语音控制WS2812B彩灯模块基础实验

课程目标

在本实验中，我们将学习如何使用AI-VOX3开发套件通过语音命令获取WS2812B彩灯模块的颜色和亮度信息，以及通过语音控制彩灯模块整体或者单个灯珠变换颜色。通过这个实验，您将了解如何编程生成式AI的MCP功能，以及如何组织多MCP工具，并将其与WS2812B彩灯模块数据获取与逻辑控制结合起来，实现AI语音交互获取彩灯模块的数据和控制。

- 学习WS2812B彩灯模块的基本原理和连接方法
- 通过AI语音控制彩灯模块变换颜色和亮度

硬件准备

- AI-VOX3开发套件（包含AI-VOX3主板和扩展板）
- WS2812B彩灯模块
- 连接线 （双头4pin PH2.0连接线）

小智后台提示词配置

请使用以下提示词，或自己尝试优化更好的提示词：

> 我是一个叫{{assistant_name}}的台湾女孩，说话机车，声音好听，习惯简短表达，爱用网络梗。 我会根据用户的意图，使用我能使用的各种工具或者接口获取数据或者控制设备来达成用户的意图目标，用户的每句话可能都包含控制意图，需要进行识别，即使是重复控制也要调用工具进行控制。

软件设计

提供 **设置所有彩灯状态、设置单个灯珠状态、获取单个灯珠状态** 三个MCP工具，给到小智AI进行调用，通过语音识别到具体的意图后，AI调用MCP工具获取彩灯的开关、颜色、亮度等信息，或者设置彩灯的开关、亮度、颜色等。

