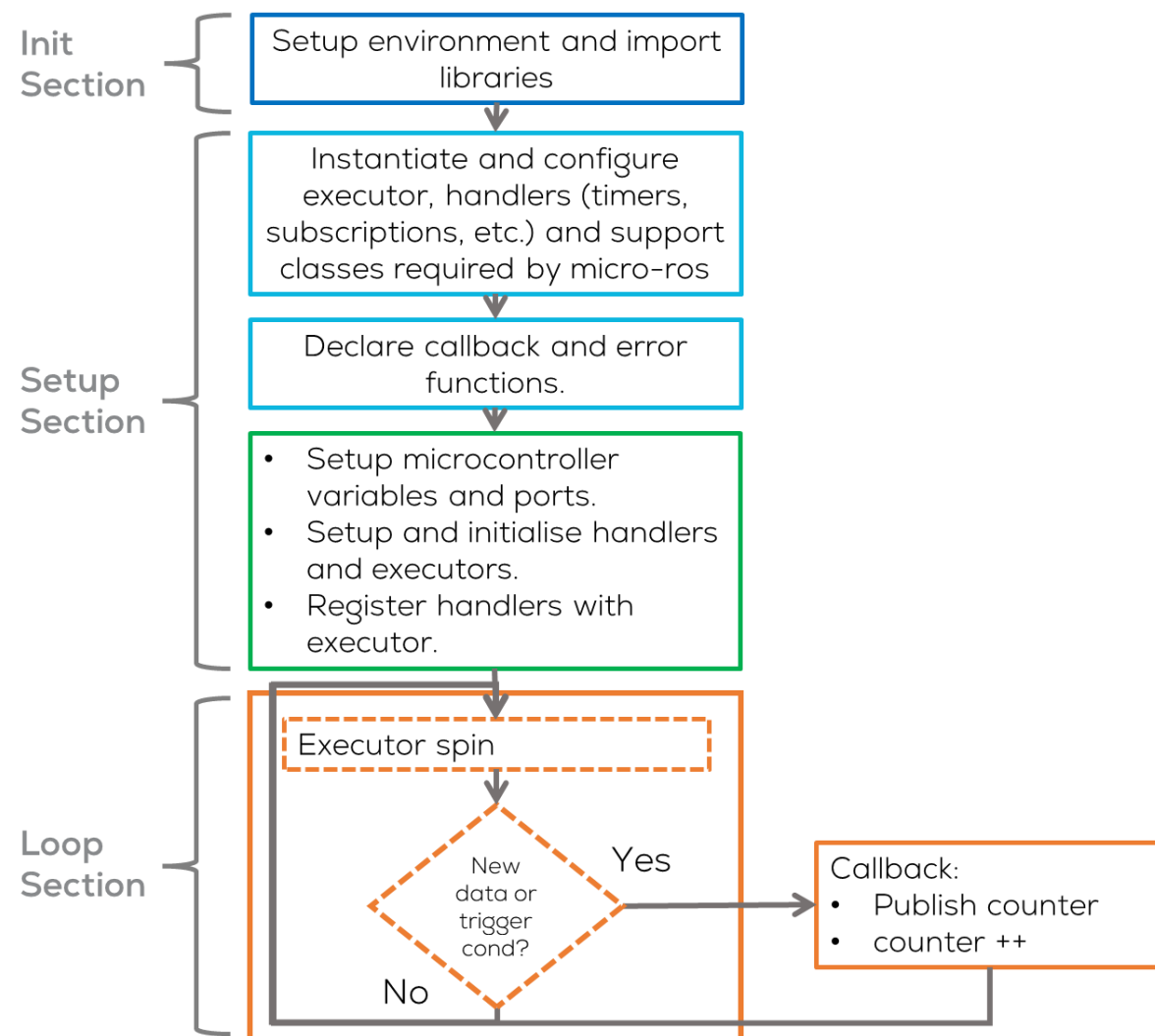


Micro-ros Publisher



Introduction

- The node will be named "micro_ros_pub"
- The node must perform the following:
 - Setup a timer to run control the publishing of information.
 - When the timer times out, the node will publish the value of a counter in the topic "*micro_ros_counter*".
 - The value of the counter must be increased.
- The computer will receive display subscribe to that topic and publish the information on the terminal.





Activity



Instructions (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board
ESP32 or Arduino Due
 - For ESP32 select ESP32 Arduino > DOIT
ESP32 DEVKIT V1
- Copy and paste the following code (next slides)

A screenshot of the Arduino IDE 2.1.1 interface. The main window displays a C++ code file named 'nlights_1Way_arduino.ino'. The code is a traffic light simulation. The top menu bar includes 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. Below the menu bar, there are icons for checking, compiling, and uploading code, along with a dropdown menu currently set to 'Arduino Mega or Mega 2560'. The code editor shows the following code:

```
1 // Traffic Light Basic Code
2 /* Kat Nelms, Manchester Robotics, Feb 19 2022
3 // adapted from Arduino Neopixel Library "RGBWstrandtest"
4 // "Nlight_oneway" = choose between one or two TLBs, only one-way traffic
5 // one MCU board (MCUB) and up to two traffic lights (TLB)
6 // next level of complexity up from "lint_1way" script
7 // should initialize TLB1 to green and, if TLB2, TLB2 initialized to red
8 // then change green > yellow > red > yellow > green according to hard coded timers
9 */
10
11 /* SETUP */
12 #define LED_PIN 52 // Define the data out pin on the ESP32
13 #define greenTimer 2 //how long each color lasts, units are sec
14 #define redTimer 2
15 #define yellowTimer 1
16 /* END SETUP */
17
18
19
```

The bottom of the IDE shows a 'Serial Monitor' tab which is currently empty. The status bar at the bottom indicates 'Ln 195, Col 36' and 'Arduino Mega or Mega 2560 [not connected]'.



Micro-ros Publisher



```
// Include Libraries to be used
#include <micro_ros_arduino.h> //micro-ros-arduino library
#include <rcl/rcl.h> //Core ROS 2 Client Library (RCL) for node management.
#include <rcl/error_handling.h> //Error handling utilities for Micro-ROS.
#include <rcl/rcl.h> //Micro-ROS Client library for embedded devices.
#include <rcl/executor.h> //Micro-ROS Executor to manage callbacks
#include <std_msgs/msg/int32.h> //Predefined ROS 2 message type
#include <stdio.h> //Standard I/O library for debugging.

//Declare nodes to be used
rcl_node_t node; //Represents a ROS 2 Node running on the microcontroller.

//Instantiate executor and its support classes
rcl_executor_t executor; //Manages task execution (timers, callbacks, etc.).
rcl_support_t support; //Handles initialization & communication setup.
rcl_allocator_t allocator; //Manages memory allocation.

//Declare Publishers to be used
rcl_publisher_t publisher; //Declares a ROS 2 publisher for sending messages.

//Declare timers to be used
rcl_timer_t timer; //Creates a timer to execute functions at intervals.

//Declare Messages to be used
std_msgs__msg__Int32 msg; //Defines a message of type int32.

//Define Macros to be used
//Executes fn and calls error_loop() if it fails.
#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){error_loop();}}
// Executes fn, but ignores failures.
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){} }

#define LED_PIN 22 //Specifies GPIO pin 13 for controlling an LED
```

Init Section

- Libraries:
 - micro_ros_arduino: Main Micro-ROS library for Arduino
 - RCL (Ros Client Library): Core ROS 2 Client Library (RCL) for node management.
 - RCLC: Micro-ROS Client library for embedded devices.
 - std_msgs: ROS messages library
- Instantiate the executor
- Instantiate the publisher and timer
- Define Macros to verify if there is an error
 - RCCHECK(fn) → Executes fn and calls error_loop() if it fails.
 - RCSOFTCHECK(fn) → Executes fn, but ignores failures.
- Define Output pins



Micro-ros Publisher



```
//Define Error Functions
void error_loop(){
    while(1){
        // Toggle LED state
        digitalWrite(LED_PIN, !digitalRead(LED_PIN));
        // Wait 100 milliseconds
        delay(100);
    }
}

//Define callbacks
void timer_callback(rcl_timer_t * timer, int64_t last_call_time)
{
    //Prevents compiler warnings about an unused parameter.
    RCLC_UNUSED(last_call_time);
    //Ensures the timer event is valid before executing actions.
    if (timer != NULL) {
        //Publishes msg to the ROS 2 topic.
        RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
        //Increments the integer message value.
        msg.data++;
    }
}
```

Setup Section (Functions)

Define callback and error functions.

- Error Handling Function (error_loop()):
 - This function is called when a **critical error** occurs in Micro-ROS (LED blinks continuously).
 - The microcontroller **enters an infinite loop** to indicate an error, preventing the system from continuing in an **unstable state**.
- Timer Callback Function (timer_callback())
 - This function is triggered periodically by the Micro-ROS timer. The function publishes a message (msg) to a ROS 2 topic.
 - After each execution, msg.data is incremented.



Micro-ros Publisher



```
void setup() {  
  // Initializes communication between ESP32 and the ROS 2 agent (Serial).  
  set_microros_transports();  
  
  //Setup Microcontroller Pins  
  pinMode(LED_PIN, OUTPUT);  
  digitalWrite(LED_PIN, HIGH);  
  
  //Connection delay (waiting for agent to be available)  
  delay(2000);  
  
  //Initializes memory allocation for Micro-ROS operations.  
  allocator = rcl_get_default_allocator();  
  
  //Creates a ROS 2 support structure to manage the execution context.  
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));  
  
  // create node  
  RCCHECK(rclc_node_init_default(&node, "micro_ros_pub_node", "", &support));  
  
  // create publisher  
  RCCHECK(rclc_publisher_init_default(  
    &publisher,  
    &node,  
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),  
    "micro_ros_counter"));
```

```
  // create timer,  
  const unsigned int timer_timeout = 1000;  
  RCCHECK(rclc_timer_init_default(  
    &timer,  
    &support,  
    RCL_MS_TO_NS(timer_timeout),  
    timer_callback));  
  
  // Initializes the Micro-ROS Executor, which manages tasks and callbacks.  
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));  
  // Register timer with executor  
  RCCHECK(rclc_executor_add_timer(&executor, &timer));  
  
  // Initialise message  
  msg.data = 0;  
}
```



Micro-ros Publisher



Setup Section

- `set_microros_transports()`: Initializes communication between ESP32/Arduino and the ROS 2 agent (via Serial, Wi-Fi, Ethernet, etc.).
- `delay()`: Waits to allow stable connection establishment.
- `rcl_get_default_allocator()`: Initializes memory allocation for Micro-ROS operations.
- `rcl_publisher_init_default()`: Creates a publisher for Int32 messages on the topic "micro_ros_pub".
- `rcl_timer_init_default()`: Creates a timer that executes `timer_callback()` every 1000 milliseconds. `RCL_MS_TO_NS()` converts milliseconds to nanoseconds.
- `rcl_executor_init()`: Initializes the Micro-ROS Executor, which manages tasks and callbacks.
- `rcl_executor_add_timer()`: Registers the timer with the executor so that it can execute tasks non-blocking.



Micro-ros Publisher



```
void loop() {  
    //Executor Spin  
    delay(100);  
    RCSOFTCHECK(rclc_executor_spin_some(&executor,  
RCL_MS_TO_NS(100)));  
}
```

Loop Section

- This function checks for new data/timer timeouts/triggers and executes pending callbacks (e.g., timers, subscriptions, services).
- Does not block execution, meaning it allows other tasks to continue running.
- The argument RCL_MS_TO_NS(100) specifies the timeout in nanoseconds (100ms).

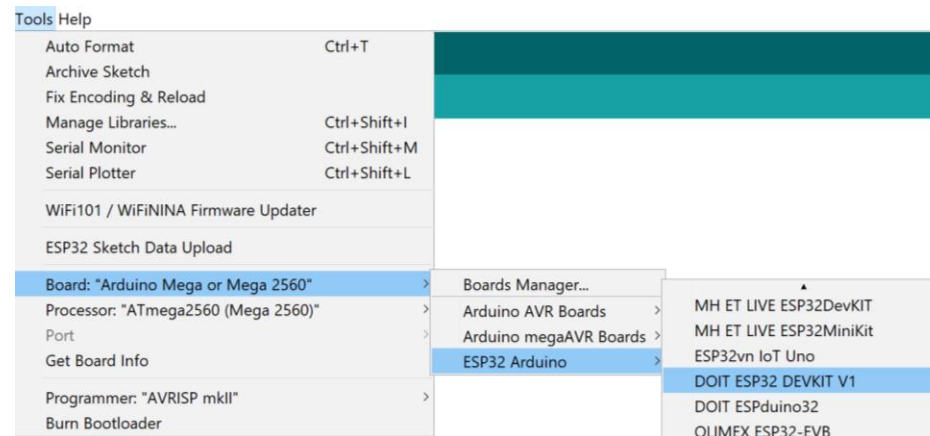


Activity



Compilation (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 (for Hackerboard is the same)
 - For ESP32 select ESP32 Arduino > DOIT ESP32 DEVKIT V1



- Compile the code using by clicking check mark button located on the upper left corner.



- The following message should be displayed:

Done compiling.

Sketch uses 9424 bytes (3%) of program storage space. Maximum is 253952 bytes.
Global variables use 1826 bytes (22%) of dynamic memory, leaving 6366 bytes for local variables. Maximum is 8192 bytes.

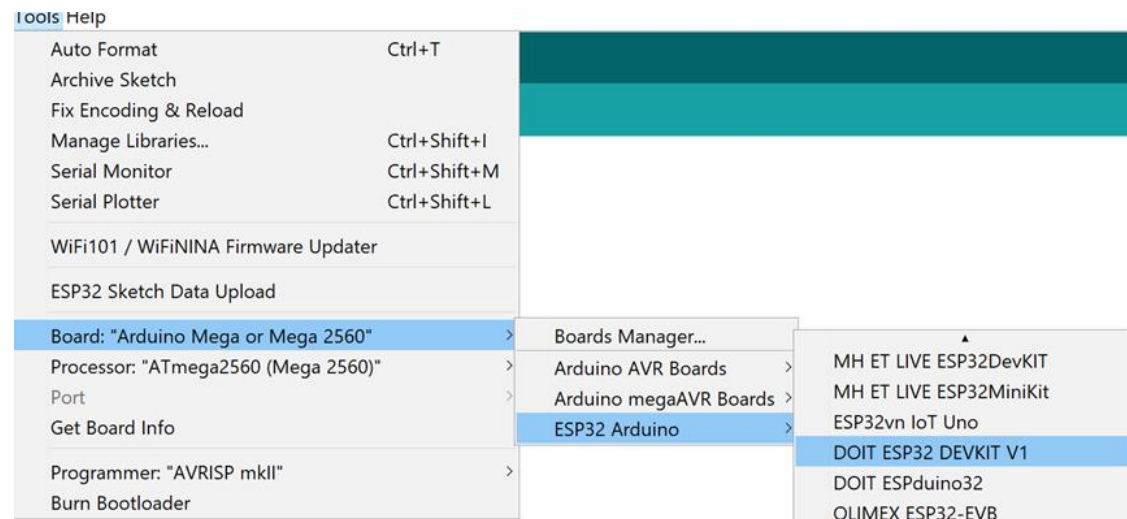
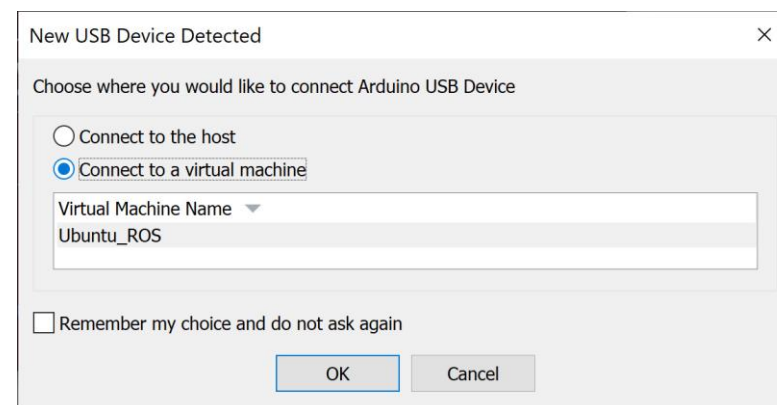


Activity



Uploading (Arduino IDE)

- Connect the Hackerboard or the ESP32 board
- Select the port to be used Tools>Port
 - If working on the VM, you must first select the option Connect to a virtual machine when automatically prompted (shown) and then select the port.
 - If in WSL, follow the steps on the presentation: "MCR2_Micro_ROS_Installation".
- Select the board to be used Tools -> ESP32 Arduino > DOIT ESP32 DEVKIT V1





Activity



Uploading (Arduino IDE)

- Upload the code using the arrow on the top left corner of the IDE.



- The following message should appear on the IDE

```
Done uploading.  
Sketch uses 1488 bytes (4%) of program storage space.  
Global variables use 198 bytes (9%) of dynamic memory
```

Running the node (Computer)

- Connect the board to the computer with ROS.
- (In Ubuntu) Make sure the port permissions are granted for the user (Skip this step if already performed).
 - In a new terminal type `cd /dev` to visualise the port designated by Ubuntu to the MCU. This port are usually called `/ttyACM0` or `/ttyUSB0`.

```
sudo chmod 666 /dev/ttyACM*  
sudo chmod 666 /dev/ttyUSB*
```



Activity



Test (Computer)

1. Open a terminal and type the following.

```
$ ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/tty***0
```

2. Reset the ESP32 (pressing the reset button) to reconnect to the computer (agent waiting timeout).

```
mario@MarioPC:~/uros_ws$ ros2 run micro_ros_agent mi
cro_ros_agent serial --dev /dev/ttyUSB0
[1737636692.137695] info | TermiosAgentLinux.cpp
| init | running...
| fd: 3
[1737636692.138467] info | Root.cpp |
set_verbose_level | logger setup |
verbose_level: 4
[1737636698.665805] info | Root.cpp |
create_client | create |
client_key: 0x0C9424D2, session_id: 0x81
[1737636698.666090] info | SessionManager.hpp |
establish_session | session established |
client_key: 0x0C9424D2, address: 0
[1737636698.702311] info | ProxyClient.cpp |
create_participant | participant created |
client_key: 0x0C9424D2, participant_id: 0x000(1)
[1737636698.721583] info | ProxyClient.cpp |
create_topic | topic created |
client_key: 0x0C9424D2, topic_id: 0x000(2), partici
ant_id: 0x000(1)
[1737636698.733573] info | ProxyClient.cpp |
create_publisher | publisher created |
client_key: 0x0C9424D2, publisher_id: 0x000(3), part
```

- Open another terminal and type the following.

```
$ ros2 topic list
```

```
mario@MarioPC:~$ ros2 topic list
/micro_ros_counter
/parameter_events
/rosout
```

- Echo the topic
“/micro_ros_arduino_node_publisher”

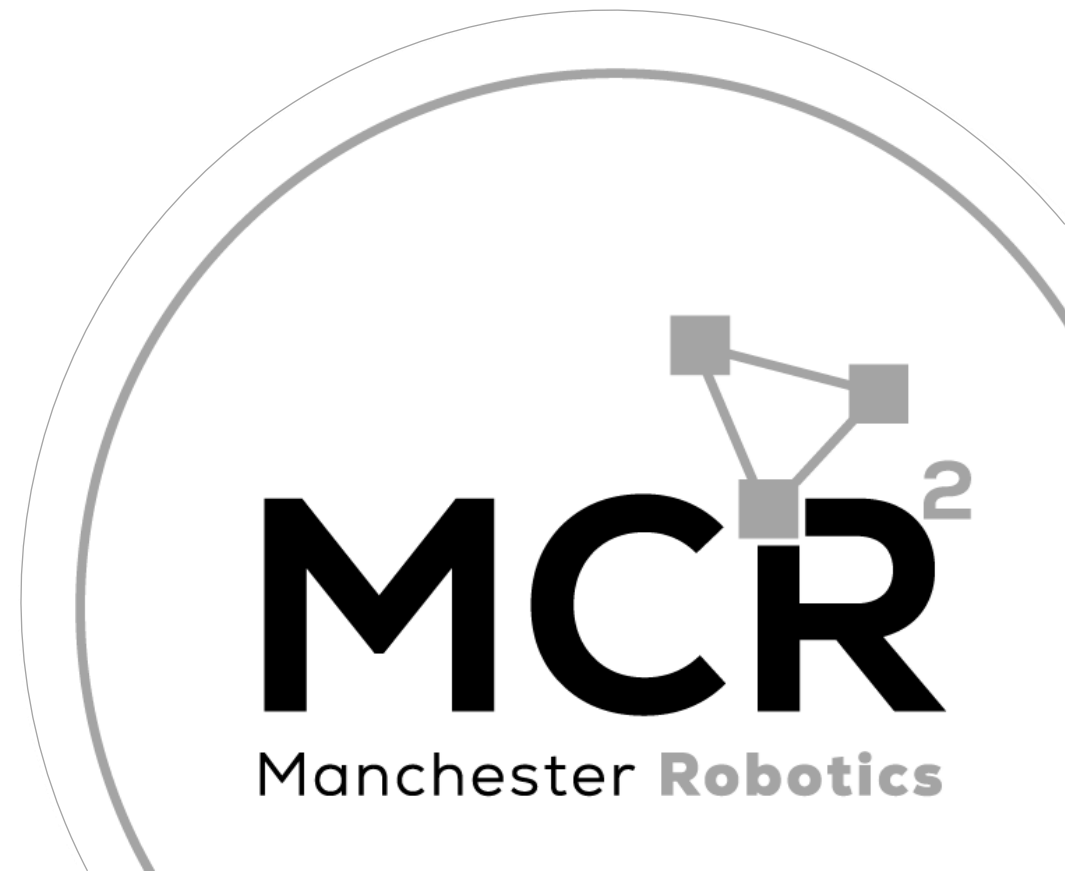
```
$ ros2 topic echo /micro_ros_arduino_node_publisher
```

```
mario@MarioPC:~$ ros2 topic echo /micro_ros_counter
data: 24
----
data: 25
----
data: 26
----
data: 27
----
data: 28
----
data: 29
----
```

