基于有源蜂鸣器AI闹钟基础实验

需求分析

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控制有源蜂鸣器，设置AI闹钟。通过这个实验，您将了解如何编程生成式AI的MCP功能，并将有源蜂鸣器模块控制逻辑和AI闹钟结合起来，实现智能语音交互控制有源蜂鸣器报警功能及AI闹钟功能。

- 学习有源蜂鸣器模块的基本使用方法
- 使用AI-VOX3 的AI框架，编写MCP工具实现蜂鸣器报警控制功能
- 使用AI-VOX3 的AI框架，编写MCP工具实现AI闹钟功能

硬件准备

- AI-VOX3开发套件（包含AI-VOX3主板和扩展板）
- 有源蜂鸣器模块
- 连接线 （双头3pin 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 **触发报警** MCP工具，给到小智AI进行调用，AI识别到设置报警的意图后，AI调用MCP工具设置触发或否关闭蜂鸣器。 提供 **设置AI闹钟** MCP工具，给到小智AI进行调用，AI识别到设置闹钟的意图后，AI调用MCP工具设置定时器。

Arduino 示例程序：./resource/ai_vox3_buzzer.zip

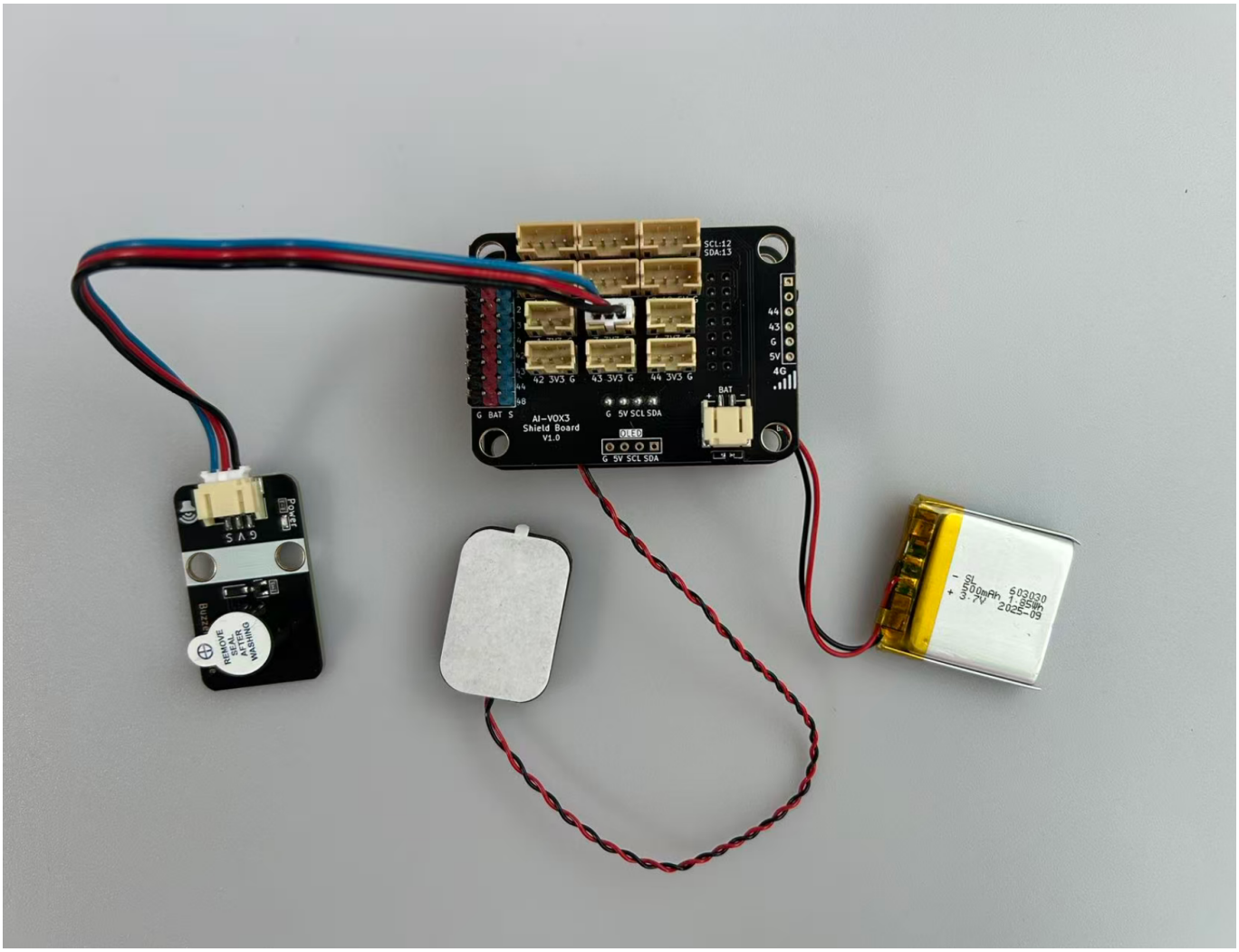
> ⚠**重要提示!**
>
> **注意：** 请修改wifi_config.h中的wifi_ssid和wifi_password，以连接WiFi。

