# Subscribe topic

## 1. Experimental purpose

Learn ESP32-microROS components, access the ROS2 environment, and subscribe to int32 topics.

## 2. Hardware connection

As shown in the figure below, the microROS control board integrates the ESP32-S3-WROOM core module, which has its own wireless WiFi function. The ESP32-S3 core module needs to be connected to an antenna, and a type-C data cable needs to be connected to the computer and the microROS control board as Burn firmware function.
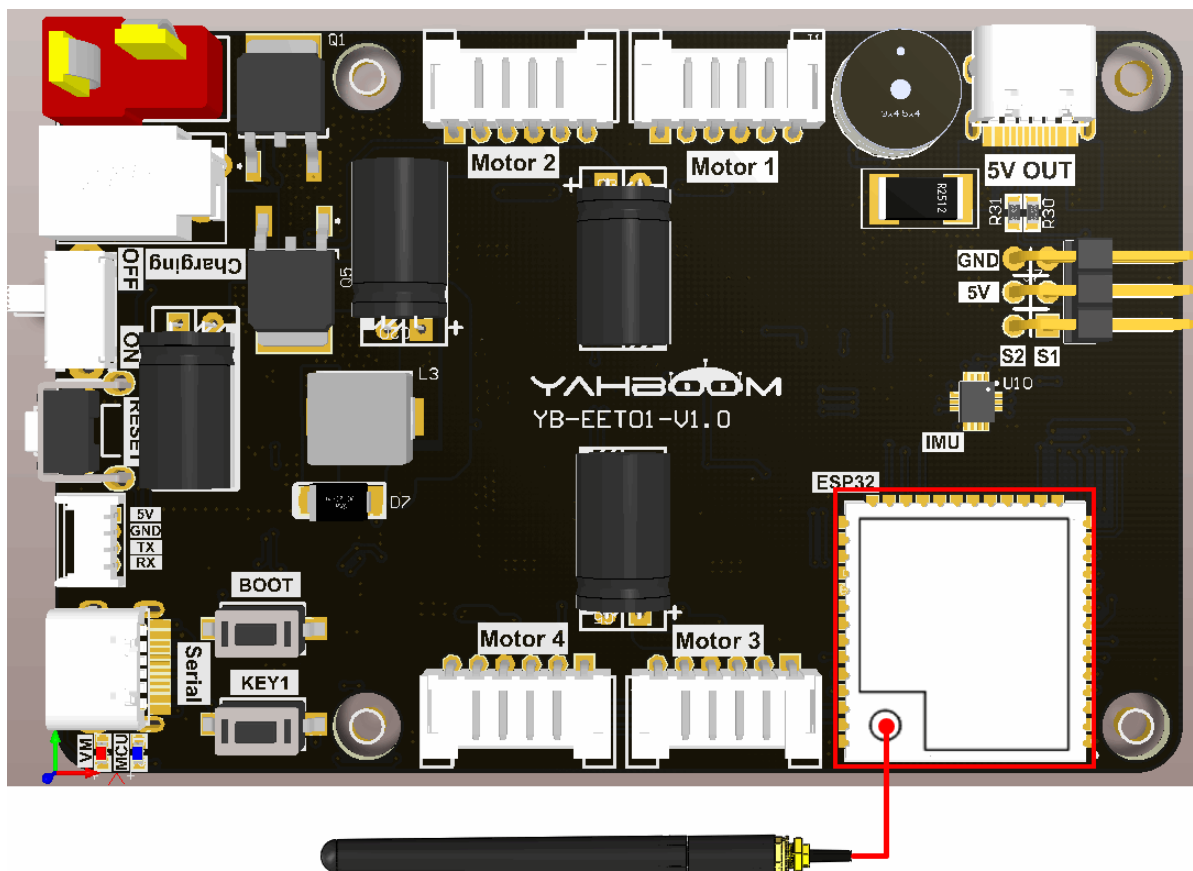
# 3. Core code analysis

The virtual machine path corresponding to the program source code is as follows

```
~/esp/Samples/microros_samples/subscriber
```

First, get the WiFi name and password to connect from the IDF configuration tool

```
#define ESP_WIFI_SSID       CONFIG_ESP_WIFI_SSID
#define ESP_WIFI_PASS       CONFIG_ESP_WIFI_PASSWORD
#define ESP_MAXIMUM_RETRY   CONFIG_ESP_MAXIMUM_RETRY
```