Micro-ROS Serial Communication

*Activity 1.2: Publisher
w/reconnection*

{Learn, Create, Innovate};





Activity 1.2



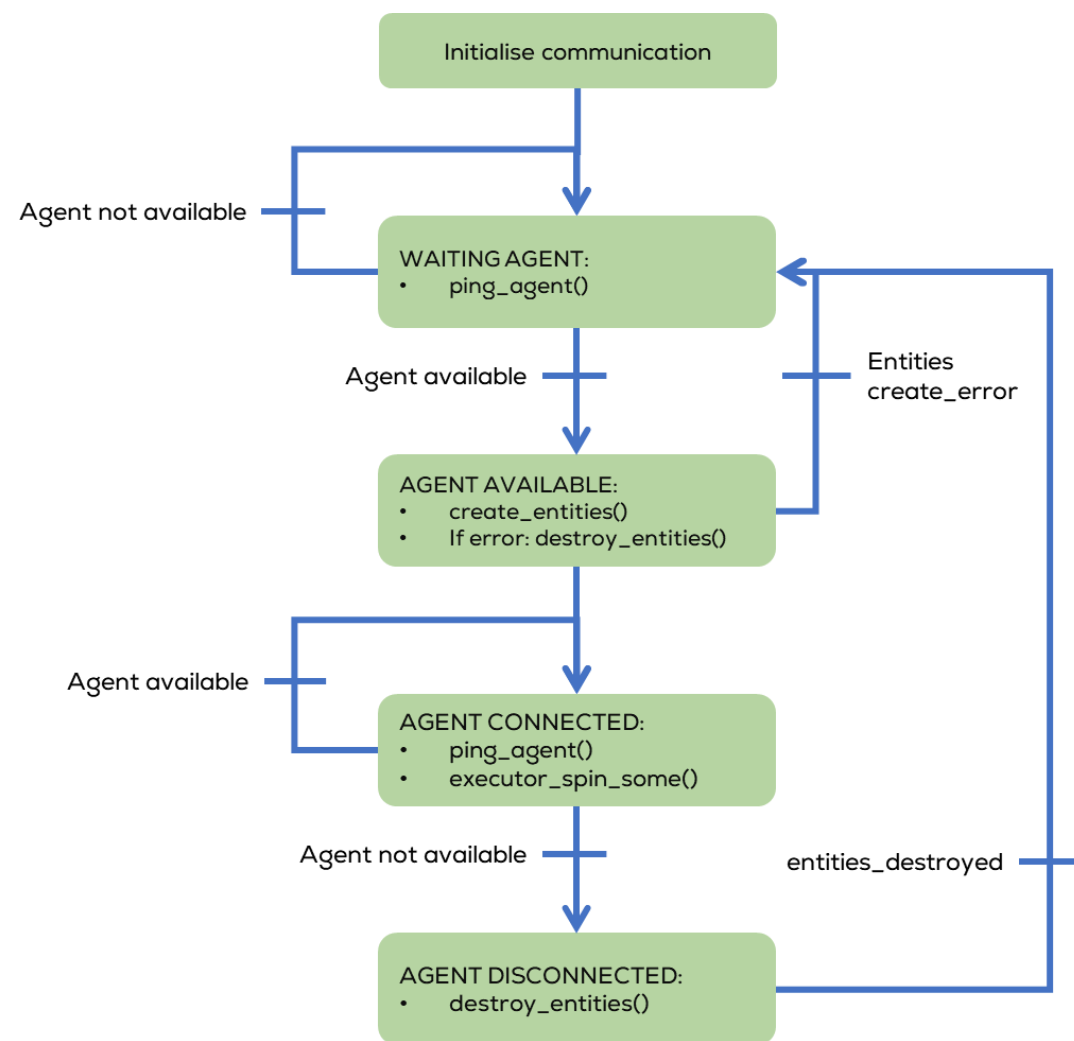
Introduction

- In the previous activity a simple publisher was declared.
- When initialising the communication with the agent a `delay()` function was used to wait for the agent to be available
- If no agent was available during that time, the user had to restart the microcontroller **after** the agent had been started on the host computer.
- Usually a restart from the microcontroller, is not recommended when working with robots.
- A reconnection function is required.

```
void setup() {  
  // Initializes communication between ESP32 and the ROS 2 agent (Serial).  
  set_microros_transports();  
  ...  
  
  //Connection delay (waiting for agent to be available)  
  delay(2000);  
  ...  
}
```

Reconnection state machine

- The reconnection function is made using a state machine.
- The machine, uses the ping function to verify if an agent is running on the host computer (waiting for a connection).
- If an agent is detected, the program initialises the “entities” required by micro-ros to communicate i.e., supports, executors, and handlers.
- If an error occurs during the creation, the program destroys the created entities.
- If there is no error, the executor starts spinning and pinging the host to verify if the agent is still available.
- If the agent gets disconnected, the entities get destroyed, and the agent continues waiting for the agent to get connected again.





Activity 1.2



Instructions (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 or Arduino Due
 - For ESP32 select ESP32 Arduino > DOIT ESP32 DEVKIT V1
- Open the previous Activity and modify it according to the following slides. Modifications are highlighted in Black.
- For this activity, no LED on Pin 22 is required

A screenshot of the Arduino IDE 2.1.1 interface. The main window displays the code for 'nlights_1Way_arduino.ino'. The code is written in C++ and includes comments and preprocessor directives. The code is as follows:

```
1 // Traffic Light Basic Code
2 /* Kat Nels, Manchester Robotics, Feb 19 2022
3 // adapted from Arduino Neopixel Library "RGBWstrandtest"
4 // "Nlight_oneway" = choose between one or two TLBs, only one-way traffic
5 // one MCU board (MCUB) and up to two traffic lights (TLB)
6 // next level of complexity up from "lint_1way" script
7 // should initialize TLB1 to green and, if TLB2, TLB2 initialized to red
8 // then change green > yellow > red > yellow > green according to hard coded timers
9 */
10
11 /* SETUP */
12 #define LED_PIN 52 // Define the data out pin on the ESP32
13 #define greenTimer 2 //how long each color lasts, units are sec
14 #define redTimer 2
15 #define yellowTimer 1
16 /* END SETUP */
17
18
```

The IDE interface shows the 'Tools' menu with 'Board' selected, and the 'Output' window at the bottom. The status bar at the bottom indicates 'Ln 195, Col 36' and 'Arduino Mega or Mega 2560 [not connected]'.



Activity 1.2



```
// Include Libraries to be used
#include <micro_ros_arduino.h> //micro-ros-arduino library
#include <rcl/rcl.h> //Core ROS 2 Client Library (RCL) for node
management.
#include <rcl/error_handling.h> //Error handling utilities for Micro-ROS.
#include <rclcpp/rclcpp.h> //Micro-ROS Client library for embedded
devices.
#include <rclcpp/executor.h> //Micro-ROS Executor to manage callbacks
#include <std_msgs/msg/int32.h> //Predefined ROS 2 message type
#include <stdio.h> //Standard I/O library for debugging.

//Declare nodes to be used
rcl_node_t node; //Represents a ROS 2 Node running on the
microcontroller.

//Instantiate executor and its support classes
rclcpp_executor_t executor; //Manages task execution (timers, callbacks, etc.).
rclcpp_support_t support; //Handles initialization & communication setup.
rcl_allocator_t allocator; //Manages memory allocation.

//Declare Publishers to be used
rcl_publisher_t publisher; //Declares a ROS 2 publisher for sending messages.

//Declare timers to be used
rcl_timer_t timer; //Creates a timer to execute functions at
intervals.

//Declare Messages to be used
std_msgs__msg__Int32 msg; //Defines a message of type int32.
```

```
//Define Macros to be used
//Executes fn and returns false if it fails.
#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){return
false;}}

// Executes fn, but ignores failures.
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}

// Executes a given statement (X) periodically every MS milliseconds
#define EXECUTE_EVERY_N_MS(MS, X) do { \
    static volatile int64_t init = -1; \
    if (init == -1) { init = uxr_millis();} \
    if (uxr_millis() - init > MS) { X; init = uxr_millis();} \
} while (0)\

//Defines State Machine States
enum states {
    WAITING_AGENT,
    AGENT_AVAILABLE,
    AGENT_CONNECTED,
    AGENT_DISCONNECTED
} state;

//Define callbacks
void timer_callback(rcl_timer_t * timer, int64_t last_call_time)
{
    RCLC_UNUSED(last_call_time);
    if (timer != NULL) {
        RCSOFTCHECK(rcl_publish(&publisher, &msg, NULL));
        msg.data++;
    }
}
```



Activity 1.2



```
bool create_entities()
{
    //Initializes memory allocation for Micro-ROS operations.
    allocator = rcl_get_default_allocator();

    //Creates a ROS 2 support structure to manage the execution context.
    RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

    // create node
    RCCHECK(rclc_node_init_default(&node, "micro_ros_pub_node", "", &support));

    // create publisher
    RCCHECK(rclc_publisher_init_default(
        &publisher,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
        "micro_ros_counter"));

    // create timer,
    const unsigned int timer_timeout = 1000;
    RCCHECK(rclc_timer_init_default(
        &timer,
        &support,
        RCL_MS_TO_NS(timer_timeout),
        timer_callback));

    // create zero initialised executor (no configured) to avoid memory problems
    executor = rclc_executor_get_zero_initialized_executor();
    // Initializes the Micro-ROS Executor, which manages tasks and callbacks.
    RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
    // Register timer with executor
    RCCHECK(rclc_executor_add_timer(&executor, &timer));

    return true;
}
```

```
void destroy_entities()
{
    rmw_context_t * rmw_context = rcl_context_get_rmw_context(&support.context);
    (void) rmw_uros_set_context_entity_destroy_session_timeout(rmw_context, 0);

    rcl_publisher_fini(&publisher, &node);
    rcl_timer_fini(&timer);
    rclc_executor_fini(&executor);
    rcl_node_fini(&node);
    rclc_support_fini(&support);
}

//Setup
void setup() {
    // Initializes communication between ESP32 and the ROS 2 agent (Serial).
    set_microros_transports();

    //Initial State
    state = WAITING_AGENT;

    // Initialise message
    msg.data = 0;
}
```



Activity 1.2



```
void loop() {
  switch (state) {

    case WAITING_AGENT:
      EXECUTE_EVERY_N_MS(500, state = (RMW_RET_OK == rmw_uros_ping_agent(100, 1)) ? AGENT_AVAILABLE : WAITING_AGENT;);
      break;

    case AGENT_AVAILABLE:
      state = (true == create_entities()) ? AGENT_CONNECTED : WAITING_AGENT;
      if (state == WAITING_AGENT) {
        destroy_entities();
      };
      break;

    case AGENT_CONNECTED:
      EXECUTE_EVERY_N_MS(200, state = (RMW_RET_OK == rmw_uros_ping_agent(100, 1)) ? AGENT_CONNECTED : AGENT_DISCONNECTED;);
      if (state == AGENT_CONNECTED) {
        rcl_executor_spin_some(&executor, RCL_MS_TO_NS(100));
      }
      break;

    case AGENT_DISCONNECTED:
      destroy_entities();
      state = WAITING_AGENT;
      break;

    default:
      break;
  }
}
```

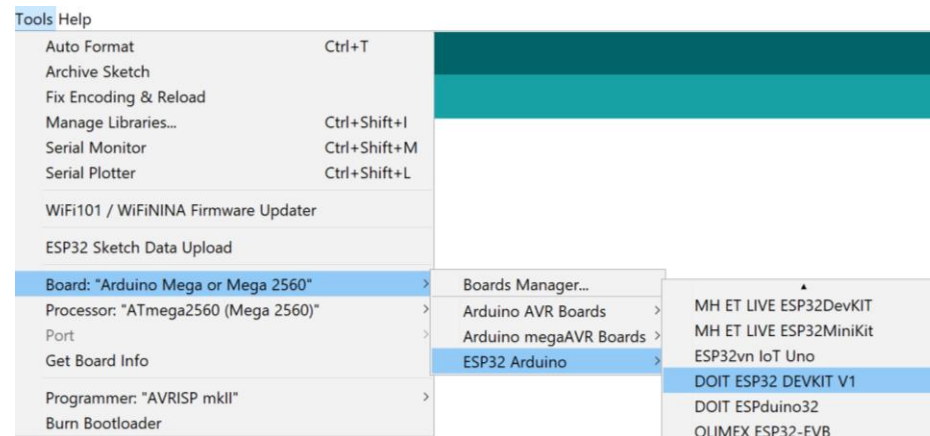


Activity



Compilation (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 (for Hackerboard is the same)
 - For ESP32 select ESP32 Arduino > DOIT ESP32 DEVKIT V1



- Compile the code using by clicking check mark button located on the upper left corner.



- The following message should be displayed:

Done compiling.

Sketch uses 9424 bytes (3%) of program storage space. Maximum is 253952 bytes.
Global variables use 1826 bytes (22%) of dynamic memory, leaving 6366 bytes for local variables. Maximum is 8192 bytes.

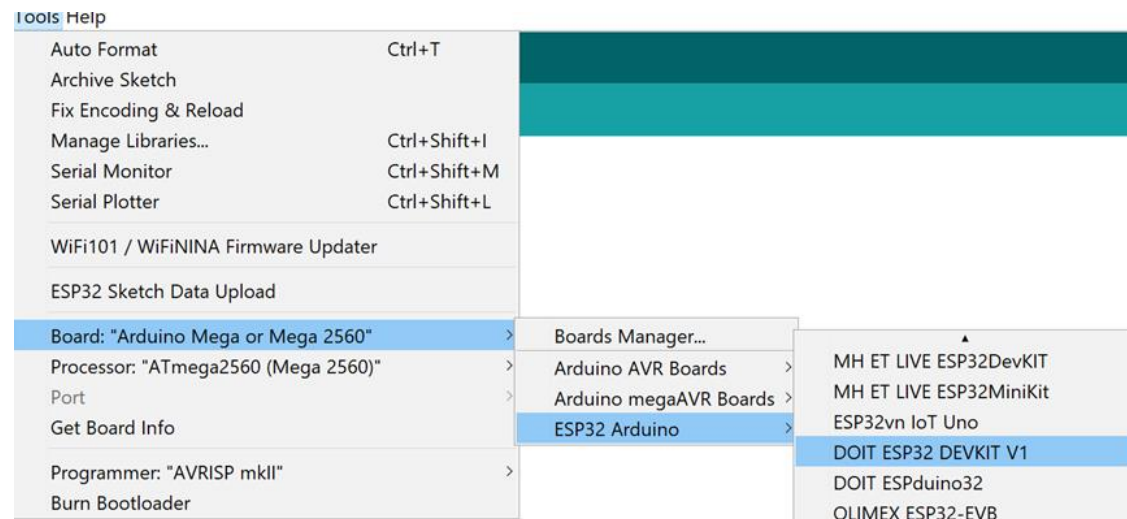
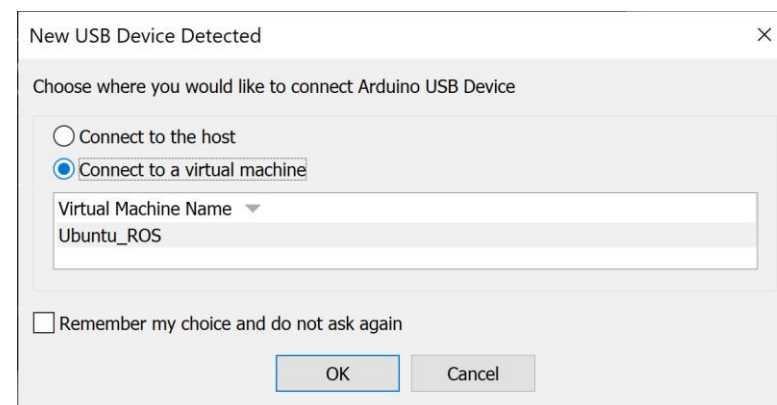


Activity



Uploading (Arduino IDE)

- Connect the Hackerboard or the ESP32 board
- Select the port to be used Tools>Port
 - If working on the VM, you must first select the option Connect to a virtual machine when automatically prompted (shown) and then select the port.
 - If in WSL, follow the steps on the presentation: "MCR2_Micro_ROS_Installation".
- Select the board to be used Tools -> ESP32 Arduino > DOIT ESP32 DEVKIT V1





Activity



Uploading (Arduino IDE)

- Upload the code using the arrow on the top left corner of the IDE.



- The following message should appear on the IDE

```
Done uploading.  
Sketch uses 1488 bytes (4%) of program storage space.  
Global variables use 198 bytes (9%) of dynamic memory
```

Running the node (Computer)

- Connect the board to the computer with ROS.
- (In Ubuntu) Make sure the port permissions are granted for the user (Skip this step if already performed).
 - In a new terminal type `cd /dev` to visualise the port designated by Ubuntu to the MCU. This port are usually called `/ttyACM0` or `/ttyUSB0`.

```
sudo chmod 666 /dev/ttyACM*  
sudo chmod 666 /dev/ttyUSB*
```



Activity 1.2



Test (Computer)

1. Open a terminal and type the following.

```
$ ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/tty***0
```

```
mario@MarioPC:~/uros_ws$ ros2 run micro_ros_agent mi
cro_ros_agent serial --dev /dev/ttyUSB0
[1737636692.137695] info      | TermiosAgentLinux.cpp
| init                      | running...
| fd: 3
[1737636692.138467] info      | Root.cpp
set_verbose_level          | logger setup
verbose_level: 4
[1737636698.665805] info      | Root.cpp
create_client              | create
client_key: 0x0C9424D2, session_id: 0x81
[1737636698.666090] info      | SessionManager.hpp
establish_session          | session established
client_key: 0x0C9424D2, address: 0
[1737636698.702311] info      | ProxyClient.cpp
create_participant         | participant created
client_key: 0x0C9424D2, participant_id: 0x000(1)
[1737636698.721583] info      | ProxyClient.cpp
create_topic               | topic created
client_key: 0x0C9424D2, topic_id: 0x000(2), particip
ant_id: 0x000(1)
[1737636698.733573] info      | ProxyClient.cpp
create_publisher           | publisher created
client_key: 0x0C9424D2, publisher_id: 0x000(3), part
```

- Open another terminal and type the following.

```
$ ros2 topic list
```

```
mario@MarioPC:~$ ros2 topic list
/micro_ros_counter
/parameter_events
/rosout
```

- Echo the topic

“/micro_ros_arduino_node_publisher”

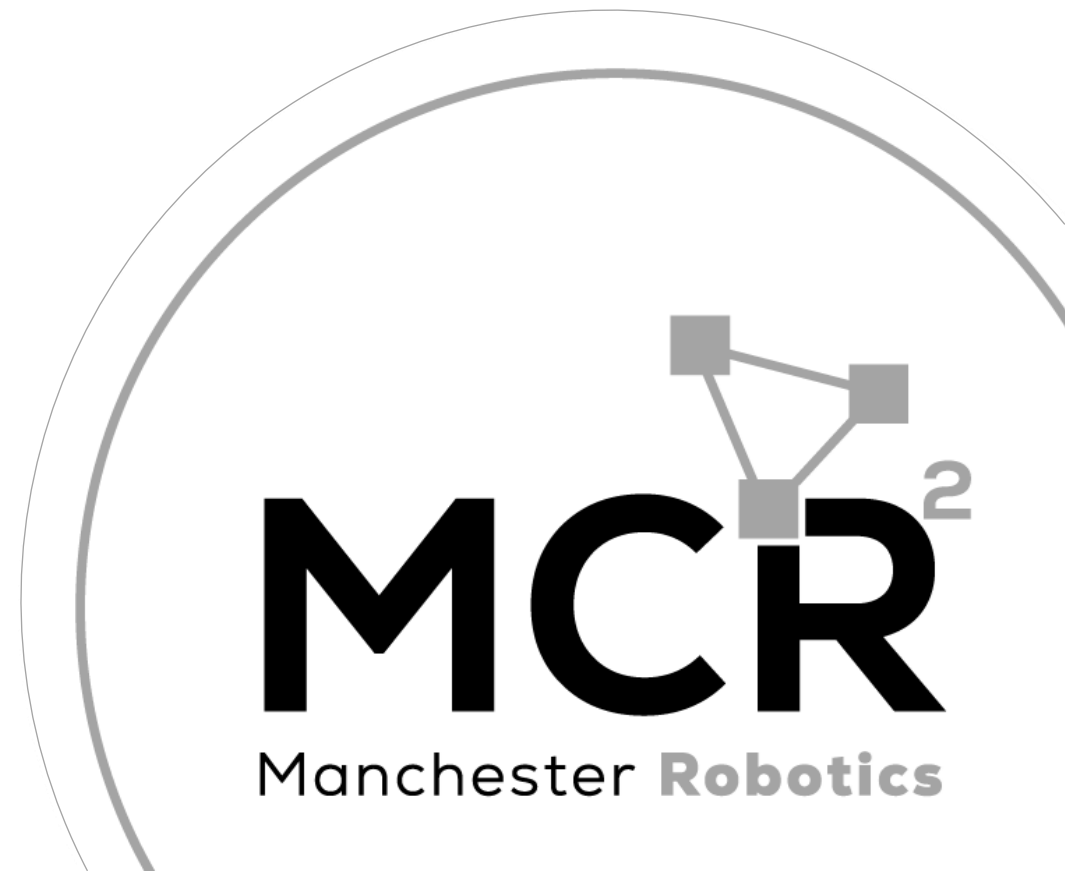
```
$ ros2 topic echo /micro_ros_arduino_node_publisher
```

```
mario@MarioPC:~$ ros2 topic echo /micro_ros_counter
data: 24
----
data: 25
----
data: 26
----
data: 27
----
data: 28
----
data: 29
----
```

