

Subscribe buzzer topics

Subscribe buzzer topics

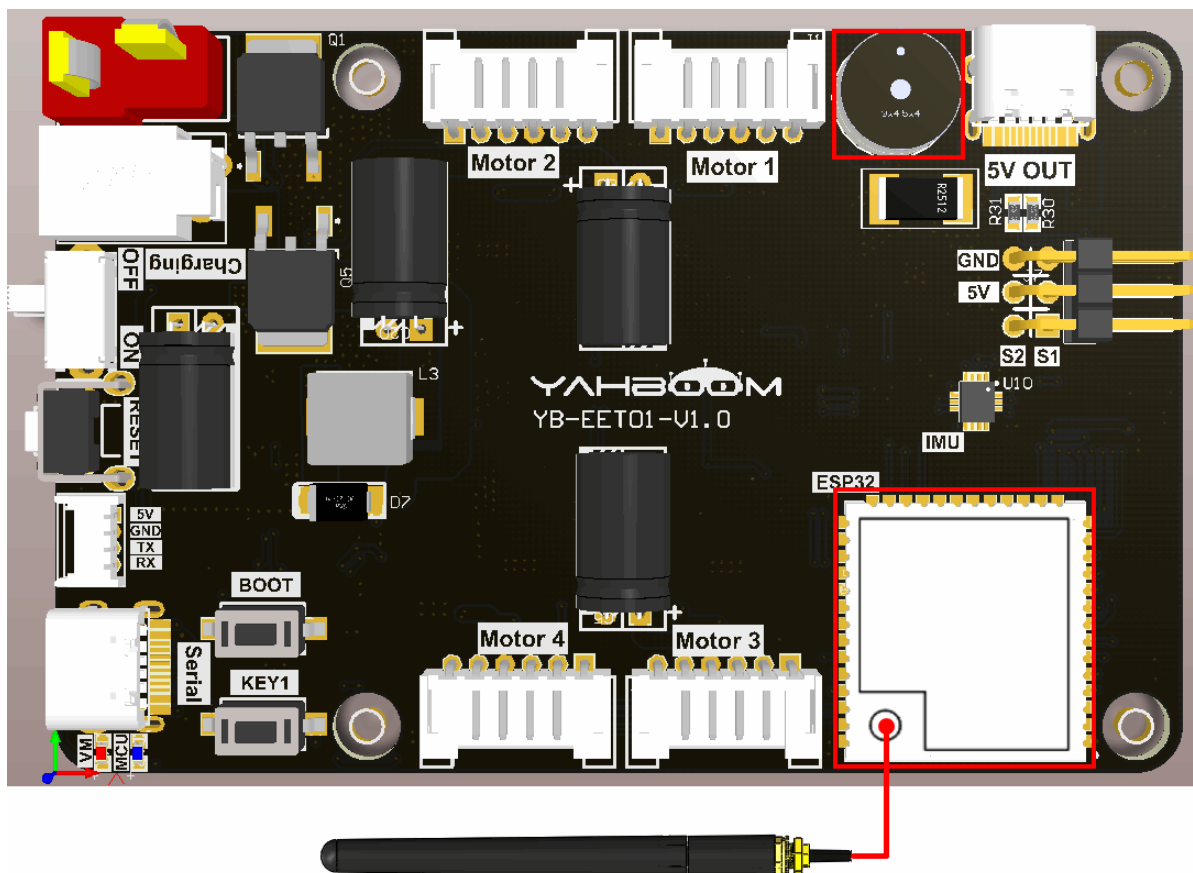
1. Experimental purpose
2. Hardware connection
3. Core code analysis
4. Compile, download and flash firmware
5. Experimental results

1. Experimental purpose

Learn ESP32-microROS components, access the ROS2 environment, and subscribe to the topic of controlling the buzzer switch.

2. Hardware connection

As shown in the figure below, the microROS control board integrates an active buzzer and 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. The computer and microROS control board function as firmware burning.

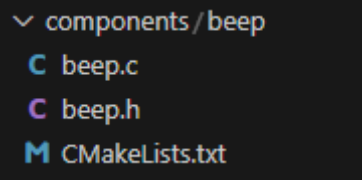


3. Core code analysis

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

```
~/esp/Samples/microros_samples/beep_subscriber
```

Since an active buzzer is used this time, and the components of the active buzzer have been made in the previous routine, the components of the active buzzer need to be copied to the components directory of the project. Call Beep_Init at the beginning of the program to initialize the buzzer.



```
components/beep
├── beep.c
├── beep.h
└── CMakeLists.txt
```

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_PORT
```

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();
RCHECK(rcl_init_options_init(&init_options, allocator));
// 修改ROS域ID
// change ros domain id
RCHECK(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
RCHECK(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 in the connected state.

```
while (1)
{
    ESP_LOGI(TAG, "Connecting agent: %s:%s", ROS_AGENT_IP, ROS_AGENT_PORT);
    state_agent = rcl_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 "beep_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;
RCHECK(rcl_node_init_default(&node, "beep_subscriber", ROS_NAMESPACE,
&support));
```

To create subscriber "beep", you need to specify the ROS topic information as std_msgs/msg/UInt16 type.