The uros_network_interface_initialize function will connect to WiFi hotspots based on the WiFi configuration in IDF.

```
ESP_ERROR_CHECK(uros_network_interface_initialize());
```

Then obtain ROS_NAMESPACE, ROS_DOMAIN_ID, ROS_AGENT_IP and ROS_AGENT_PORT from the IDF configuration tool.

```
#define ROS_NAMESPACE       CONFIG_MICRO_ROS_NAMESPACE
#define ROS_DOMAIN_ID       CONFIG_MICRO_ROS_DOMAIN_ID
#define ROS_AGENT_IP        CONFIG_MICRO_ROS_AGENT_IP
#define ROS_AGENT_PORT      CONFIG_MICRO_ROS_AGENT_PThen obtain ROS_NAMESPACE,
ROS_DOMAIN_ID, ROS_AGENT_IP and ROS_AGENT_PORT from the IDF configuration
tool.ORT
```

Initialize the configuration of microROS, in which ROS_DOMAIN_ID, ROS_AGENT_IP and ROS_AGENT_PORT are modified in the IDF configuration tool according to actual needs.

```
    rcl_allocator_t allocator = rcl_get_default_allocator();
    rclc_support_t support;

    // 创建rcl初始化选项
    // Create init_options.
    rcl_init_options_t init_options = rcl_get_zero_initialized_init_options();
    RCCHECK(rcl_init_options_init(&init_options, allocator));
    // 修改ROS域ID
    // change ros domain id
    RCCHECK(rcl_init_options_set_domain_id(&init_options, ROS_DOMAIN_ID));

    // 初始化rmw选项
    // Initialize the rmw options
    rmw_init_options_t *rmw_options =
rcl_init_options_get_rmw_init_options(&init_options);

    // 设置静态代理IP和端口
    // Setup static agent IP and port
    RCCHECK(rmw_uros_options_set_udp_address(ROS_AGENT_IP, ROS_AGENT_PORT,
rmw_options));
```

Try to connect to the proxy. If the connection is successful, go to the next step. If the connection to the proxy is unsuccessful, you will always be connected.

```
    while (1)
    {
        ESP_LOGI(TAG, "Connecting agent: %s:%s", ROS_AGENT_IP, ROS_AGENT_PORT);
        state_agent = rclc_support_init_with_options(&support, 0, NULL,
&init_options, &allocator);
        if (state_agent == ESP_OK)
        {
            ESP_LOGI(TAG, "Connected agent: %s:%s", ROS_AGENT_IP,
ROS_AGENT_PORT);
            break;
        }
        vTaskDelay(pdMS_TO_TICKS(500));
    }
```

Create the node "esp32_subscriber", in which ROS_NAMESPACE is empty by default and can be modified in the IDF configuration tool according to actual conditions.

```
    rcl_node_t node;
    RCCHECK(rclc_node_init_default(&node, "esp32_subscriber", ROS_NAMESPACE,
&support));
```

To create a subscriber "subscriber", you need to specify the ROS topic information as std_msgs/msg/Int32 type.

```
    RCCHECK(rclc_subscription_init_default(
        &subscriber,
        &node,
        ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
        "subscriber"));
```

Add subscribers to the executor, where the handle_num parameter of the executor represents the number added to the executor.

```
    rclc_executor_t executor;
    int handle_num = 1;
    RCCHECK(rclc_executor_init(&executor, &support.context, handle_num,
&allocator));

    RCCHECK(rclc_executor_add_subscription(&executor, &subscriber, &recv_msg,
&subscription_callback, ON_NEW_DATA));
```

When microros subscribers receive topic data, the subscription_callback callback function is triggered and the received data is printed out.

```
void subscription_callback(const void * msgin)
{
    const std_msgs__msg__Int32 * msg = (const std_msgs__msg__Int32 *)msgin;
    printf("Received: %d\n",  (int)  msg->data);
}
```

Call rclc_executor_spin_some in the loop to make microros work normally.

```
    while (1)
    {
        rclc_executor_spin_some(&executor, RCL_MS_TO_NS(100));
        usleep(1000);
    }
```

# 4. Compile and download the burning firmware

Use a Type-C data cable to connect the virtual machine/computer and the microROS control board. If the system pops up, choose to connect to the virtual machine.