Micro-ROS Serial Communication

Activity 2: Subscriber

{Learn, Create, Innovate};





Requirements



- The following activity is based on the example tutorial found in the provided micro-ROS libraries.
- This activity requires Arduino IDE to be installed and configured as shown in “MCR2_Micro_ROS_Installation”.
- Requirements:
 - Microcontroller
 - Computer
 - micro-usb to USB cable (Data)
 - Breadboard
 - LED
 - 2 x 150 Ohm Resistor

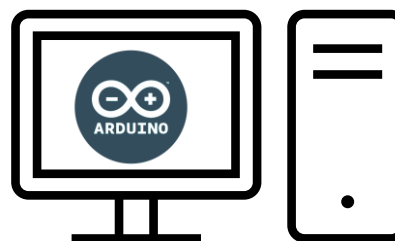


Or

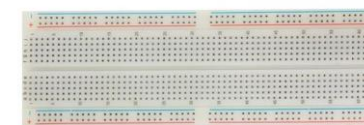


Hackerboard

ESP32 board



Computer



Breadboard



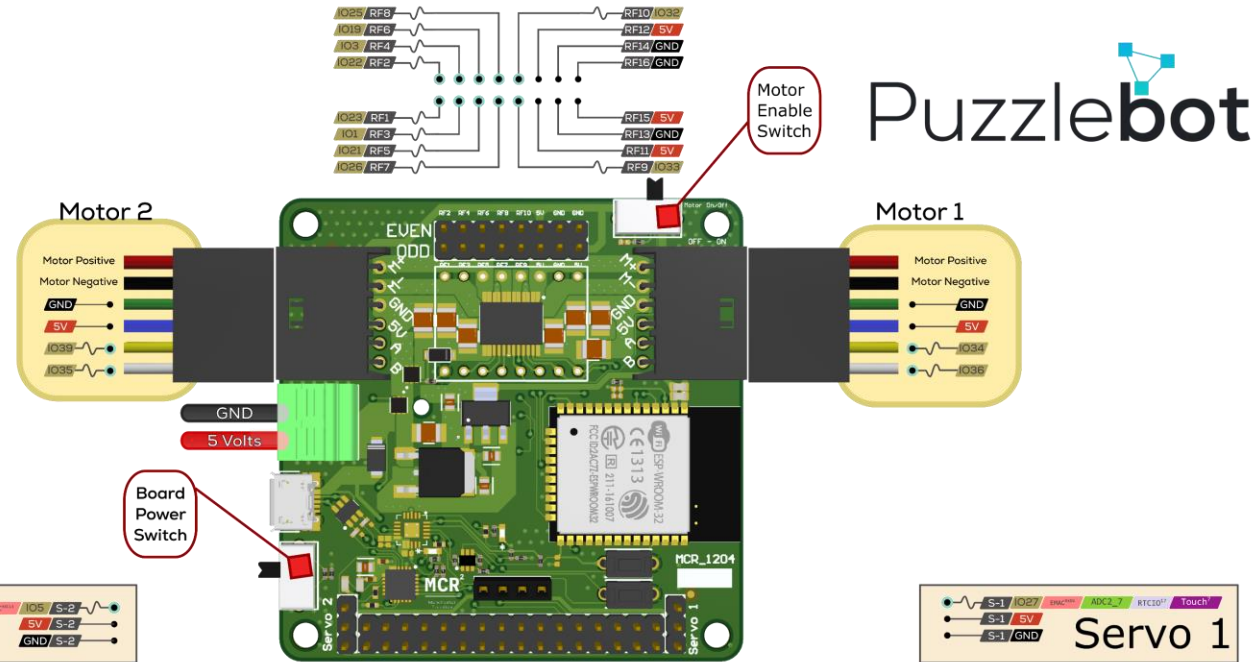
LED



150 Ω Resistor



Hackerboard Pinout



- 5V comes from LDO regulated supply, max current draw is 800mA
- Absolute maximum current draw per GPIO pin is 40mA
- Input voltage is 5-13.5V via terminal block
- Current draw via microUSB socket is 1.8A max

- Puzzlebot Devices
- Power
- GND
- Serial Pin
- Analogue Pin
- Control Pin
- Physical Pin Number
- GPIO Pin
- Touch Pin
- DAC Pin
- Port Pin
- PWM Pin

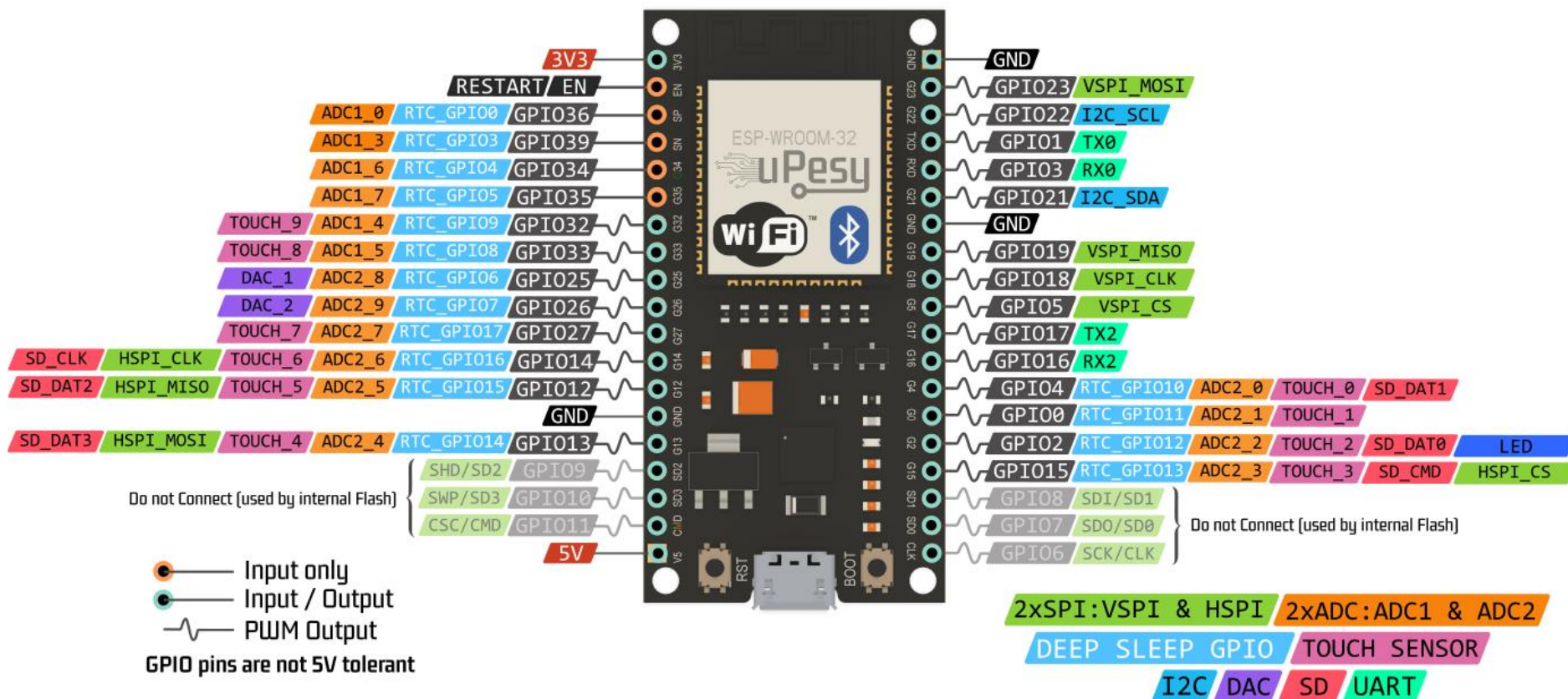
- Notes:**
- These pins are **input only**. They cannot output signals
 - These pins are integrated to the Microcontroller's flash memory, and are only present for debugging purposes. They should not be used for general purpose
 - Outputs PWM signal at boot time
 - Boot fails if pulled high
 - Boot fails if pulled low

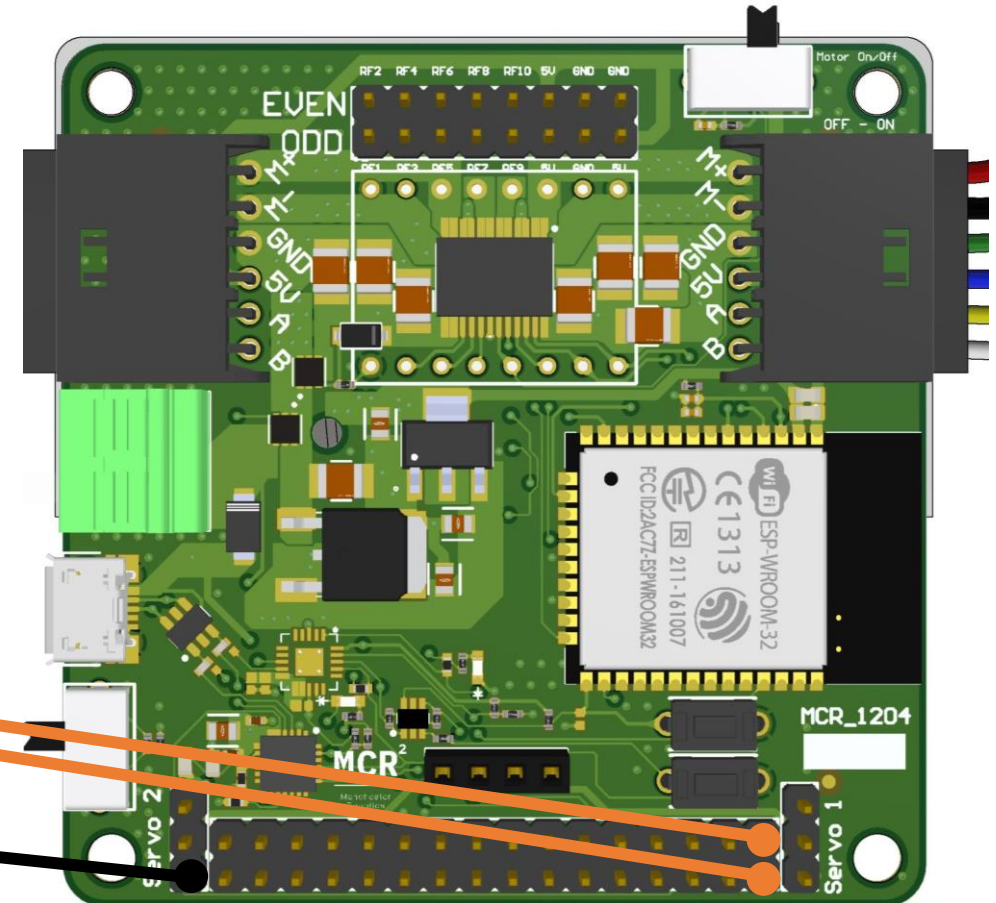
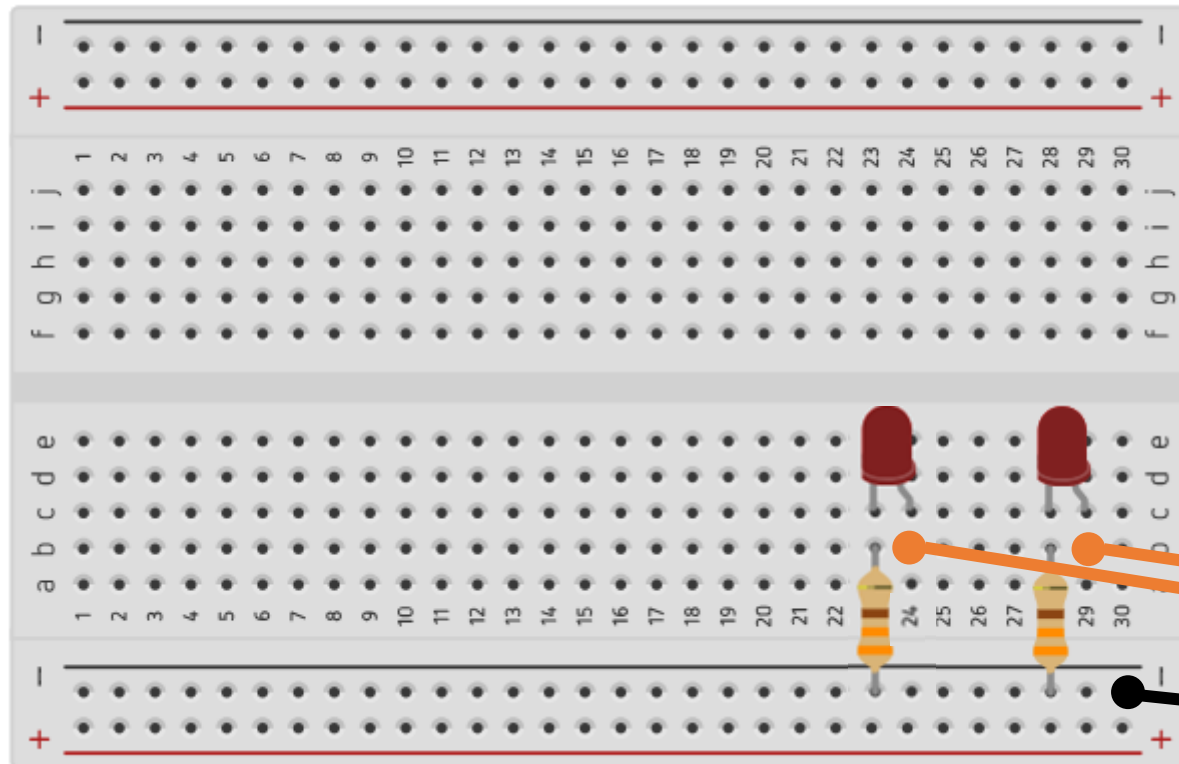
<https://www.manchester-robotics.com>

01 FEB 2022
v1

Derivative of "Puzzlebot Template" by AthlonOne, used under CC BY SA.
MCR_1204 Pinout is licensed under CC BY SA by MCR

ESP32 Wroom DevKit Full Pinout





GND

Pin 22

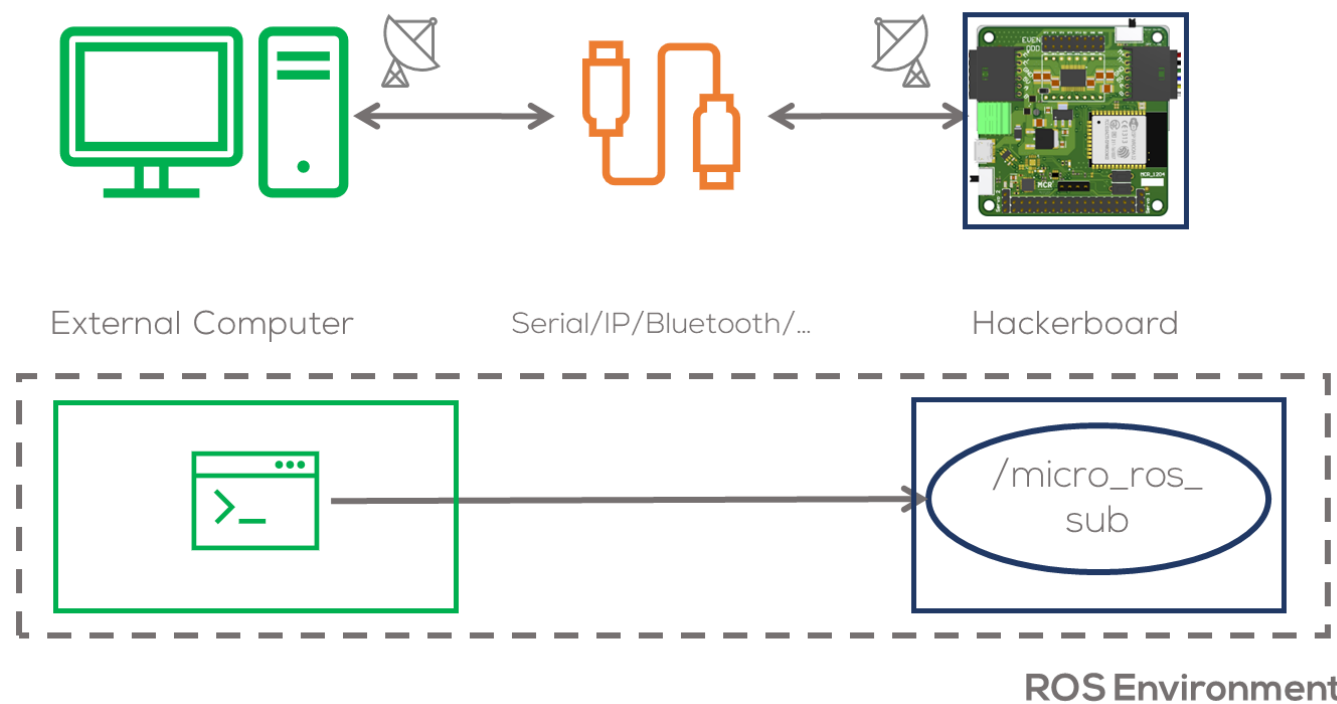
Pin 23



Description



- In this activity, a node running a simple subscriber will be made.
- This node will run inside the microcontroller and will communicate with the computer via UART.
- The node will subscribe to a Float32 message.
- This activity will be divided into two parts. The first part involves the Arduino IDE to program the MCU.
- The second part involves the commands required to connect to the board to the computer.



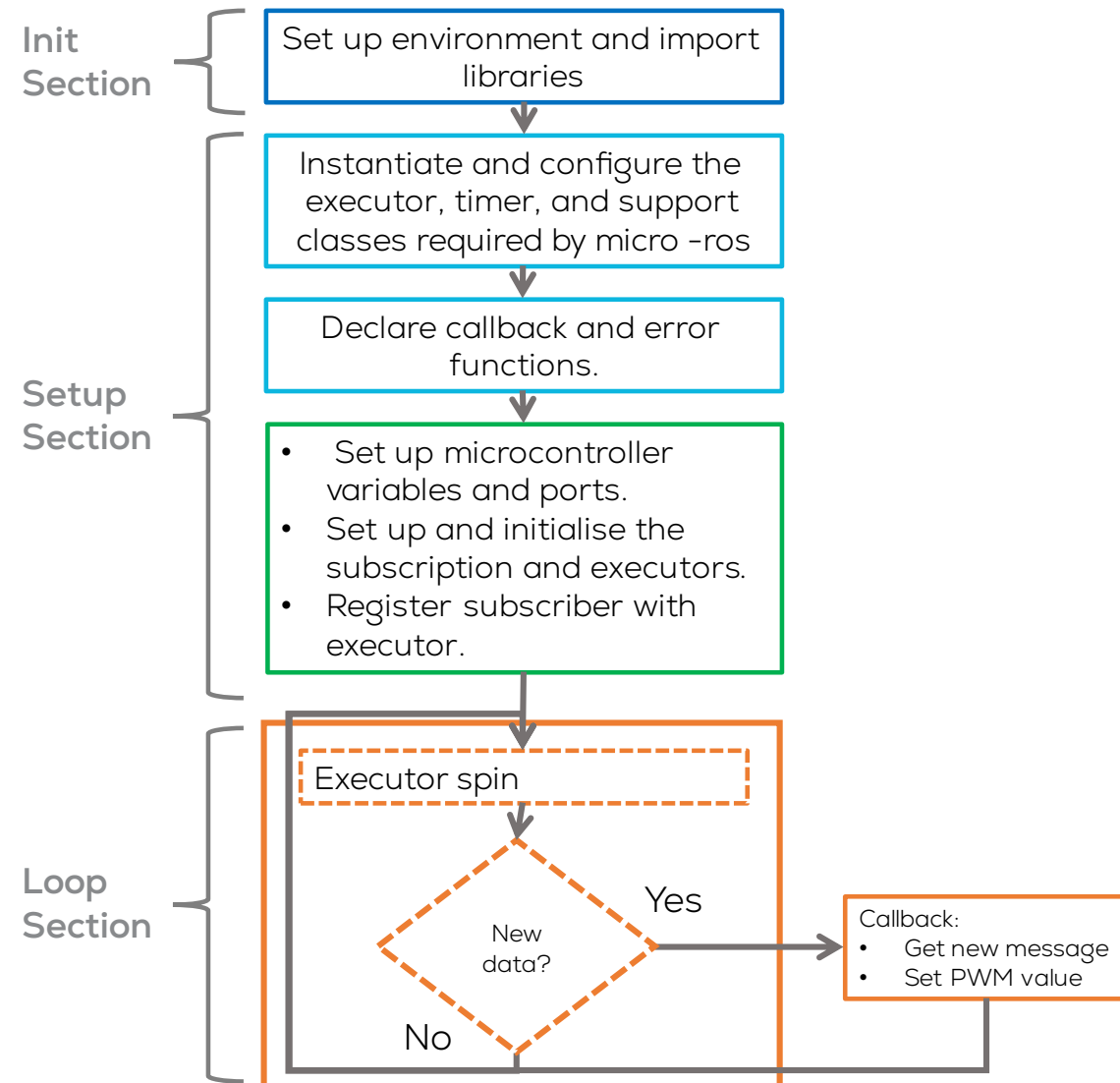


Description



Objective

- The objective of this activity is for the microcontroller to control the brightness of an LED from the computer using ROS2.
- To this end, the microcontroller must set a node called “micro_ros_sub_node” and subscribe to a topic “micro_ros_sub”.
- The computer will send a value in the range [0,1].
- The microcontroller must set a PWM Value from [0%,100%] to dim the LED.