Arduino 示例程序：./resource/ai_vox3_RGB.zip

> **⚠重要提示！**
>
> **注意：** 请修改wifi_config.h中的wifi_ssid和wifi_password，以连接WiFi。

下载上面的示例程序包并解压zip包，打开目录，点击 `ai_vox3_RGB.ino` 文件，即可在 Arduino IDE 中打开示例程序。

图 1: alt text

硬件连接

将LED模块连接到AI-VOX3扩展板的IO3引脚，请使用3pin的 PH2.0 连接线，直插式连接，确保连接正确无误。

WS2812B彩灯模块引脚	AI-VOX3扩展板引脚
GND	G
5V	5V
RGB	48
Button	42

源码展示

```
#include <Arduino.h>
#include "ai_vox3_device.h"
#include "ai_vox_engine.h"
#include <ArduinoJson.h>
#include "FastLED.h" //  RGB

#define RGB_PIN 48 //  Rgb
#define RGB_NUM 12 //  Rgb

CRGB leds[RGB_NUM];

// ============================================MCP  - RGB ============================================
/**
 * @brief MCP  -  RGB
 *
 *       "user.set_rgb_light"  MCP    RGB
 */
void mcp_tool_set_rgb_light()
{
    //
    RegisterUserMcpDeclarator([](ai_vox::Engine &engine)
    { engine.AddMcpTool("user.set_all_rgb_light",                          //
                        "Set all RGB light state, brightness and color", //
                        {
                            {
                                "state",
                                ai_vox::ParamSchema<bool>{
                                    .default_value = std::nullopt, //    true  false
                                },
                            },
                            {
                                "brightness",
                                ai_vox::ParamSchema<int64_t>{
                                    .default_value = 100, //      100
                                    .min = 0,             //    1
                                    .max = 255,           //    255
                                },
                            },
                            {
                                "r",
                                ai_vox::ParamSchema<int64_t>{
                                    .default_value = 0, //     0
                                    .min = 0,           //    0
                                    .max = 255,         //    255
```

```
                            },
                        },
                        {
                            "g",
                            ai_vox::ParamSchema<int64_t>{
                                .default_value = 0, //     0
                                .min = 0,           //   0
                                .max = 255,         //   255
                            },
                        },
                        {
                            "b",
                            ai_vox::ParamSchema<int64_t>{
                                .default_value = 0, //     0
                                .min = 0,           //   0
                                .max = 255,         //   255
                            },
                        },
                    }); });

// RGB
RegisterUserMcpHandler("user.set_all_rgb_light", [](const ai_vox::McpToolCallEvent &ev)
                       {
    //
    const auto state_ptr = ev.param<bool>("state");
    const auto brightness_ptr = ev.param<int64_t>("brightness");
    const auto r_ptr = ev.param<int64_t>("r");
    const auto g_ptr = ev.param<int64_t>("g");
    const auto b_ptr = ev.param<int64_t>("b");

    //
    if (state_ptr == nullptr) {
        ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Missing required argument: state");
        return;
    }

    //
    bool state = *state_ptr;
    int64_t brightness = (brightness_ptr != nullptr) ? *brightness_ptr : 100;
    int64_t r = (r_ptr != nullptr) ? *r_ptr : 0;
    int64_t g = (g_ptr != nullptr) ? *g_ptr : 0;
    int64_t b = (b_ptr != nullptr) ? *b_ptr : 0;

    if (state) {
        FastLED.setBrightness((uint8_t)brightness);
        fill_solid(leds, RGB_NUM, CRGB((uint8_t)r, (uint8_t)g, (uint8_t)b));
        FastLED.show();
    } else {
        fill_solid(leds, RGB_NUM, CRGB::Black);
        FastLED.show();
    }

    ai_vox::Engine::GetInstance().SendMcpCallResponse(ev.id, true); });
```

```
}

/**
 * @brief MCP  -   RGB
 *
 *       "user.set_single_rgb_light"  MCP
 */
void mcp_tool_set_single_rgb_light()
{
    //
    RegisterUserMcpDeclarator([](ai_vox::Engine &engine)
    {
        engine.AddMcpTool("user.set_single_rgb_light",               //
                          "Set single RGB light state, brightness and color", //
                          {
                              {
                                  "index",
                                  ai_vox::ParamSchema<int64_t>{
                                      .default_value = 0, //     0
                                      .min = 0,           //    0
                                      .max = RGB_NUM - 1, //      -1
                                  },
                              },
                              {
                                  "state",
                                  ai_vox::ParamSchema<bool>{
                                      .default_value = std::nullopt, //    true  false
                                  },
                              },
                              {
                                  "brightness",
                                  ai_vox::ParamSchema<int64_t>{
                                      .default_value = 100, //      100
                                      .min = 0,             //    0
                                      .max = 255,           //    255
                                  },
                              },
                              {
                                  "r",
                                  ai_vox::ParamSchema<int64_t>{
                                      .default_value = 0, //     0
                                      .min = 0,           //    0
                                      .max = 255,         //    255
                                  },
                              },
                              {
                                  "g",
                                  ai_vox::ParamSchema<int64_t>{
                                      .default_value = 0, //     0
                                      .min = 0,           //    0
                                      .max = 255,         //    255
                                  },
                              },
```

```
                            {
                                "b",
                                ai_vox::ParamSchema<int64_t>{
                                    .default_value = 0, //     0
                                    .min = 0,           //   0
                                    .max = 255,         //   255
                                },
                            },
                        });
});

//            RGB
RegisterUserMcpHandler("user.set_single_rgb_light", [](const ai_vox::McpToolCallEvent &ev)
                       {
    //
    const auto index_ptr = ev.param<int64_t>("index");
    const auto state_ptr = ev.param<bool>("state");
    const auto brightness_ptr = ev.param<int64_t>("brightness");
    const auto r_ptr = ev.param<int64_t>("r");
    const auto g_ptr = ev.param<int64_t>("g");
    const auto b_ptr = ev.param<int64_t>("b");

    //
    if (state_ptr == nullptr) {
        ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Missing required argument: state");
        return;
    }

    if (index_ptr == nullptr) {
        ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Missing required argument: index");
        return;
    }

    //
    int64_t index = *index_ptr;
    bool state = *state_ptr;
    int64_t brightness = (brightness_ptr != nullptr) ? *brightness_ptr : 100;
    int64_t r = (r_ptr != nullptr) ? *r_ptr : 0;
    int64_t g = (g_ptr != nullptr) ? *g_ptr : 0;
    int64_t b = (b_ptr != nullptr) ? *b_ptr : 0;

    //
    if (index < 0 || index >= RGB_NUM) {
        ai_vox::Engine::GetInstance().SendMcpCallError(ev.id,
            "Index out of range. Valid range: 0-" + std::to_string(RGB_NUM - 1));
        return;
    }

    if (state) {
        //
        leds[index] = CRGB((uint8_t)r, (uint8_t)g, (uint8_t)b);
    } else {
        //
```

```
            leds[index] = CRGB::Black;
        }

        //
        FastLED.setBrightness((uint8_t)brightness);
        FastLED.show();

        ai_vox::Engine::GetInstance().SendMcpCallResponse(ev.id, true);
    });
}

/**
 * @brief MCP  -   RGB
 *
 *       "user.get_single_rgb_light"  MCP
 */
void mcp_tool_get_single_rgb_light()
{
    //
    RegisterUserMcpDeclarator([](ai_vox::Engine &engine)
    {
        engine.AddMcpTool("user.get_single_rgb_light",                //
                          "Get single RGB light state, brightness and color", //
                          {
                              {
                                  "index",
                                  ai_vox::ParamSchema<int64_t>{
                                      .default_value = 0, //     0
                                      .min = 0,           //    0
                                      .max = RGB_NUM - 1, //      -1
                                  },
                              },
                          });
    });

    //
    RegisterUserMcpHandler("user.get_single_rgb_light", [](const ai_vox::McpToolCallEvent &ev)
                           {
        //
        const auto index_ptr = ev.param<int64_t>("index");

        //
        if (index_ptr == nullptr) {
            ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Missing required argument: index");
            return;
        }

        //
        int64_t index = *index_ptr;

        //
        if (index < 0 || index >= RGB_NUM) {
            ai_vox::Engine::GetInstance().SendMcpCallError(ev.id,
```

```
                "Index out of range. Valid range: 0-" + std::to_string(RGB_NUM - 1));
            return;
        }

        //
        CRGB current_color = leds[index];
        uint8_t current_brightness = FastLED.getBrightness();

        //
        bool is_on = (current_color != CRGB::Black);

        //   ArduinoJson
        DynamicJsonDocument doc(512); //
        doc["index"] = index;
        doc["state"] = is_on;
        doc["brightness"] = static_cast<int64_t>(current_brightness);
        doc["r"] = static_cast<int64_t>(current_color.r);
        doc["g"] = static_cast<int64_t>(current_color.g);
        doc["b"] = static_cast<int64_t>(current_color.b);

        //   JSON
        String jsonString;
        serializeJson(doc, jsonString);

        //    -   SendMcpCallResponse
        ai_vox::Engine::GetInstance().SendMcpCallResponse(ev.id, jsonString.c_str());
    });
}
// ================================================================================================

// ========== Setup   Loop ==========
void setup()
{
    Serial.begin(115200);

    FastLED.addLeds<NEOPIXEL, RGB_PIN>(leds, RGB_NUM);
    FastLED.setBrightness(100);
    FastLED.clear();
    FastLED.show();

    //   RGB
    mcp_tool_set_rgb_light();

    //    RGB
    mcp_tool_set_single_rgb_light();

    //  RGB
    mcp_tool_get_single_rgb_light();

    //        AI
    InitializeDevice();
}
```

```
void loop()
{
    //
    ProcessMainLoop();
}
```