```
RCHECK(rcl_subscription_init_default(
    &subscriber,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, UInt16),
    "beep"));
```

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

```
rcl_executor_t executor;
int handle_num = 1;
RCHECK(rcl_executor_init(&executor, &support.context, handle_num,
&allocator));

RCHECK(rcl_executor_add_subscription(&executor, &subscriber, &msg_beep,
&beep_callback, ON_NEW_DATA));
```

When microros subscribers receive topic data, the beep_callback callback function is triggered and the active buzzer is controlled based on the received value.

```
void beep_callback(const void * msgin)
{
    const std_msgs__msg__UInt16 * msg = (const std_msgs__msg__UInt16 *)msgin;
    printf("Beep: %d\n", (int) msg->data);
    // 根据接收到的数据控制蜂鸣器的状态
    // Control the state of the buzzer based on the received data
    Beep_On_Time(msg->data);
}
```

Call `rcl_executor_spin_some` in the loop to make microros work normally.

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

4. Compile, download and flash 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
```

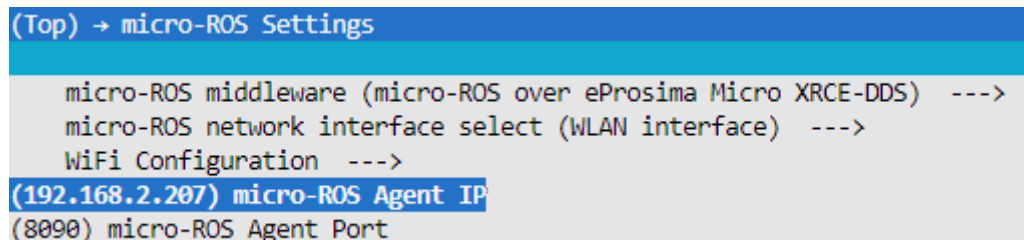
Enter the project directory

```
cd ~/esp/Samples/microros_samples/beep_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.



The screenshot shows the 'micro-ROS Settings' menu in the ESP-IDF configuration tool. The menu is expanded, showing options for 'micro-ROS middleware (micro-ROS over eProsima Micro XRCE-DDS)', 'micro-ROS network interface select (WLAN interface)', and 'WiFi Configuration'. The 'WiFi Configuration' option is selected, and the 'micro-ROS Agent IP' is set to '192.168.2.207' and the 'micro-ROS Agent Port' is set to '8090'.

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.

```
(Top) → micro-ROS Settings → WiFi Configuration
(YAHBOOM) WiFi SSID
(12345678) WiFi Password
(5) Maximum retry
```

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.

```
(Top) → micro-ROS example-app settings
(16000) Stack the micro-ROS app (Bytes)
(5) Priority of the micro-ROS app
(20) Ros domain id of the micro-ROS
() Ros namespace of the micro-ROS
```

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

Compile, flash, and open the serial port simulator.

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

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

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
```

```
~$ docker run -it --rm -v /dev:/dev -v /dev/shm:/dev/shm --privileged --net=host microros/micro-ros-agent:humble udp4 --port 8090 -v4
[1705475406.254095] info | UDPv4AgentLinux.cpp | init |
running... | port: 8090
[1705475406.254622] info | Root.cpp | set_verbose_level | 1
logger setup | verbose_level: 4
```

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

```
I (2051) MAIN: Connecting agent: 192.168.2.207:8090
I (2059) main_task: Returned from app_main()
I (2070) MAIN: Connected agent: 192.168.2.207:8090
```

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

```
ros2 node list
ros2 node info /beep_subscriber
```

```
~$ ros2 node list
/beep_subscriber
~$ ros2 node info /beep_subscriber
/beep_subscriber
Subscribers:
  /beep: std_msgs/msg/UInt16
Publishers:

Service Servers:

Service Clients:

Action Servers:

Action Clients:
```

Publish data to the /beep topic and control the buzzer to sound continuously.

```
ros2 topic pub --once /beep std_msgs/msg/UInt16 "data: 1"
```

Publish data to the /beep topic and control the buzzer to turn off.

```
ros2 topic pub --once /beep std_msgs/msg/UInt16 "data: 0"
```

Publish data to the /beep topic and control the buzzer to sound for 300 milliseconds and then turn off automatically.

```
ros2 topic pub --once /beep std_msgs/msg/UInt16 "data: 300"
```

```
~$ ros2 topic pub --once /beep std_msgs/msg/UInt16 "data: 1"
publisher: beginning loop
publishing #1: std_msgs.msg.UInt16(data=1)

~$ ros2 topic pub --once /beep std_msgs/msg/UInt16 "data: 0"
publisher: beginning loop
publishing #1: std_msgs.msg.UInt16(data=0)

~$ ros2 topic pub --once /beep std_msgs/msg/UInt16 "data: 300"
Waiting for at least 1 matching subscription(s)...
publisher: beginning loop
publishing #1: std_msgs.msg.UInt16(data=300)
```

You can see the printed beep information on the serial port simulator, indicating that the subscription is successful.

```
Beep: 1
Beep: 0
Beep: 300
□
```