Subscriber set up



Configuring a Subscriber in Micro-ROS

What is a Subscriber?

- A subscriber listens to a ROS 2 topic and processes incoming messages.
- In Micro-ROS, subscribers receive messages asynchronously using a callback function.

1. Define the Message Type and a subscriber object.
2. Create the Subscriber Callback
3. Initialize/configure the Subscriber
 - ON_NEW_DATA: Executes the callback only when new data arrives.
 - ALWAYS: Executes the callback on every executor cycle, even if no new data is available.
4. Add the Subscriber to the Executor
5. Process the Executor in loop()

```
#include <std_msgs/msg/int32.h> //Predefined ROS 2 message type
//Declare Messages to be used
std_msgs__msg__Int32 msg; //Defines a message of type int32.
rcl_subscription_t subscriber;
...
void subscription_callback(const void * msg_in) {
    const std_msgs__msg__Int32 * msg = (const std_msgs__msg__Int32 *)msg_in;
    Serial.print("Received: ");
    Serial.println(msg->data);
}

void setup() {
    // create subscriber
    rcl_subscription_init_default(
        &subscriber,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
        "micro_ros_subscriber_topic");
    // create executor
    RCCHECK(rcl_executor_init(&executor, &support.context, 1, &allocator));
    // Register suscription with executor
    RCCHECK(rcl_executor_add_subscription(&executor, &subscriber, &msg,
    &subscription_callback, ON_NEW_DATA));
}

void loop() {
    //Executor Spin
    delay(100);
    RCCHECK(rcl_executor_spin_some(&executor, RCL_MS_TO_NS(100)));
}
```



Activity 2



Instructions (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board
ESP32 or Arduino Due
 - For ESP32 select ESP32 Arduino > DOIT
ESP32 DEVKIT V1
- For this activity, no LED on Pin 22 is required

A screenshot of the Arduino IDE 2.1.1 interface. The main window displays the 'nlights_1Way_arduino.ino' file. The code is written in C++ and includes comments in English. The code defines a traffic light system with three colors: green, yellow, and red. The setup function initializes the LED pin and the timers. The main loop function controls the traffic light sequence. The IDE interface shows the 'Tools' menu with 'Board' selected, and the 'Board' dropdown menu is open, showing 'Arduino Mega or Mega 2560' selected. The status bar at the bottom indicates 'Ln 195, Col 36' and 'Arduino Mega or Mega 2560 [not connected]'.



Activity 2



```
// Include Libraries to be used
#include <micro_ros_arduino.h> //micro-ros-arduino library
#include <rcl/rcl.h> //Core ROS 2 Client Library (RCL) for node
management.
#include <rcl/error_handling.h> //Error handling utilities for Micro-ROS.
#include <rclcpp/rclcpp.h> //Micro-ROS Client library for embedded
devices.
#include <rclcpp/executor.h> //Micro-ROS Executor to manage callbacks
#include <std_msgs/msg/float32.h> //Predefined ROS 2 message type
#include <stdio.h> //Standard I/O library for debugging.

//Declare nodes to be used
rcl_node_t node; //Represents a ROS 2 Node running on the
microcontroller.

//Instantiate executor and its support classes
rclcpp_executor_t executor; //Manages task execution (timers, callbacks, etc.).
rclcpp_support_t support; //Data structure that holds the execution context
of Micro-ROS, including its communication state, memory management, and
initialization data.
rcl_allocator_t allocator; //Manages memory allocation.

//Declare Subscribers to be used
rcl_subscription_t subscriber;

//Declare timers to be used
rcl_timer_t timer; //Creates a timer to execute functions at
intervals.

//Declare Messages to be used
std_msgs__msg__Float32 msg; //Defines a message of type float32.
```

```
//Define Macros to be used
//Executes fn and goes to error_loop() function if fn fails.
#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc !=
RCL_RET_OK)){error_loop();}}
// Executes fn, but ignores failures.
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}

//Specifies GPIO pin 13 for controlling an LED
#define LED_PIN 23 //Define LED_PIN
#define PWM_PIN 22 //DEFINE PWM_PIN
#define PWM_FRQ 5000 //Define PWM Frequency
#define PWM_RES 8 //Define PWM Resolution
#define PWM_CHNL 0 //Define Channel
#define MSG_MIN_VAL 0 //Define min input value
#define MSG_MAX_VAL 1 //Define max input value
//Variables to be used
float pwm_set_point = 0.0;

//Define Error Functions
void error_loop(){
    while(1){
        digitalWrite(LED_PIN, !digitalRead(LED_PIN)); // Toggle LED state
        delay(100); // Wait 100 milliseconds
    }
}

//Define callbacks
void subscription_callback(const void * msgin)
{
    //Get the message received and store it on the message msg
    const std_msgs__msg__Float32 * msg = (const std_msgs__msg__Float32 *)msgin;
    pwm_set_point = constrain(msg->data, MSG_MIN_VAL, MSG_MAX_VAL);
    digitalWrite(LED_PIN, (uint32_t) (pow(2, PWM_RES) * (pwm_set_point / 1.0)));
}
```



Activity 2



```
//Setup
void setup() {
    // Initializes communication between ESP32 and the ROS 2 agent (Serial).
    set_microros_transports();
    //Setup Microcontroller Pins
    pinMode(LED_PIN, OUTPUT);
    pinMode(PWM_PIN, OUTPUT);
    digitalWrite(LED_PIN, HIGH);
    ledcSetup(PWM_CHNL, PWM_FRQ, PWM_RES); //Setup the PWM
    ledcAttachPin(PWM_PIN, PWM_CHNL);      //Setup Attach the Pin to the Channel
    //Connection delay
    delay(2000);
    //Initializes memory allocation for Micro-ROS operations.
    allocator = rcl_get_default_allocator();

    //Creates a ROS 2 support structure to manage the execution context.
    RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

    // create node
    RCCHECK(rclc_node_init_default(&node, "micro_ros_sub_node", "", &support));

    // create subscriber
    RCCHECK(rclc_subscription_init_default(
        &subscriber,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
        "micro_ros_sub"));

    // create executor
    RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
    // Register suscription with executor
    RCCHECK(rclc_executor_add_subscription(&executor, &subscriber, &msg,
    &subscription_callback, ON_NEW_DATA));
}
```

```
void loop() {
    //Executor Spin
    delay(100);
    RCCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100)));
}
```

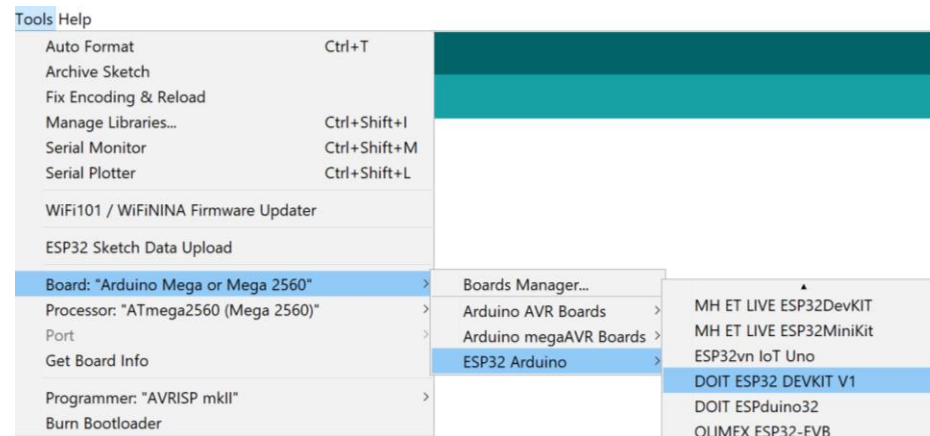


Activity 2



Compilation (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 (for Hackerboard is the same)
 - For ESP32 select ESP32 Arduino > DOIT ESP32 DEVKIT V1



- Compile the code using by clicking check mark button located on the upper left corner.



- The following message should be displayed:

Done compiling.

Sketch uses 9424 bytes (3%) of program storage space. Maximum is 253952 bytes.
Global variables use 1826 bytes (22%) of dynamic memory, leaving 6366 bytes for local variables. Maximum is 8192 bytes.

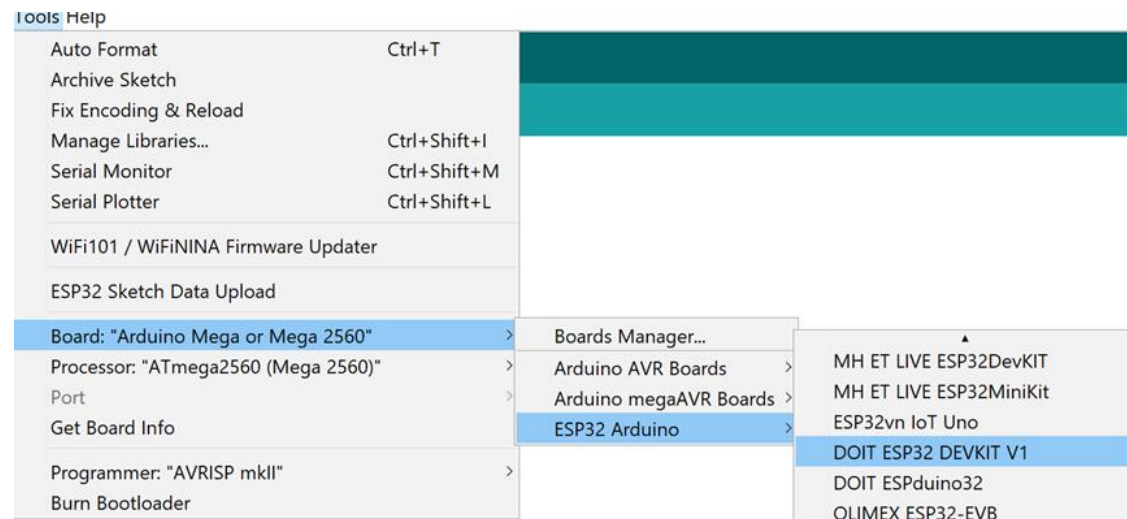
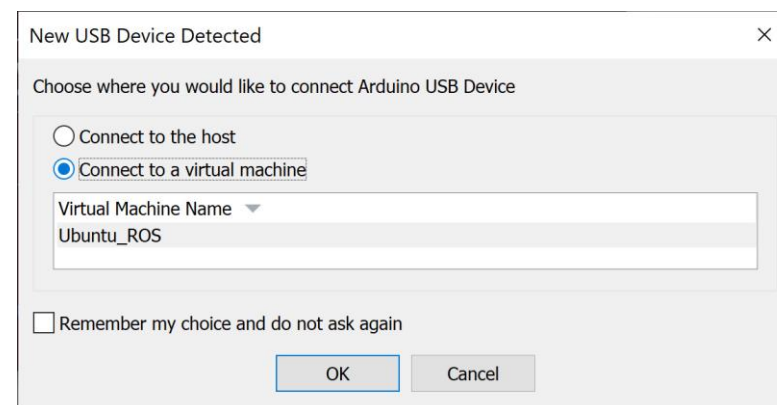


Activity 2



Uploading (Arduino IDE)

- Connect the Hackerboard or the ESP32 board
- Select the port to be used Tools>Port
 - If working on the VM, you must first select the option Connect to a virtual machine when automatically prompted (shown) and then select the port.
 - If in WSL, follow the steps on the presentation: "MCR2_Micro_ROS_Installation".
- Select the board to be used Tools -> ESP32 Arduino > DOIT ESP32 DEVKIT V1





Activity 2



Uploading (Arduino IDE)

- Upload the code using the arrow on the top left corner of the IDE.



- The following message should appear on the IDE

```
Done uploading.  
  
Sketch uses 1488 bytes (4%) of program storage space.  
Global variables use 198 bytes (9%) of dynamic memory
```

Running the node (Computer)

- Connect the board to the computer with ROS.
- (In Ubuntu) Make sure the port permissions are granted for the user (Skip this step if already performed).
 - In a new terminal type `cd /dev` to visualise the port designated by Ubuntu to the MCU. This port are usually called `/ttyACM0` or `/ttyUSB0`.

```
sudo chmod 666 /dev/ttyACM*  
sudo chmod 666 /dev/ttyUSB*
```




Activity 2



Test (Computer)

1. Open a terminal and type the following.

```
$ ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/tty***0
```

2. Reset the ESP32 (pressing the reset button) to reconnect to the computer (agent waiting timeout).

```
mario@MarioPC:~/uros_ws$ ros2 run micro_ros_agent mi
cro_ros_agent serial --dev /dev/ttyUSB0
[1737636692.137695] info | TermiosAgentLinux.cpp
| init | running...
| fd: 3
[1737636692.138467] info | Root.cpp |
set_verbose_level | logger setup
verbose_level: 4
[1737636698.665805] info | Root.cpp |
create_client | create
client_key: 0x0C9424D2, session_id: 0x81
[1737636698.666090] info | SessionManager.hpp |
establish_session | session established
client_key: 0x0C9424D2, address: 0
[1737636698.702311] info | ProxyClient.cpp |
create_participant | participant created
client_key: 0x0C9424D2, participant_id: 0x000(1)
[1737636698.721583] info | ProxyClient.cpp |
create_topic | topic created
client_key: 0x0C9424D2, topic_id: 0x000(2), particip
ant_id: 0x000(1)
[1737636698.733573] info | ProxyClient.cpp |
create_publisher | publisher created
client_key: 0x0C9424D2, publisher_id: 0x000(3), part
```

- Open another terminal and type the following.

```
$ ros2 topic list
```

```
mario@MarioPC:~$ ros2 topic list
/micro_ros_sub
/parameter_events
/rosout
```

- Publish to the topic
“/micro_ros_subscriber_topic”

```
$ ros2 topic pub /micro_ros_sub std_msgs/msg/Float32 "data: 0.3"
```

```
mario@MarioPC:~$ ros2 topic pub /micro_ros_sub std_m
sgs/msg/Float32 "data: 0.3"
publisher: beginning loop
publishing #1: std_msgs.msg.Float32(data=0.3)

publishing #2: std_msgs.msg.Float32(data=0.3)

publishing #3: std_msgs.msg.Float32(data=0.3)

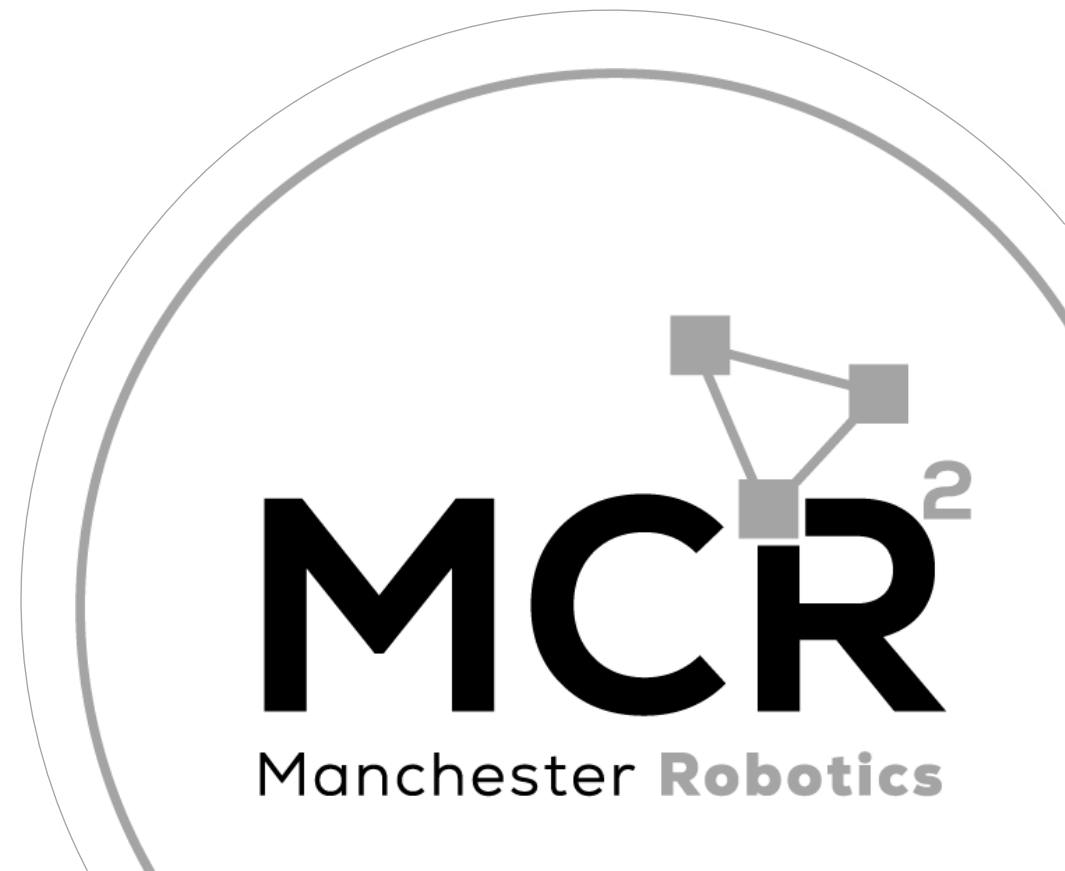
publishing #4: std_msgs.msg.Float32(data=0.3)

publishing #5: std_msgs.msg.Float32(data=0.3)
```

Micro-ROS Serial Communication

*Activity 2.2: Subscriber
w/reconnection*

{Learn, Create, Innovate};





Activity 2.2



Instructions (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 or Arduino Due
 - For ESP32 select ESP32 Arduino > DOIT ESP32 DEVKIT V1
- Open the previous Activity and modify it according to the following slides. Modifications are highlighted in Black.
- For this activity, no LED on Pin 22 is required

```
1 // Traffic Light Basic Code
2 /* Kat Nelms, Manchester Robotics, Feb 19 2022
3 // adapted from Arduino Neopixel Library "RGBWstrandtest"
4 // "Nlight_oneway" = choose between one or two TLBs, only one-way traffic
5 // one MCU board (MCUB) and up to two traffic lights (TLB)
6 // next level of complexity up from "lint_1way" script
7 // should initialize TLB1 to green and, if TLB2, TLB2 initialized to red
8 // then change green > yellow > red > yellow > green according to hard coded timers
9 */
10
11 /* SETUP */
12 #define LED_PIN 52 // Define the data out pin on the ESP32
13 #define greenTimer 2 //how long each color lasts, units are sec
14 #define redTimer 2
15 #define yellowTimer 1
16 /* END SETUP */
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2
```



Activity 2.2



```
// Include Libraries to be used
#include <micro_ros_arduino.h> //micro-ros-arduino library
#include <rcl/rcl.h> //Core ROS 2 Client Library (RCL) for
node management.
#include <rcl/error_handling.h> //Error handling utilities for Micro-ROS.
#include <rclcpp/rclcpp.h> //Micro-ROS Client library for embedded
devices.
#include <rclcpp/executor.h> //Micro-ROS Executor to manage callbacks
#include <std_msgs/msg/float32.h> //Predefined ROS 2 message type
#include <rmw_microros/rmw_microros.h> //ROS Middleware for Micro-ROS,
provides functions for interfacing Micro-ROS with DDS.
#include <stdio.h> //Standard I/O library for debugging.
```

```
//Declare nodes to be used
rcl_node_t node; //Represents a ROS 2 Node running on the
microcontroller.
```

```
//Instantiate executor and its support classes
rclcpp_executor_t executor; //Manages task execution (timers, callbacks,
etc.).
rclcpp_support_t support; //Data structure that holds the execution
context of Micro-ROS, including its communication state, memory management,
and initialization data.
rcl_allocator_t allocator; //Manages memory allocation.
```

```
//Declare Subscribers to be used
rcl_subscription_t subscriber;
```

```
//Declare Messages to be used
std_msgs__msg__Float32 msg; //Defines a message of type float32.
```

```
//Define Macros to be used
//Executes fn and returns false if it fails.
#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){return
false;}}
// Executes fn, but ignores failures.
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc != RCL_RET_OK)){}}
// Executes a given statement (X) periodically every MS milliseconds
#define EXECUTE_EVERY_N_MS(MS, X) do { \
    static volatile int64_t init = -1; \
    if (init == -1) { init = uxr_millis();} \
    if (uxr_millis() - init > MS) { X; init = uxr_millis();} \
} while (0)\
```

```
//Defines State Machine States
enum states {
    WAITING_AGENT,
    AGENT_AVAILABLE,
    AGENT_CONNECTED,
    AGENT_DISCONNECTED
} state;
```

```
//Specifies GPIO pin 13 for controlling an LED
#define PWM_PIN 22 //DEFINE PWM_PIN
#define PWM_FRQ 5000 //Define PWM Frequency
#define PWM_RES 8 //Define PWM Resolution
#define PWM_CHNL 0 //Define Channel
#define MSG_MIN_VAL 0 //Define min input value
#define MSG_MAX_VAL 1 //Define max input value
//Variables to be used
float pwm_set_point = 0.0;
```