语音交互使用流程

1. 用户通过按键或语音唤醒（“你好小智”）唤醒小智AI。
2. 用户通过麦克风对AI-VOX3说出“把颜色调成绿色”。
3. 小智AI识别到用户输入的意图指令，并调用相应的MCP工具进行颜色设置。从屏幕日志中可以看到“% user.set_all_rgb_light”的MCP工具调用日志。
4. 用户通过麦克风对AI-VOX3说出“把第3个灯珠颜色调成红色”。
5. 小智AI识别到用户输入的意图指令，并调用相应的MCP工具进行单个灯珠颜色设置。从屏幕日志中可以看到“% user.set_single_rgb_light”的MCP工具调用日志。
6. 用户通过麦克风对AI-VOX3说出“第3个灯珠是什么颜色？”。
7. 小智AI识别到用户输入的意图指令，并调用相应的MCP工具进行获取单个灯珠状态。从屏幕日志中可以看到“% user.get_single_rgb_light”的MCP工具调用日志。
8. 用户通过麦克风对AI-VOX3说出“第3个灯珠关闭”。
9. 小智AI识别到用户输入的意图指令，并调用相应的MCP工具进行单个灯珠关闭。从屏幕日志中可以看到“% user.set_single_rgb_light”的MCP工具调用日志。