Activate the ESP-IDF development environment. Note that every time you open a new terminal, you need to activate the ESP-IDF development environment before compiling the firmware.

```
source ~/esp/esp-idf/export.sh
```
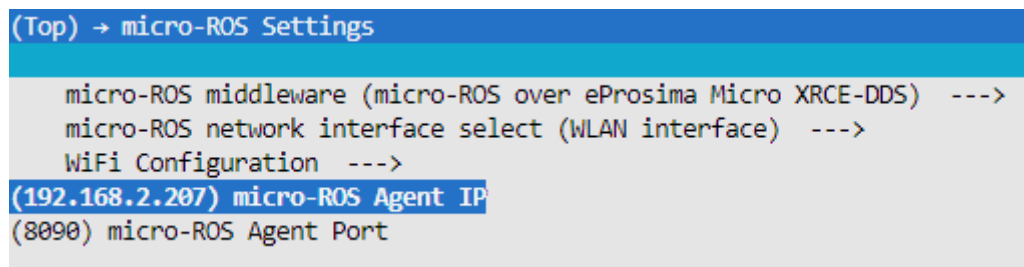
Enter the project directory

```
cd ~/esp/Samples/microros_samples/subscriber
```
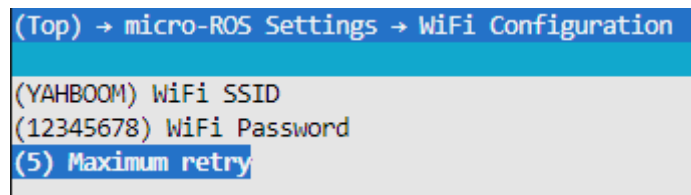
Open the ESP-IDF configuration tool.

```
idf.py menuconfig
```

Open micro-ROS Settings, fill in the IP address of the agent host in micro-ROS Agent IP, and fill in the port number of the agent host in micro-ROS Agent Port.



Open micro-ROS Settings->WiFi Configuration in sequence, and fill in your own WiFi name and password in the WiFi SSID and WiFi Password fields.



Open the micro-ROS example-app settings. The Ros domain id of the micro-ROS defaults to 20. If multiple users are using it at the same time in the LAN, the parameters can be modified to avoid conflicts. Ros namespace of the micro-ROS is empty by default and does not need to be modified under normal circumstances. If non-empty characters (within 10 characters) are modified, the namespace parameter will be added before the node and topic.

After modification, press S to save, and then press Q to exit the configuration tool.

Compile, burn, and open the serial port simulator.

```
idf.py build flash monitor
```

If you need to exit the serial port simulator, press **Ctrl+]**

## 5. Experimental results

After powering on, ESP32 tries to connect to the WiFi hotspot, and then tries to connect to the proxy IP and port.

If the agent is not turned on in the virtual machine/computer terminal, please enter the following command to turn on the agent. If the agent is already started, there is no need to start the agent again.

```
docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privileged --net=host
microros/micro-ros-agent:humble udp4 --port 8090 -v4
```



After the connection is successful, a node and a subscriber are created.



At this time, you can open another terminal in the virtual machine/computer and view the /esp32_publisher node.

```
ros2 node list
ros2 node info /esp32_subscriber
```

```
                         :~$ ros2 node info /esp32_subscriber
/esp32_subscriber
  Subscribers:
    /subscriber: std_msgs/msg/Int32
  Publishers:

  Service Servers:

  Service Clients:

  Action Servers:

  Action Clients:
```

Publish information with int data 123 to the /subscriber topic

```
ros2 topic pub /subscriber std_msgs/msg/Int32 "data: 123"
```

Press Ctrl+C to end the command.

```
                      :~$ ros2 topic pub /subscriber std_msgs/msg/Int32 "data: 123"
publisher: beginning loop
publishing #1: std_msgs.msg.Int32(data=123)

publishing #2: std_msgs.msg.Int32(data=123)

publishing #3: std_msgs.msg.Int32(data=123)

publishing #4: std_msgs.msg.Int32(data=123)
```

On the serial port simulator, you can see that the printed information is 123, indicating that the subscription is successful.

```
I (5582) SUBSCRIBER_MAIN: Connecting agent: 192.168.2.207:8090
I (5590) main_task: Returned from app_main()
I (5634) SUBSCRIBER_MAIN: Connected agent: 192.168.2.207:8090
Received: 123
Received: 123
Received: 123
Received: 123
```