Activity 2.2



```
//Create entity functions
bool create_entities();
void destroy_entities();

//Define callbacks
void subscription_callback(const void * msgin)
{
    //Get the message received and store it on the message msg
    const std_msgs__msg__Float32 * msg = (const std_msgs__msg__Float32
*)msgin;
    pwm_set_point = constrain(msg->data, MSG_MIN_VAL, MSG_MAX_VAL);
    ledcWrite(PWM_CHNL, (uint32_t) (pow(2, PWM_RES) * (pwm_set_point /
1.0)));
}

//Setup
void setup() {
    set_microros_transports(); // Initializes communication between ESP32 and
the ROS 2 agent (Serial).
    //Setup Microcontroller Pins
    pinMode(PWM_PIN, OUTPUT);
    ledcSetup(PWM_CHNL, PWM_FRQ, PWM_RES); //Setup the PWM
    ledcAttachPin(PWM_PIN, PWM_CHNL); //Setup Attach the Pin to the Channel
}
```

```
void loop() {
    switch (state) {

        case WAITING_AGENT:
            EXECUTE_EVERY_N_MS(500, state = (RMW_RET_OK == rmw_uros_ping_agent(100,
1)) ? AGENT_AVAILABLE : WAITING_AGENT;);
            break;

        case AGENT_AVAILABLE:
            state = (true == create_entities()) ? AGENT_CONNECTED : WAITING_AGENT;
            if (state == WAITING_AGENT) {
                destroy_entities();
            };
            break;

        case AGENT_CONNECTED:
            EXECUTE_EVERY_N_MS(200, state = (RMW_RET_OK == rmw_uros_ping_agent(100,
1)) ? AGENT_CONNECTED : AGENT_DISCONNECTED;);
            if (state == AGENT_CONNECTED) {
                rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100));
            }
            break;

        case AGENT_DISCONNECTED:
            destroy_entities();
            state = WAITING_AGENT;
            break;

        default:
            break;
    }
}
```



Activity 2.2



```
bool create_entities()
{
    //Initializes memory allocation for Micro-ROS operations.
    allocator = rcl_get_default_allocator();

    //Creates a ROS 2 support structure to manage the execution context.
    RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

    // create node
    RCCHECK(rclc_node_init_default(&node, "micro_ros_sub_node", "",
    &support));

    // create subscriber
    RCCHECK(rclc_subscription_init_default(
        &subscriber,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
        "micro_ros_sub"));

    // create zero initialised executor (no configured) to avoid memory
problems
    executor = rclc_executor_get_zero_initialized_executor();
    // Initializes the Micro-ROS Executor, which manages tasks and callbacks.
    RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
    // Register suscription with executor
    RCCHECK(rclc_executor_add_subscription(&executor, &subscriber, &msg,
    &subscription_callback, ON_NEW_DATA));

    return true;
}
```

```
void destroy_entities()
{
    rmw_context_t * rmw_context = rcl_context_get_rmw_context(&support.context);
    (void) rmw_uos_set_context_entity_destroy_session_timeout(rmw_context, 0);

    rcl_subscription_fini(&subscriber, &node);
    rclc_executor_fini(&executor);
    rcl_node_fini(&node);
    rclc_support_fini(&support);
}
```

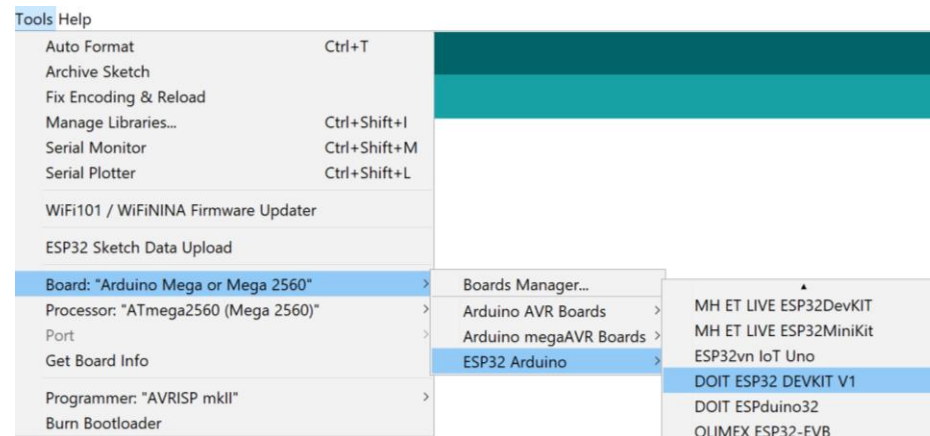


Activity



Compilation (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 (for Hackerboard is the same)
 - For ESP32 select ESP32 Arduino > DOIT ESP32 DEVKIT V1



- Compile the code using by clicking check mark button located on the upper left corner.



- The following message should be displayed:

Done compiling.

Sketch uses 9424 bytes (3%) of program storage space. Maximum is 253952 bytes.
Global variables use 1826 bytes (22%) of dynamic memory, leaving 6366 bytes for local variables. Maximum is 8192 bytes.

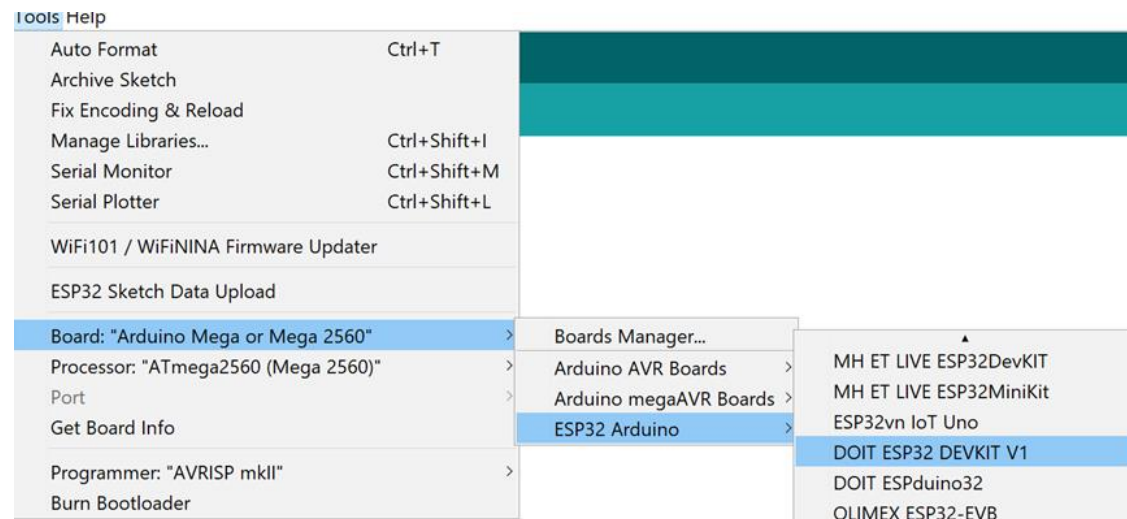
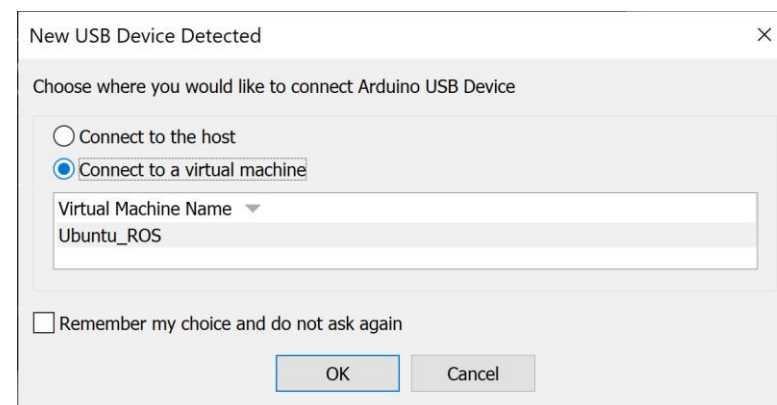


Activity



Uploading (Arduino IDE)

- Connect the Hackerboard or the ESP32 board
- Select the port to be used Tools>Port
 - If working on the VM, you must first select the option Connect to a virtual machine when automatically prompted (shown) and then select the port.
 - If in WSL, follow the steps on the presentation: "MCR2_Micro_ROS_Installation".
- Select the board to be used Tools -> ESP32 Arduino > DOIT ESP32 DEVKIT V1





Activity



Uploading (Arduino IDE)

- Upload the code using the arrow on the top left corner of the IDE.



- The following message should appear on the IDE

```
Done uploading.  
Sketch uses 1488 bytes (4%) of program storage space.  
Global variables use 198 bytes (9%) of dynamic memory
```

Running the node (Computer)

- Connect the board to the computer with ROS.
- (In Ubuntu) Make sure the port permissions are granted for the user (Skip this step if already performed).
 - In a new terminal type `cd /dev` to visualise the port designated by Ubuntu to the MCU. This port are usually called `/ttyACM0` or `/ttyUSB0`.

```
sudo chmod 666 /dev/ttyACM*  
sudo chmod 666 /dev/ttyUSB*
```



Activity 2.2



Test (Computer)

1. Open a terminal and type the following.

```
$ ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/tty***0
```

```
mario@MarioPC:~/uros_ws$ ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/ttyUSB0
[1737636692.137695] info | TermiosAgentLinux.cpp |
| init | running... |
| fd: 3 | |
[1737636692.138467] info | Root.cpp |
set_verbose_level | logger setup |
verbose_level: 4 |
[1737636698.665805] info | Root.cpp |
create_client | create |
client_key: 0x0C9424D2, session_id: 0x81 |
[1737636698.666090] info | SessionManager.hpp |
establish_session | session established |
client_key: 0x0C9424D2, address: 0 |
[1737636698.702311] info | ProxyClient.cpp |
create_participant | participant created |
client_key: 0x0C9424D2, participant_id: 0x000(1) |
[1737636698.721583] info | ProxyClient.cpp |
create_topic | topic created |
client_key: 0x0C9424D2, topic_id: 0x000(2), participant_id: 0x000(1) |
[1737636698.733573] info | ProxyClient.cpp |
create_publisher | publisher created |
client_key: 0x0C9424D2, publisher_id: 0x000(3), part
```

- Open another terminal and type the following.

```
$ ros2 topic list
```

```
mario@MarioPC:~$ ros2 topic list
/micro_ros_sub
/parameter_events
/rosout
```

- Publish to the topic “/micro_ros_sub”

```
$ ros2 topic pub /micro_ros_sub std_msgs/msg/Float32 "data: 0.3"
```

```
mario@MarioPC:~$ ros2 topic pub /micro_ros_sub std_msgs/msg/Float32 "data: 0.3"
publisher: beginning loop
publishing #1: std_msgs.msg.Float32(data=0.3)

publishing #2: std_msgs.msg.Float32(data=0.3)

publishing #3: std_msgs.msg.Float32(data=0.3)

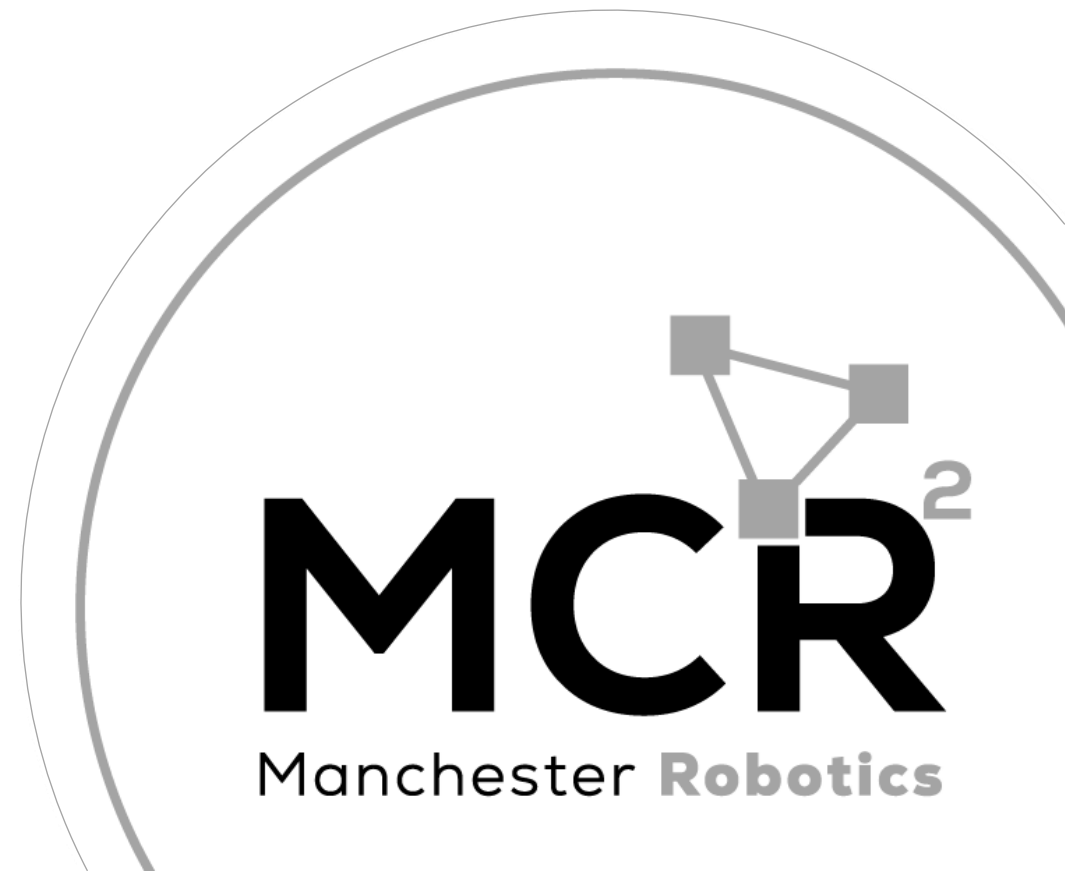
publishing #4: std_msgs.msg.Float32(data=0.3)

publishing #5: std_msgs.msg.Float32(data=0.3)
```

Micro-ROS Serial Communication

*Activity 3: Publisher
and Subscriber*

{Learn, Create, Innovate};



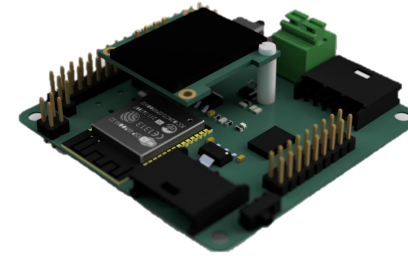


Requirements



- The following activity is based on the example tutorial found in the provided micro-ROS libraries.
- This activity requires Arduino IDE to be installed and configured as shown in “MCR2_Micro_ROS_Installation”.

- Requirements:
 - Microcontroller
 - Computer
 - micro-usb to USB cable (Data)
 - Breadboard
 - LED
 - 1 x 150 Ohm Resistor
 - 1x 10kOhm Resistor
 - NO Pushbutton

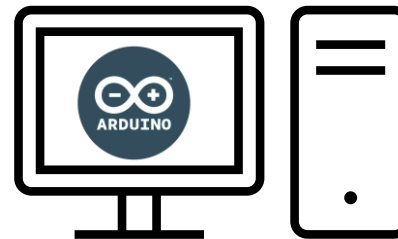


Hackerboard

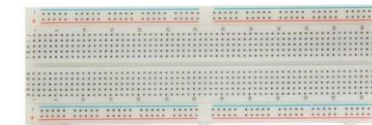
Or



ESP32 board



Computer



Breadboard



LED



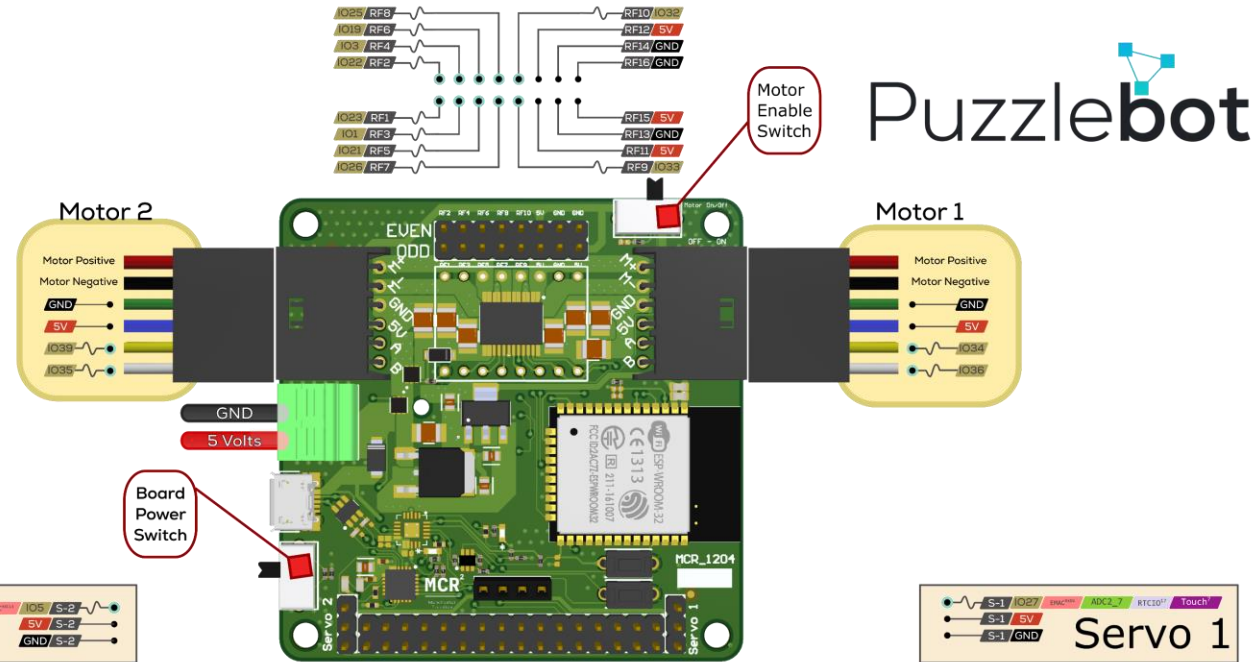
Push Button
(Normally
open)



10kΩ , 150Ω
Resistor



Hackerboard Pinout



- 5V comes from LDO regulated supply, max current draw is 800mA
- Absolute maximum current draw per GPIO pin is 40mA
- Input voltage is 5-13.5V via terminal block
- Current draw via microUSB socket is 1.8A max

- Puzzlebot Devices
- Power
- GND
- Serial Pin
- Analogue Pin
- Control Pin
- Physical Pin Number
- GPIO Pin
- Touch Pin
- DAC Pin
- Port Pin
- PWM Pin

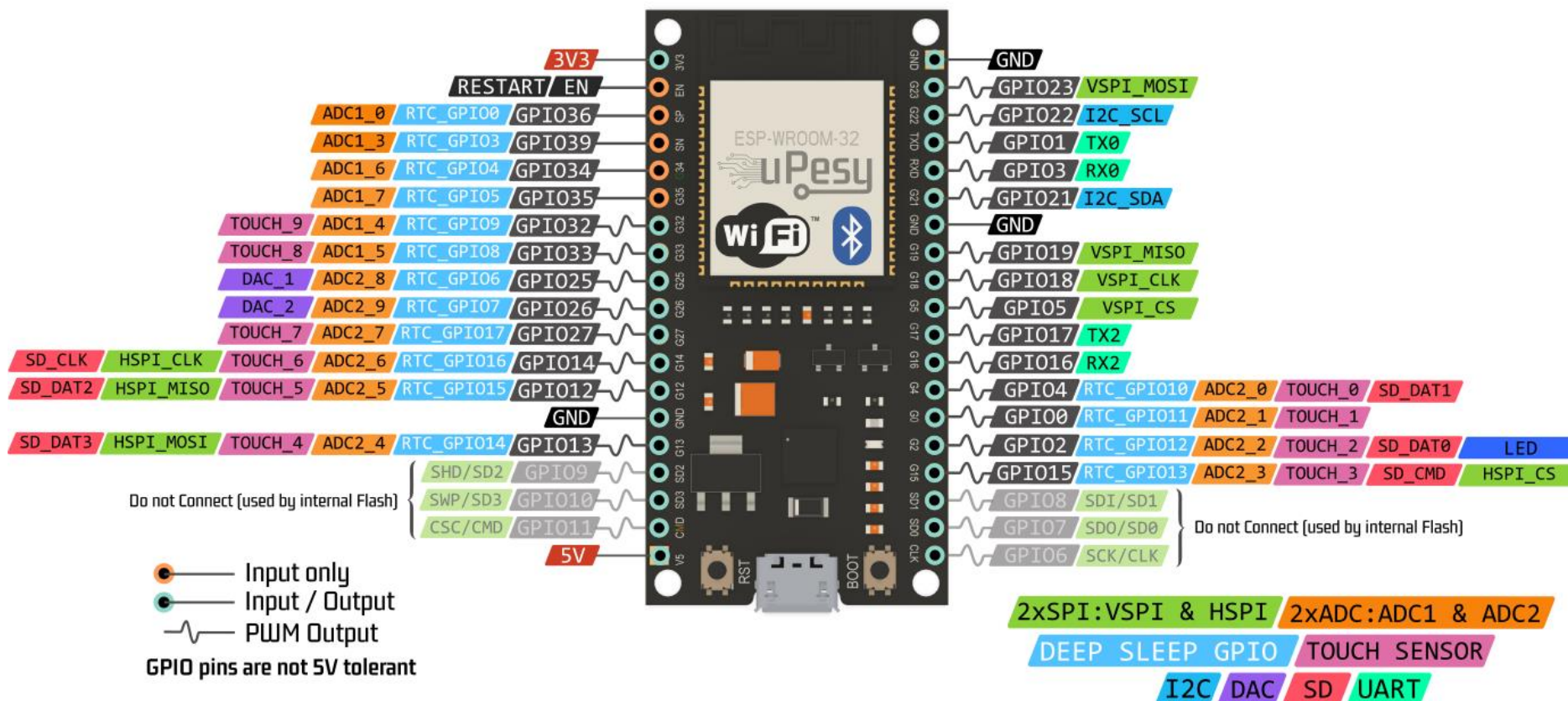
- Notes:**
- These pins are **input only**. They cannot output signals
 - These pins are integrated to the Microcontroller's flash memory, and are only present for debugging purposes. They should not be used for general purpose
 - Outputs PWM signal at boot time
 - Boot fails if pulled high
 - Boot fails if pulled low

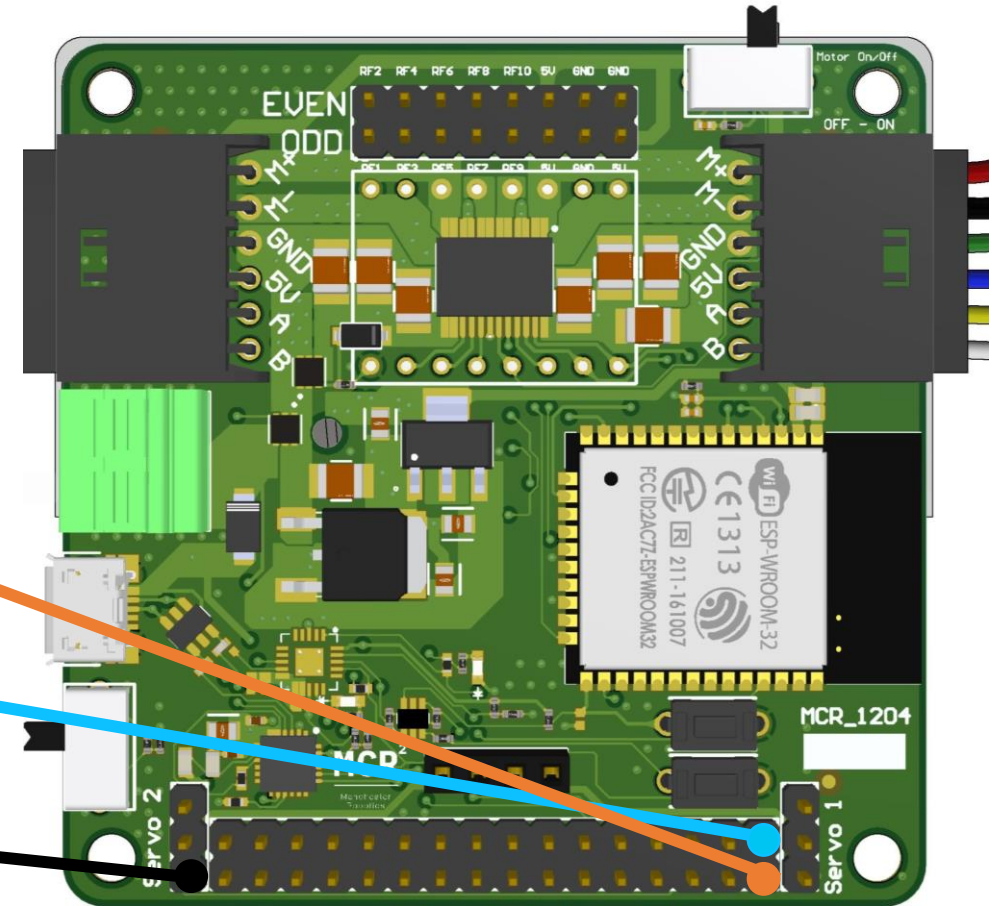
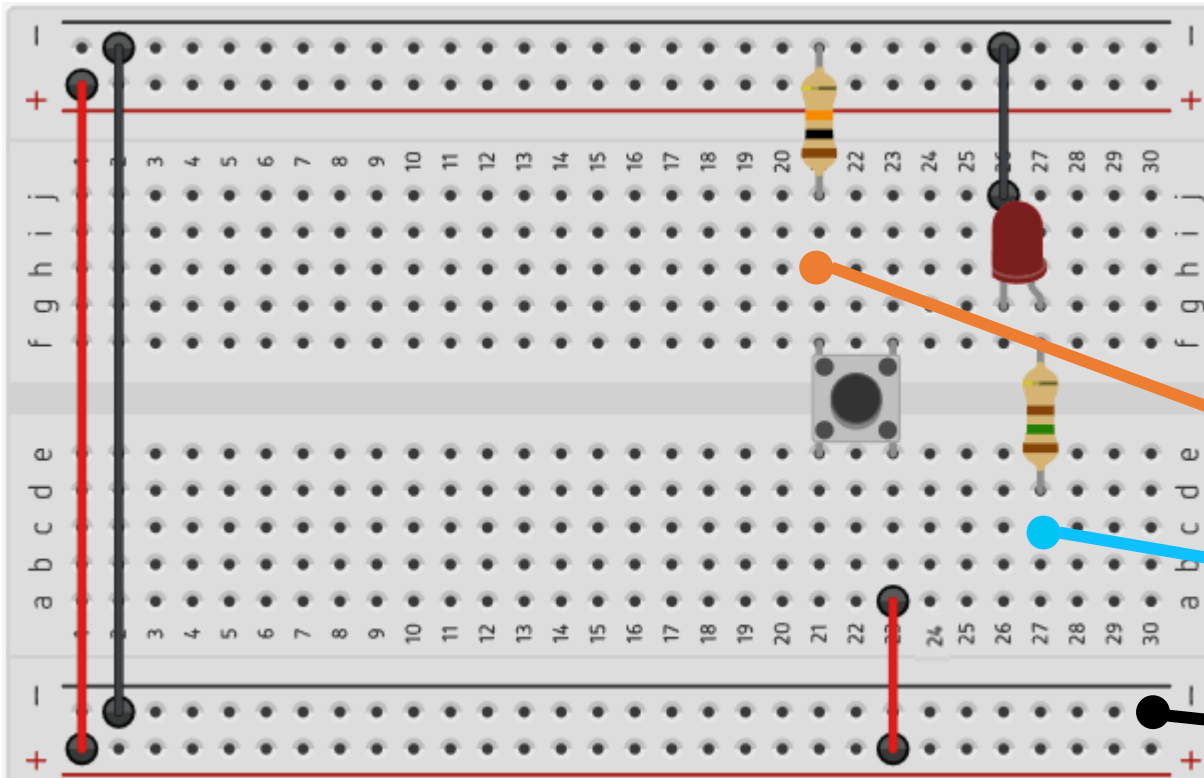
<https://www.manchester-robotics.com>



01 FEB 2022
v1
Derivative of "Puzzlebot Template" by AthlonOne, used under CC BY SA.
MCR_1204 Pinout is licensed under CC BY SA by MCR

ESP32 Wroom DevKit Full Pinout





GND

Pin 22

Pin 23



Description



Objective

- The microcontroller should detect the state of a pushbutton to enable or disable LED brightness control.
 - If active, the user can control the LED brightness from a host computer via ROS 2.
 - If inactive, the LED brightness remains fixed at 0% (off).

Implementation:

- The system must be implemented using ROS 2 and Micro-ROS.
- The button's state should only update after a **complete press-release cycle**.

ROS 2 Node & Communication

- The microcontroller will run a ROS 2 node called "esp32_button_led_node".
- Publish button state using a String ("active" or "inactive") to "button_state" topic (every 100ms).
- Subscribe to "led_brightness" (0 to 1) to control LED intensity.

System Behavior

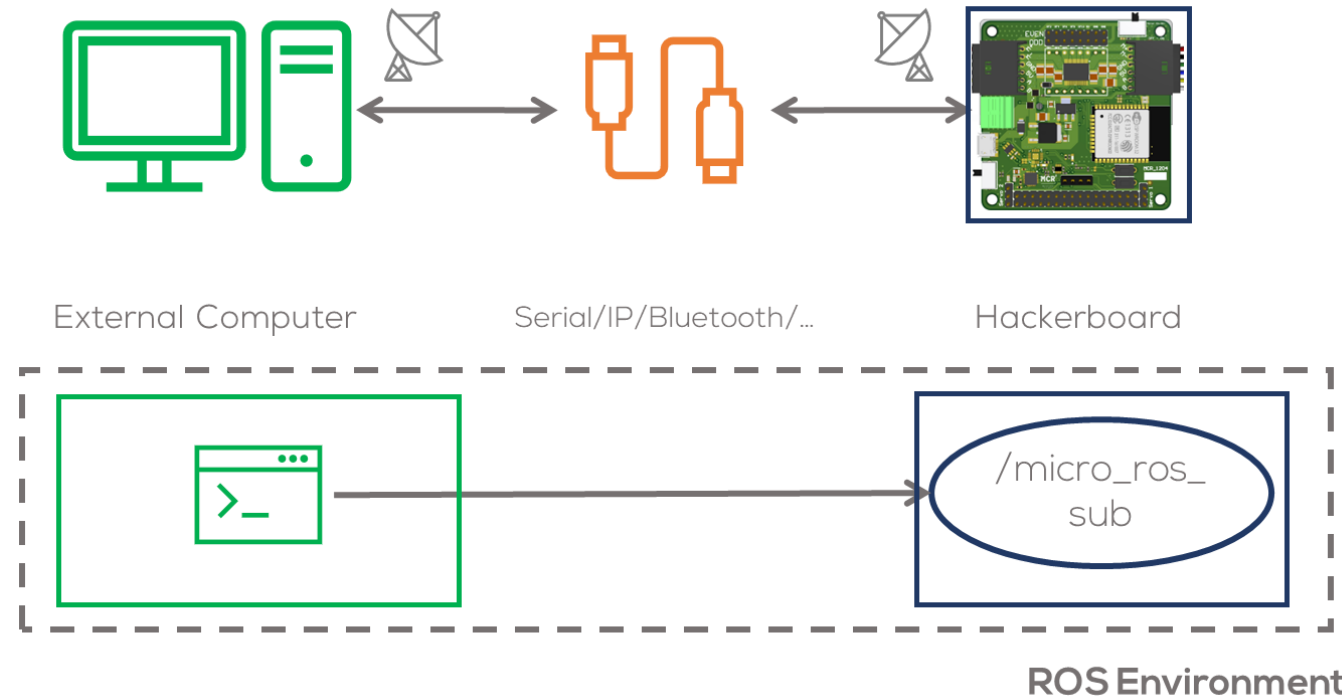
- The pushbutton state is debounced to prevent false activations.
- The host computer sends a value in the range [0,1] to adjust LED brightness. 0 = LED OFF (0%), 1 = LED FULL BRIGHTNESS (100%)



Description



- In this activity, a node running a publisher and a subscriber will be made.
- This node will run inside the microcontroller and will communicate with the computer via UART.
- The node will subscribe to a Float32 message and publish a String message.
- This activity will be divided into two parts. The first part involves the Arduino IDE in programming the MCU.
- The second part involves the commands to connect the board to the computer.



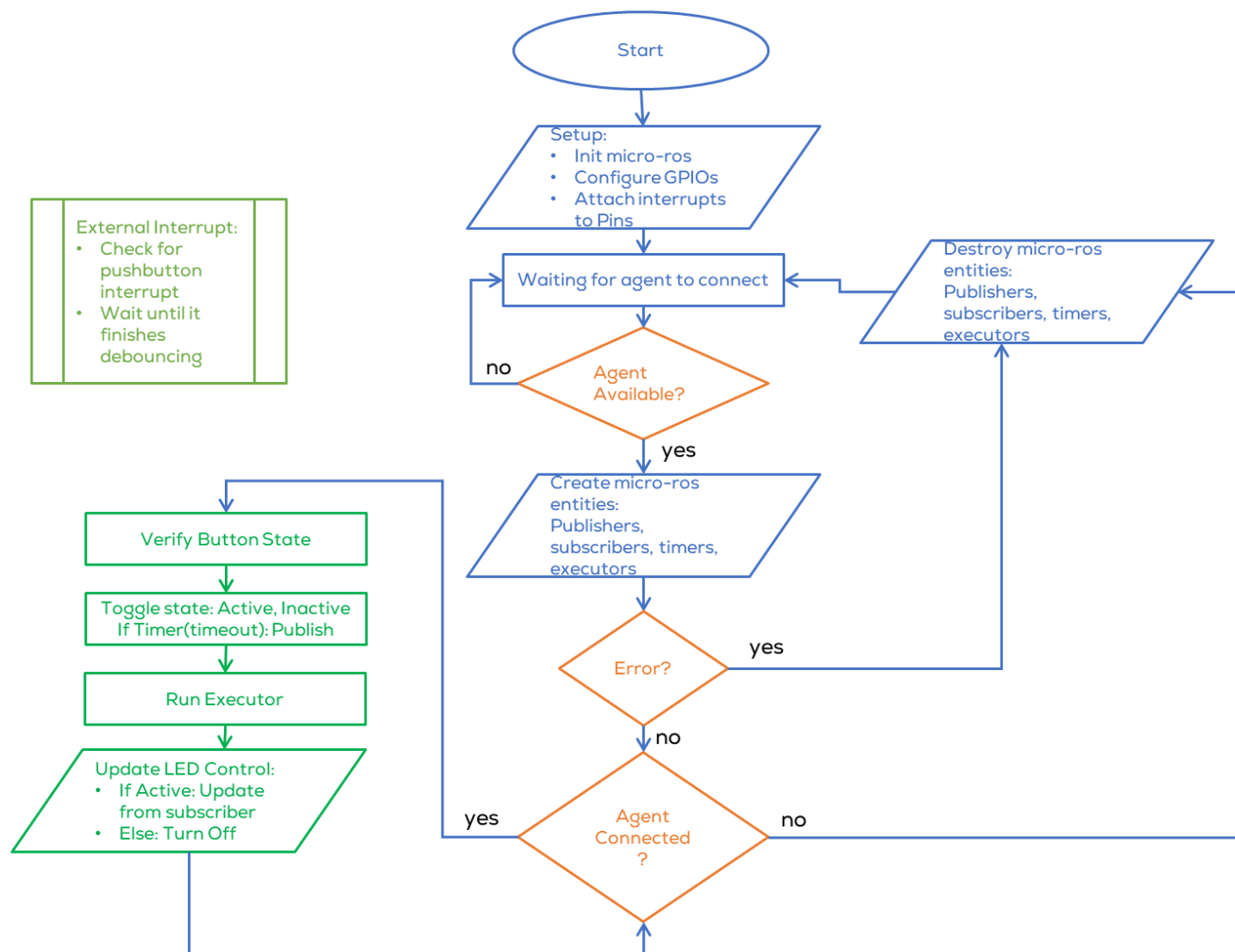


Activity 3: Publisher and Subscriber



Introduction

- This activity is focused on integrating the previous concepts into a bigger project.
- Verify the correct wiring and connections before powering up the system.
- Open the file “publisher_subscriber.ino”
- The code for the ESP32 is too large, a flowchart shown describes how the code works.



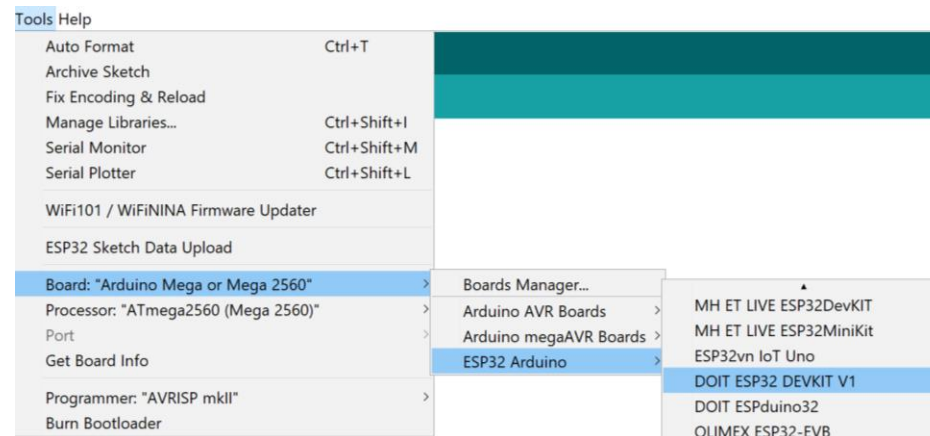


Activity



Compilation (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 (for Hackerboard is the same)
 - For ESP32 select ESP32 Arduino > DOIT ESP32 DEVKIT V1



- Compile the code using by clicking check mark button located on the upper left corner.



- The following message should be displayed:

Done compiling.

Sketch uses 9424 bytes (3%) of program storage space. Maximum is 253952 bytes.
Global variables use 1826 bytes (22%) of dynamic memory, leaving 6366 bytes for local variables. Maximum is 8192 bytes.

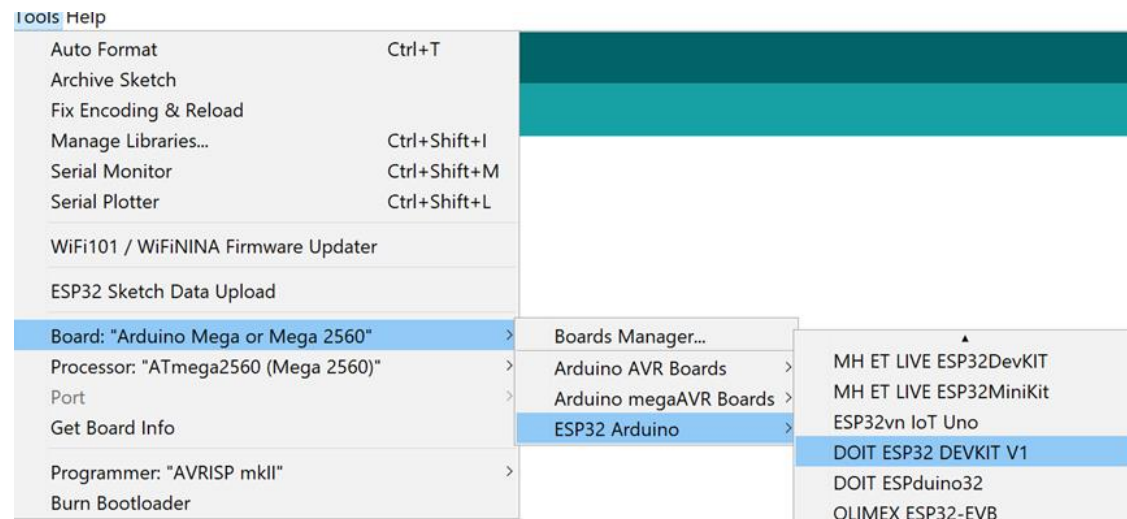
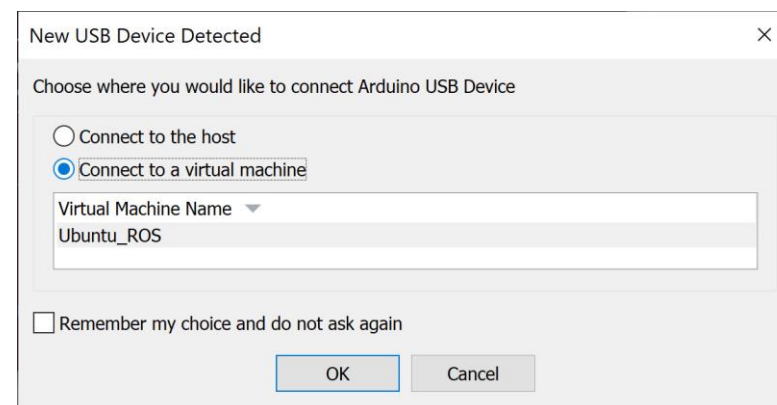


Activity



Uploading (Arduino IDE)

- Connect the Hackerboard or the ESP32 board
- Select the port to be used Tools>Port
 - If working on the VM, you must first select the option Connect to a virtual machine when automatically prompted (shown) and then select the port.
 - If in WSL, follow the steps on the presentation: "MCR2_Micro_ROS_Installation".
- Select the board to be used Tools -> ESP32 Arduino > DOIT ESP32 DEVKIT V1





Activity



Uploading (Arduino IDE)

- Upload the code using the arrow on the top left corner of the IDE.



- The following message should appear on the IDE

```
Done uploading.  
Sketch uses 1488 bytes (4%) of program storage space.  
Global variables use 198 bytes (9%) of dynamic memory
```

Running the node (Computer)

- Connect the board to the computer with ROS.
- (In Ubuntu) Make sure the port permissions are granted for the user (Skip this step if already performed).
 - In a new terminal type `cd /dev` to visualise the port designated by Ubuntu to the MCU. This port are usually called `/ttyACM0` or `/ttyUSB0`.

```
sudo chmod 666 /dev/ttyACM*  
sudo chmod 666 /dev/ttyUSB*
```



Activity 3



Test (Computer)

1. Open a terminal and type the following.

```
$ ros2 run micro_ros_agent micro_ros_agent serial --dev /dev/tty***0
```

```
mario@MarioPC:~/uros_ws$ ros2 run micro_ros_agent mi
cro_ros_agent serial --dev /dev/ttyUSB0
[1737636692.137695] info      | TermiosAgentLinux.cpp
| init                      | running...
| fd: 3
[1737636692.138467] info      | Root.cpp
set_verbose_level          | logger setup
verbose_level: 4
[1737636698.665805] info      | Root.cpp
create_client              | create
client_key: 0x0C9424D2, session_id: 0x81
[1737636698.666090] info      | SessionManager.hpp
establish_session          | session established
client_key: 0x0C9424D2, address: 0
[1737636698.702311] info      | ProxyClient.cpp
create_participant         | participant created
client_key: 0x0C9424D2, participant_id: 0x000(1)
[1737636698.721583] info      | ProxyClient.cpp
create_topic               | topic created
client_key: 0x0C9424D2, topic_id: 0x000(2), particip
ant_id: 0x000(1)
[1737636698.733573] info      | ProxyClient.cpp
create_publisher           | publisher created
client_key: 0x0C9424D2, publisher_id: 0x000(3), part
```

2. Open another terminal and type the following.

```
$ ros2 topic list
```

```
mario@MarioPC:~$ ros2 topic list
/button_state
/led_brightness
/parameter_events
/rosout
```

3. Echo the topic “/button_state”. Press the Button!

```
$ ros2 topic echo /button_state
```

4. Publish to the LED on the topic “/led_brightness”

```
$ ros2 topic pub /led_brightness std_msgs/msg/Float32 "data: 0.5"
```

```
mario@MarioPC:~$ ros2 topic echo /button_state
data: inactive
---
data: inactive
---
data: inactive
---
```

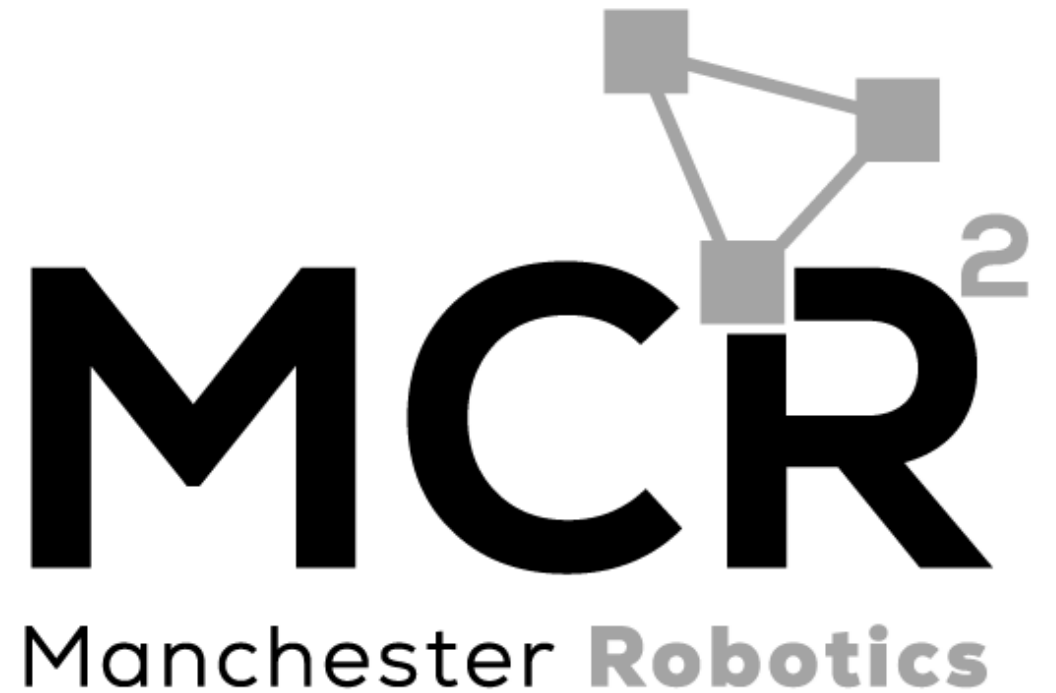
```
mario@MarioPC:~$ ros2 topic pub /led_brightness std_
msgs/msg/Float32 "data: 0.5"
publisher: beginning loop
publishing #1: std_msgs.msg.Float32(data=0.5)

publishing #2: std_msgs.msg.Float32(data=0.5)
publishing #3: std_msgs.msg.Float32(data=0.5)
```

Micro-ROS Communication

*Micro-ros Wi-Fi
communication*

{Learn, Create, Innovate};



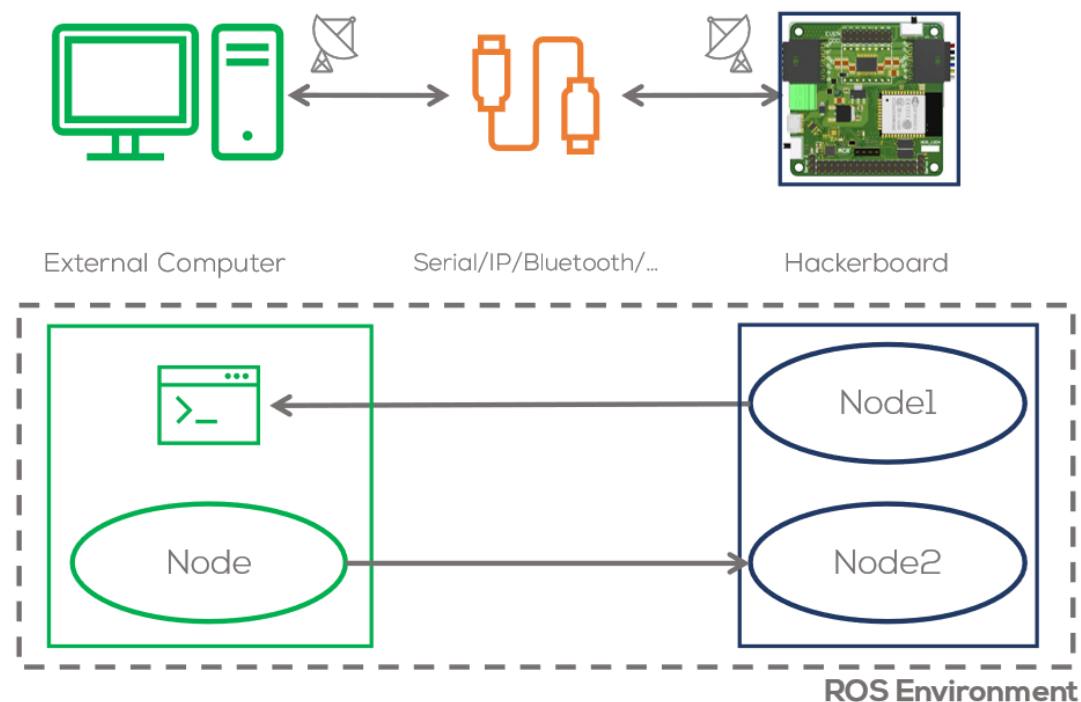


Wi-Fi communication



Introduction

- Embedded systems. Typically communicate via serial, communication.
- Micro-ROS enables communication between embedded systems and a ROS 2 network using various wired and wireless protocols.
- Micro-ROS provides flexibility in choosing transport layers for data exchange.
- Selecting the right transport depends on hardware capabilities, network requirements, and application constraints.



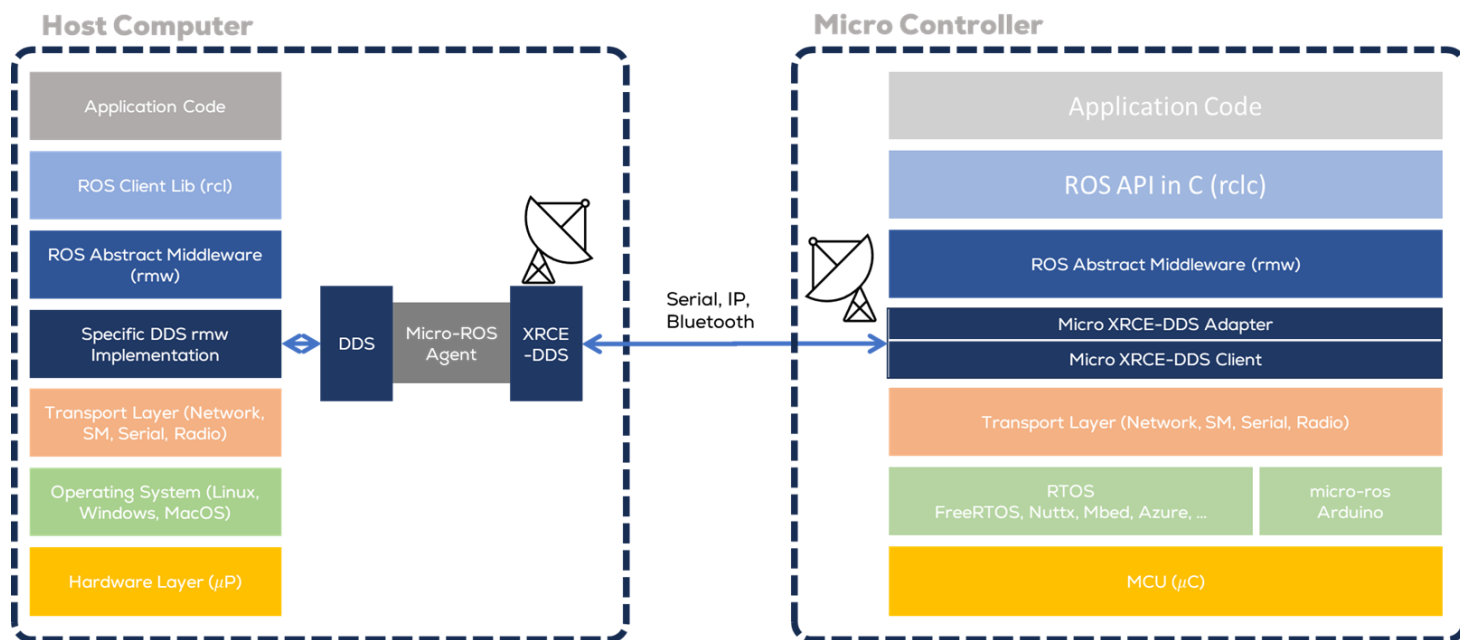


Wi-Fi communication



Micro-ros transports

- Micro-ROS transports define how data is sent between embedded devices and the ROS 2 agent.
- They enable communication between microcontrollers and the ROS 2 agent running on a host.
- The user can define its own communication protocol (micro-ros transports)

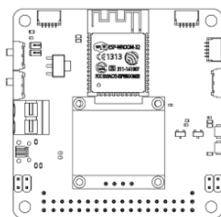




External Computer



Wi-Fi



Hackerboard
Control Unit

Actuators:

Servo Motors, DC
Motors, Stepper
Motors

Sensors:

Encoders,
Reflectance,
Sonar, etc.

Types of predefined Micro-ROS Transports

- Serial (UART, USB): Direct connection via serial ports (e.g., USB, UART).
- Wi-Fi (UDP/TCP): Wireless communication over a network.
- Ethernet: Wired high-speed communication.
- CAN Bus: Common in automotive and industrial applications.

This section will focus on Wi-Fi communication using micro-ros.



Wi-Fi communication



Wi-Fi for Micro-ROS

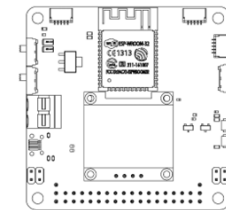
- Provides wireless communication between microcontrollers and a ROS 2 host.
- Enables remote control and monitoring of embedded robotic systems.
- Suitable for mobile robots that require untethered operation.



External Computer



Wi-Fi



Hackerboard
Control Unit

Actuators:
Servo Motors, DC
Motors, Stepper
Motors

Sensors:
Encoders,
Reflectance,
Sonar, etc.



Wi-Fi communication



HOST
(AGENT RUNNING)

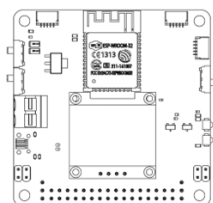


External Computer



Wi-Fi

Microcontroller



Hackerboard
Control Unit

Actuators:

Servo Motors, DC
Motors, Stepper
Motors

Sensors:

Encoders,
Reflectance,
Sonar, etc.

How Wi-Fi Transport Works

The microcontroller connects to a Wi-Fi network or generates an access point (AP).

The Micro-ROS Agent runs on a host computer.

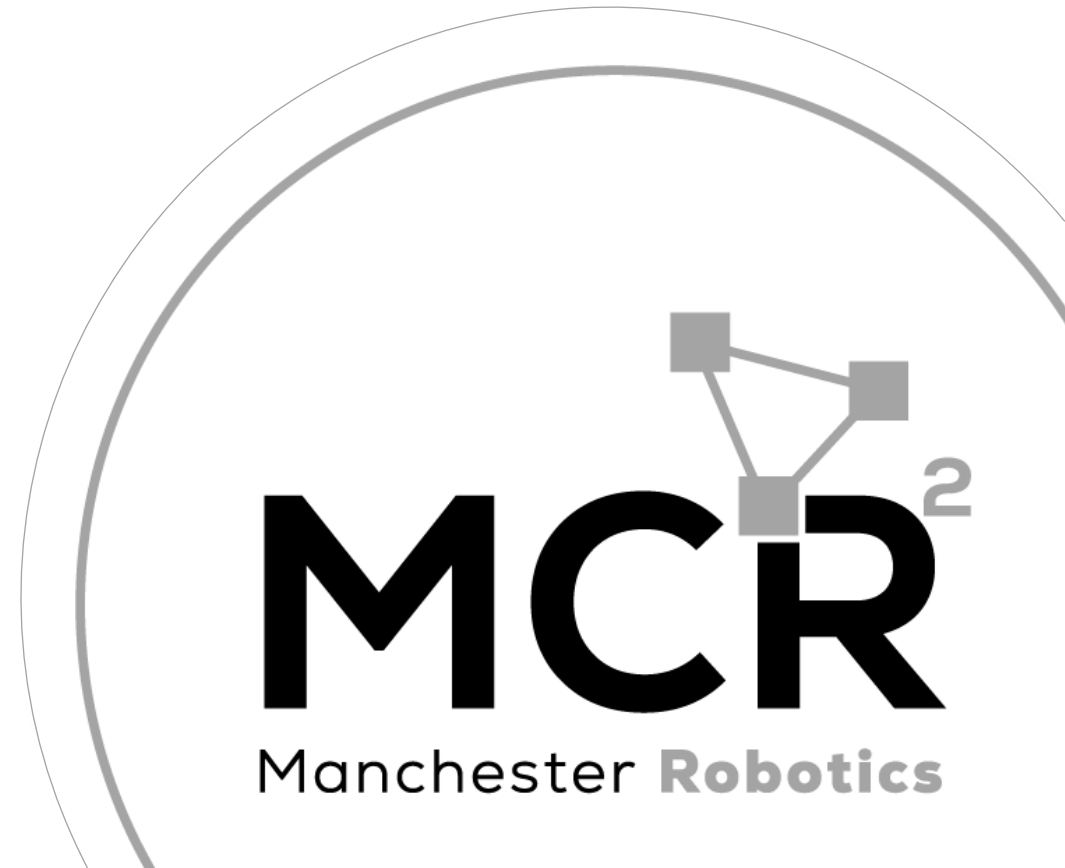
Data is exchanged between the microcontroller and the host using UDP or TCP.

In this section the microcontroller will be set as an access point and communicate using Wi-Fi to the agent.

Micro-ROS Serial Communication

*Activity 4: Wi-Fi
Publisher*

{Learn, Create, Innovate};

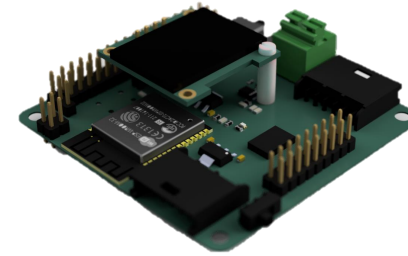




Requirements



- The following activity is based on the example tutorial found in the provided micro-ROS libraries.
- This activity requires Arduino IDE to be installed and configured as shown in “MCR2_Micro_ROS_Installation”.
- Requirements:
 - Microcontroller
 - Computer
 - micro-usb to USB cable (Data)

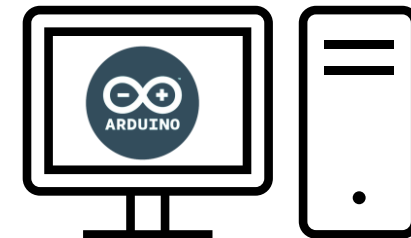


Hackerboard

Or



ESP32 board



Computer

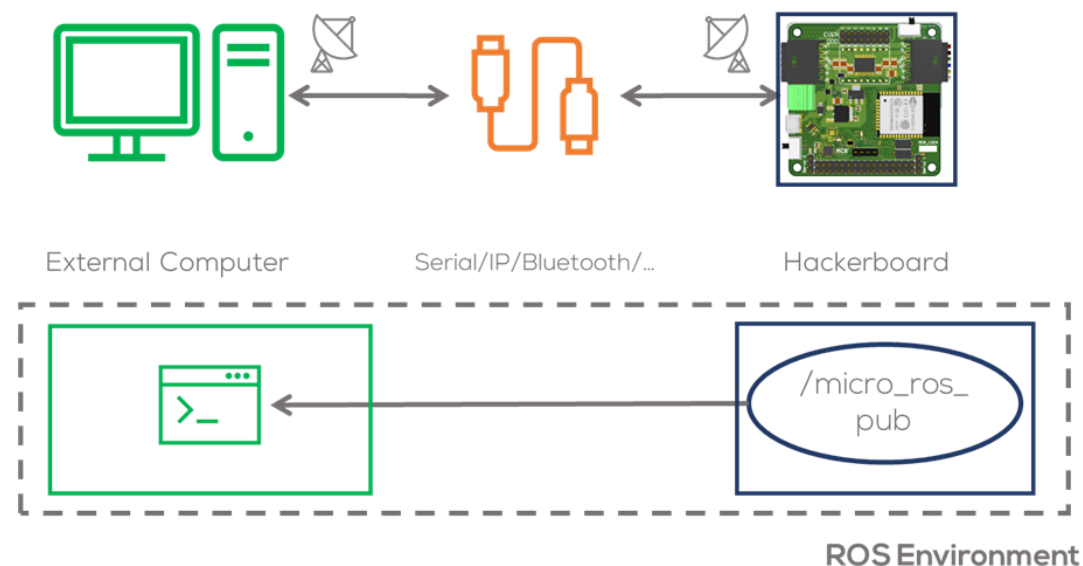


Description



Objective

- In this activity, a node running a simple publisher inside the microcontroller will be declared.
- This node will run inside the microcontroller and will communicate with the computer via Wi-Fi.
- The node will publish a simple int32 message.
- This activity will be divided into two parts. The first part involves the Arduino IDE to program the MCU.
- The second part involving the commands required to connect to the board to the computer.





Activity 4



```
#include <micro_ros_arduino.h> // Micro-ROS library for Arduino
#include <WiFi.h>               // Wi-Fi library for ESP32
#include <WiFiUdp.h>            // UDP communication over Wi-Fi

#include <stdio.h>              // Standard I/O library
#include <rcl/rcl.h>             // Core ROS 2 Client Library (RCL) for node
                                // management
#include <rcl/error_handling.h> // Error handling utilities
#include <rcl/rcl.h>             // Micro-ROS client library for embedded
                                // devices
#include <rcl/executor.h>        // Micro-ROS Executor to manage callbacks
#include <rmw_microros/rmw_microros.h> // ROS Middleware for Micro-ROS

#include <std_msgs/msg/int32.h> // Predefined ROS 2 message type (integer
                                // messages)
```

```
// Macros for Error Checking
#define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc !=
RCL_RET_OK)){return false;}} // Return false on failure
#define RMCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc !=
RMW_RET_OK)){error_loop();}} // Enter error loop on failure
#define RCSOFTCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc !=
RCL_RET_OK)){}}
```

```
// Macro for executing a function every N milliseconds
#define EXECUTE_EVERY_N_MS(MS, X) do { \
    static volatile int64_t init = -1; \
    if (init == -1) { init = uxr_millis();} \
    if (uxr_millis() - init > MS) { X; init = uxr_millis();} \
} while (0)\
```

```
// Micro-ROS entities
rcl_support_t support; // Holds the execution context of Micro-ROS
rcl_executor_t executor; // Manages task execution (timers, callbacks,
etc.)
rcl_allocator_t allocator; // Memory allocation manager

rcl_node_t node; // Represents a ROS 2 Node running on the
microcontroller
rcl_timer_t timer; // Timer for periodic message publishing
rcl_publisher_t publisher; // Publisher for sending messages to ROS 2
```

```
std_msgs__msg__Int32 msg; // Integer message type
```

```
micro_ros_agent_locator locator; // Stores connection details for Micro-ROS
Agent
// Static IP configuration for ESP32 Access Point
IPAddress local_ip = {10, 16, 1, 1};
IPAddress gateway = {10, 16, 1, 1};
IPAddress subnet = {255, 255, 255, 0};
// Wi-Fi credentials (ESP32 acting as an Access Point)
const char* ssid = "ESP32-Access-Point";
const char* password = "123456789";
// Micro-ROS Agent configuration (host machine)
const char* agent_ip = "10.16.1.2";
const int agent_port = 8888;
```




Activity 4



```
// Enum representing different connection states of the microcontroller
enum states {
    WAITING_AGENT,          // Waiting for a connection to the Micro-ROS agent
    AGENT_AVAILABLE,        // Agent found, trying to establish communication
    AGENT_CONNECTED,        // Connected to the agent, publishing messages
    AGENT_DISCONNECTED      // Lost connection, trying to reconnect
} state;

// Function that gets called if there is a failure in initialization
void error_loop(){
    while(1){
        // Toggle LED state
        printf("Failed initialisation. Aborting.\n"); // Print error message
        // Wait for 100 milliseconds before retrying
        delay(100);
    }
}

// Timer callback function, runs periodically to publish messages
void timer_callback(rcl_timer_t * timer, int64_t last_call_time)
{
    (void) last_call_time;
    if (timer != NULL) {
        rcl_publish(&publisher, &msg, NULL); // Publish message to ROS 2 topic
        msg.data++; // Increment message data for the next cycle
    }
}
```

```
// Function to create Micro-ROS entities (node, publisher, timer)
bool create_entities()
{
    // Initialize Micro-ROS support
    RCCHECK(rclc_support_init(&support, 0, NULL, &allocator));

    // Create ROS 2 node
    RCCHECK(rclc_node_init_default(&node, "int32_publisher_rclc", "",
    &support));

    // Create a best-effort publisher (non-reliable, no message history) (QoS)
    RCCHECK(rclc_publisher_init_best_effort(
        &publisher,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
        "std_msgs_msg_Int32"));

    // Create a timer to publish messages every 1000ms (1 second)
    const unsigned int timer_timeout = 1000;
    RCCHECK(rclc_timer_init_default(
        &timer,
        &support,
        RCL_MS_TO_NS(timer_timeout),
        timer_callback));

    // Initialize Executor (handles timer callbacks)
    executor = rclc_executor_get_zero_initialized_executor();
    RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator));
    RCCHECK(rclc_executor_add_timer(&executor, &timer));
    return true; // Return true if all entities are successfully created
}
```



Activity 4



```
// Function to clean up Micro-ROS entities when disconnected
void destroy_entities()
{
    rmw_context_t * rmw_context =
    rcl_context_get_rmw_context(&support.context);
    (void)
    rmw_uos_set_context_entity_destroy_session_timeout(rmw_context, 0);

    rcl_publisher_fini(&publisher, &node);
    rcl_timer_fini(&timer);
    rcl_executor_fini(&executor);
    rcl_node_fini(&node);
    rcl_support_fini(&support);
}
```

```
// Setup function - Runs once when ESP32 starts
void setup() {

    // Initialize memory allocator
    allocator = rcl_get_default_allocator();

    // Set up Micro-ROS agent connection details
    locator.address.fromString(agent_ip);
    locator.port = agent_port;

    // Set up ESP32 as a Wi-Fi Access Point
    WiFi.mode(WIFI_AP_STA);
    WiFi.softAP(ssid,password);
    delay(1000);
    WiFi.softAPConfig(local_ip, gateway, subnet);

    // Configure Micro-ROS transport using Wi-Fi
    RMCHECK(rmw_uos_set_custom_transport(
        false,
        (void *) &locator,
        arduino_wifi_transport_open,
        arduino_wifi_transport_close,
        arduino_wifi_transport_write,
        arduino_wifi_transport_read
    ));

    // Set initial state to waiting for ROS 2 Agent
    state = WAITING_AGENT;
    // Initialize message data
    msg.data = 0;
}
```



Activity 4



```
// Loop function - Runs continuously
void loop() {
    switch (state) {

        case WAITING_AGENT:
            // Try to ping the Micro-ROS agent every second
            EXECUTE_EVERY_N_MS(1000, state = (RMW_RET_OK ==
rmw_uros_ping_agent(100, 1)) ? AGENT_AVAILABLE : WAITING_AGENT;);
            break;

        case AGENT_AVAILABLE:
            // Try to create ROS entities, move to connected state if successful
            state = (true == create_entities()) ? AGENT_CONNECTED : WAITING_AGENT;
            if (state == WAITING_AGENT) {
                destroy_entities();
            };
            break;
    }
}
```

```
case AGENT_CONNECTED:
    // Check connection every second, if lost move to disconnected state
    EXECUTE_EVERY_N_MS(1000, state = (RMW_RET_OK ==
rmw_uros_ping_agent(500, 1)) ? AGENT_CONNECTED : AGENT_DISCONNECTED;);
    if (state == AGENT_CONNECTED) {
        rcl_executor_spin_some(&executor, RCL_MS_TO_NS(1));
    }
    break;

case AGENT_DISCONNECTED:
    // Destroy entities and try reconnecting
    destroy_entities();
    state = WAITING_AGENT;
    break;

default:
    break;
}
}
```

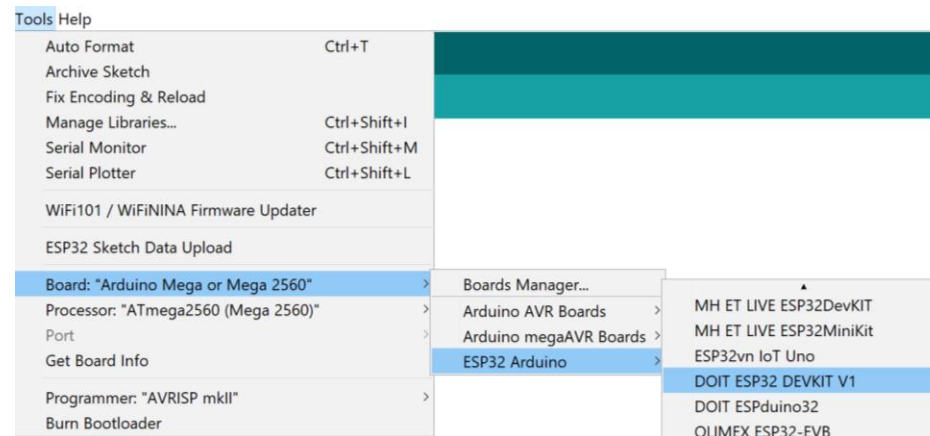


Activity



Compilation (Arduino IDE)

- Open Arduino IDE (previously configured).
- Select the board to be used Tools -> Board ESP32 (for Hackerboard is the same)
 - For ESP32 select ESP32 Arduino > DOIT ESP32 DEVKIT V1



- Compile the code using by clicking check mark button located on the upper left corner.



- The following message should be displayed:

Done compiling.

Sketch uses 9424 bytes (3%) of program storage space. Maximum is 253952 bytes.
Global variables use 1826 bytes (22%) of dynamic memory, leaving 6366 bytes for local variables. Maximum is 8192 bytes.

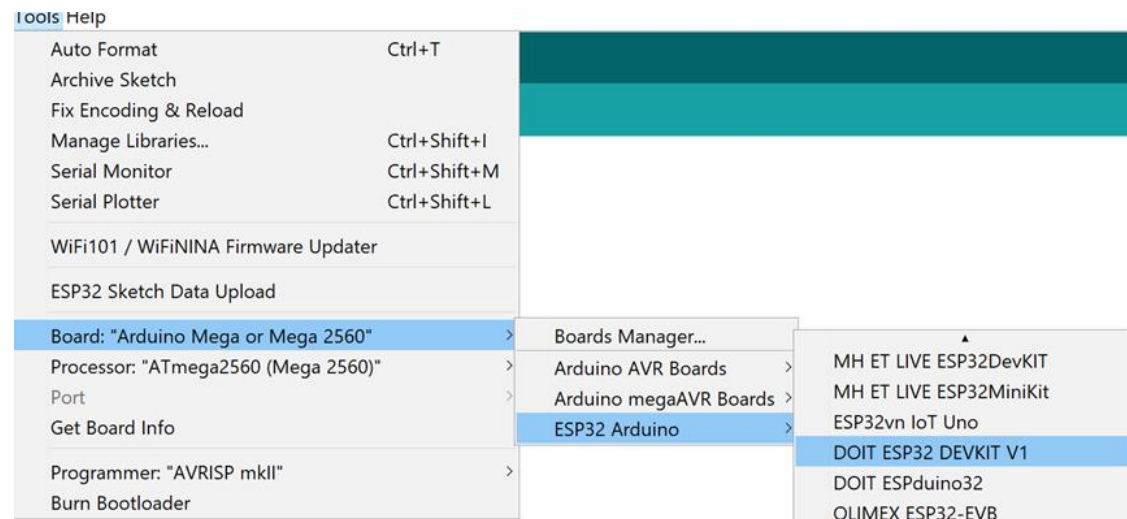
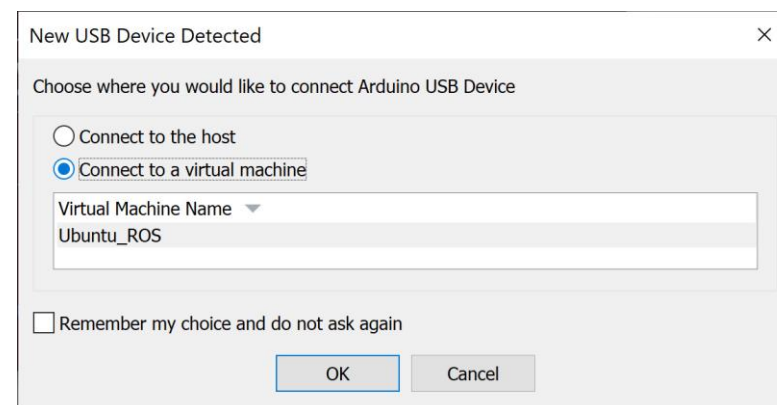


Activity



Uploading (Arduino IDE)

- Connect the Hackerboard or the ESP32 board
- Select the port to be used Tools>Port
 - If working on the VM, you must first select the option Connect to a virtual machine when automatically prompted (shown) and then select the port.
 - If in WSL, follow the steps on the presentation: "MCR2_Micro_ROS_Installation".
- Select the board to be used Tools -> ESP32 Arduino > DOIT ESP32 DEVKIT V1





Activity



Uploading (Arduino IDE)

- Upload the code using the arrow on the top left corner of the IDE.



- The following message should appear on the IDE

```
Done uploading.  
Sketch uses 1488 bytes (4%) of program storage space.  
Global variables use 198 bytes (9%) of dynamic memory
```

Running the node (Computer)

- Connect the board to the computer with ROS.
- (In Ubuntu) Make sure the port permissions are granted for the user (Skip this step if already performed).
 - In a new terminal type `cd /dev` to visualise the port designated by Ubuntu to the MCU. This port are usually called `/ttyACM0` or `/ttyUSB0`.

```
sudo chmod 666 /dev/ttyACM*  
sudo chmod 666 /dev/ttyUSB*
```



Activity 4



Test (Computer)

1. Connect to the AP “ESP-Access-Point”. Connect like a normal Wi-Fi Network.

- The network won't have Internet access (AP).

2. Make sure your IP Address is “10.16.1.2”.

1. Open a terminal and type

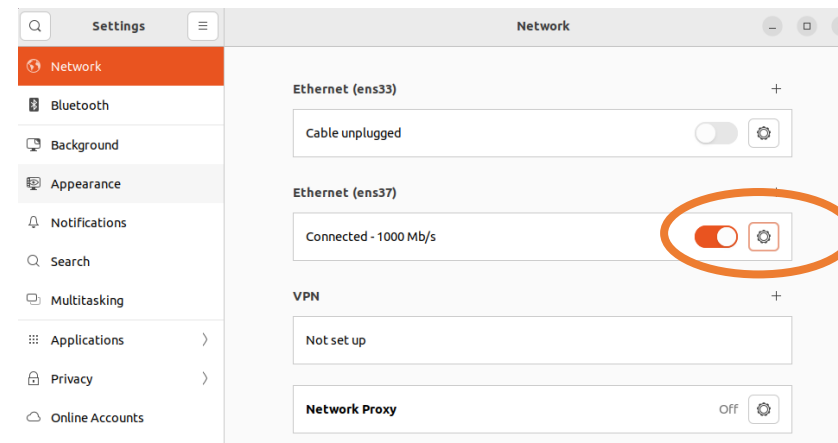
```
$ ifconfig
```

```
ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.16.1.2 netmask 255.255.255.0 broadcast 10.16.1.255
    inet6 fe80::b42:eb0:b61c:19e prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:33:89:1e txqueuelen 1000 (Ethernet)
    RX packets 10063 bytes 1008323 (1.0 MB)
    RX errors 0 dropped 13 overruns 0 frame 0
    TX packets 9746 bytes 1515880 (1.5 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

2. If not “10.16.1.2” go to configure IP section, else skip the section

Configure IP (Ubuntu)

1. Open Ubuntu settings>>Network
2. Click on the “gear” figure of the network adapter.



3. A pop-up window will open



Activity 4



Configure IP (Ubuntu)

4. Go to the IPV4 tab, select manual and type the following. Click Apply.

Cancel **Wired** Apply

Details Identity **IPv4** IPv6 Security

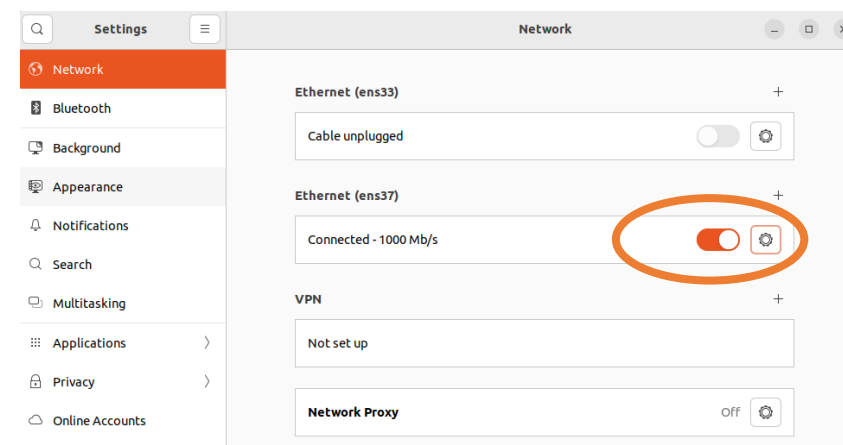
IPv4 Method

- ☐ Automatic (DHCP)
- ☒ Manual
- ☐ Link-Local Only
- ☐ Disable
- ☐ Shared to other computers

Addresses

Address	Netmask	Gateway	
10.16.1.2	255.255.255.0	10.16.1.1	

5. Reset the adapter by turning on and off the slider on the side of the “gear” figure.



6. Check your IP Address again using “ifconfig”

```
ens37: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.16.1.2 netmask 255.255.255.0 broadcast 10.16.1.255
    inet6 fe80::b432:eb0:b61c:19e prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:33:89:1e txqueuelen 1000 (Ethernet)
    RX packets 10063 bytes 1008323 (1.0 MB)
    RX errors 0 dropped 13 overruns 0 frame 0
    TX packets 9746 bytes 1515880 (1.5 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```




Activity 3



Test (Computer)

1. Open a terminal and type the following.

```
$ ros2 run micro_ros_agent micro_ros_agent udp4 --port 8888
```

```
mario@MarioPC:~/uros_ws$ ros2 run micro_ros_agent mi
cro_ros_agent serial --dev /dev/ttyUSB0
[1737636692.137695] info      | TermiosAgentLinux.cpp
| init                      | running...
| fd: 3
[1737636692.138467] info      | Root.cpp
set_verbose_level          | logger setup
verbose_level: 4
[1737636698.665805] info      | Root.cpp
create_client              | create
client_key: 0x0C9424D2, session_id: 0x81
[1737636698.666090] info      | SessionManager.hpp
establish_session          | session established
client_key: 0x0C9424D2, address: 0
[1737636698.702311] info      | ProxyClient.cpp
create_participant         | participant created
client_key: 0x0C9424D2, participant_id: 0x000(1)
[1737636698.721583] info      | ProxyClient.cpp
create_topic               | topic created
client_key: 0x0C9424D2, topic_id: 0x000(2), particip
ant_id: 0x000(1)
[1737636698.733573] info      | ProxyClient.cpp
create_publisher           | publisher created
client_key: 0x0C9424D2, publisher_id: 0x000(3), part
```

- Open another terminal and type the following.

```
$ ros2 topic list
```

```
mcr2@mcr2-virtual-machine:~$ ros2 topic list
/parameter_events
/rosout
/std_msgs/msg/Int32
```

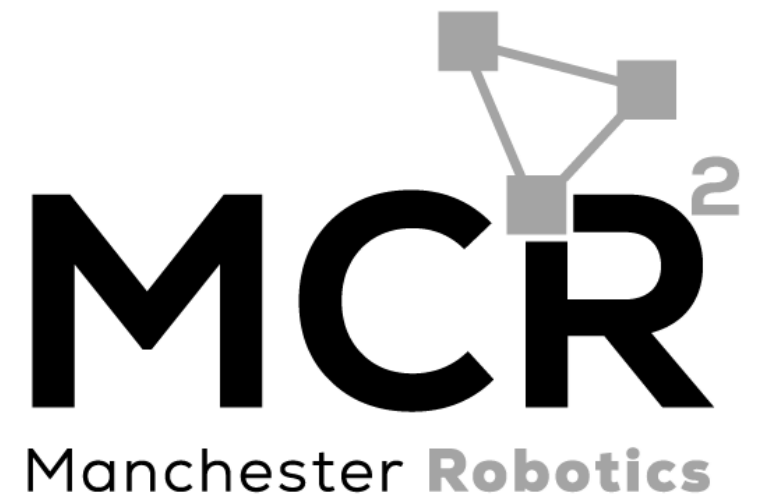
- Echo the topic “/button_state” . Press the Button!

```
$ ros2 topic echo /std_msgs_msg_Int32
```

```
mcr2@mcr2-virtual-machine:~$ ros2 topic echo /std_msgs_msg_Int32
data: 48
---
data: 49
---
data: 50
---
data: 51
```

Thank you

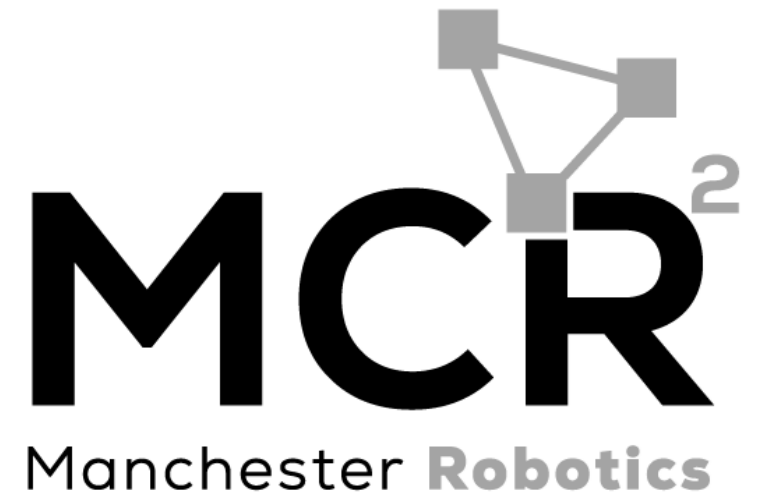
{Learn, Create, Innovate};



T&C

Terms and conditions

{Learn, Create, Innovate};





Terms and conditions



- *THE PIECES, IMAGES, VIDEOS, DOCUMENTATION, ETC. SHOWN HERE ARE FOR INFORMATIVE PURPOSES ONLY. THE DESIGN IS PROPRIETARY AND CONFIDENTIAL TO MANCHESTER ROBOTICS LTD. (MCR2). THE INFORMATION, CODE, SIMULATORS, DRAWINGS, VIDEOS PRESENTATIONS ETC. CONTAINED IN THIS PRESENTATION IS THE SOLE PROPERTY OF MANCHESTER ROBOTICS LTD. ANY REPRODUCTION, RESELL, REDISTRIBUTION OR USAGE IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF MANCHESTER ROBOTICS LTD. IS STRICTLY PROHIBITED.*
- *THIS PRESENTATION MAY CONTAIN LINKS TO OTHER WEBSITES OR CONTENT BELONGING TO OR ORIGINATING FROM THIRD PARTIES OR LINKS TO WEBSITES AND FEATURES IN BANNERS OR OTHER ADVERTISING. SUCH EXTERNAL LINKS ARE NOT INVESTIGATED, MONITORED, OR CHECKED FOR ACCURACY, ADEQUACY, VALIDITY, RELIABILITY, AVAILABILITY OR COMPLETENESS BY US.*
- *WE DO NOT WARRANT, ENDORSE, GUARANTEE, OR ASSUME RESPONSIBILITY FOR THE ACCURACY OR RELIABILITY OF ANY INFORMATION OFFERED BY THIRD-PARTY WEBSITES LINKED THROUGH THE SITE OR ANY WEBSITE OR FEATURE LINKED IN ANY BANNER OR OTHER ADVERTISING.*