下载上面的示例程序包并解压zip包，打开目录，点击 `ai_vox3_buzzer.ino` 文件，即可在 Arduino IDE 中打开示例程序。

硬件连接

将有源蜂鸣器模块连接到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。

有源蜂鸣器模块引脚	AI-VOX3扩展板引脚
G	3V3
V	G
S	3



源码展示

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

#define BUZZER_PIN 3   // 蜂鸣器引脚

// 全局变量用于存储定时器状态
struct AlarmTimer {
    bool active = false;
    unsigned long start_time = 0;
    unsigned long duration_ms = 0;
```

```
    int buzzer_state = 1;
    int64_t event_id = 0;
} current_alarm;

/**
 * @brief MCP工具 - 控制有源蜂鸣器报警
 *
 * 该函数注册一个名为 "user.buzzer.control" 的MCP工具，
 * 用于控制有源蜂鸣器的开关状态
 */
void mcp_tool_buzzer_control()
{
    // 注册工具声明器，定义工具的名称和描述
    RegisterUserMcpDeclarator([](ai_vox::Engine &engine)
                              {
                                  engine.AddMcpTool("user.buzzer.control",  // 工
具名称
                                                    "Control buzzer on/off", // 工
具描述
                                                    {
                                                        {"state",

ai_vox::ParamSchema<int64_t>{
                                                             .default_value =
std::nullopt, // 状态参数，默认值为空

                                                             .min = 0,
// 0表示关闭蜂鸣器

                                                             .max = 1,
// 1表示开启蜂鸣器
                                                         }}
                                                    }); // 需要传入state参数，0为关
闭，1为开启
                              });

    // 注册工具处理器，收到调用时，控制蜂鸣器
    RegisterUserMcpHandler("user.buzzer.control", [](const
ai_vox::McpToolCallEvent &ev)
                           {
        // 解析参数
        const auto state_ptr = ev.param<int64_t>("state");

        // 检查必需参数是否存在
        if (state_ptr == nullptr) {
            ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Missing
required argument: state (0=off, 1=on)");
            return;
        }

        // 获取参数值
        int64_t state = *state_ptr;

        // 参数验证
        if (state != 0 && state != 1) {
            ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "State must be 0
```

```
(off) or 1 (on)");
            return;
        }

        // 控制蜂鸣器
        digitalWrite(BUZZER_PIN, state);

        const char* state_str = (state == 1) ? "ON" : "OFF";
        printf("Buzzer turned %s (GPIO %d)\n", state_str, BUZZER_PIN);

        // 创建响应
        DynamicJsonDocument doc(256);
        doc["status"] = "success";
        doc["state"] = state;
        doc["gpio"] = BUZZER_PIN;

        // 将 JSON 文档转换为字符串
        String jsonString;
        serializeJson(doc, jsonString);

        // 发送响应
        ai_vox::Engine::GetInstance().SendMcpCallResponse(ev.id,
jsonString.c_str());
                            });
}

/**
 * @brief MCP工具 - 设置定时闹钟
 *
 * 该函数注册一个名为 "user.alarm.timer" 的MCP工具，
 * 用于设置倒计时，在指定时间后激活蜂鸣器报警
 */
void mcp_tool_alarm_timer()
{
    // 注册工具声明器，定义工具的名称和描述
    RegisterUserMcpDeclarator([](ai_vox::Engine &engine)
                              {
                                  engine.AddMcpTool("user.alarm.timer",     // 工
具名称
                                                    "Set a countdown timer that
triggers buzzer when time is up", // 工具描述
                                                    {
                                                        {"seconds",

ai_vox::ParamSchema<int64_t>{
                                                             .default_value =
std::nullopt, // 倒计时秒数，必需参数
                                                             .min = 1,
// 最少1秒
                                                             .max = 3600,
// 最多1小时
                                                         }},
                                                        {"buzzer_state",
```

```
ai_vox::ParamSchema<int64_t>{
                                                        .default_value = 1,
// 默认为开启蜂鸣器
                                                        .min = 0,
// 0表示关闭蜂鸣器
                                                        .max = 1,
// 1表示开启蜂鸣器
                                                    }}
                                                }); // 需要传入seconds参数（倒
计时秒数）和可选的buzzer_state参数
                                });

    // 注册工具处理器，接收调用时启动倒计时
    RegisterUserMcpHandler("user.alarm.timer", [](const ai_vox::McpToolCallEvent
&ev)
                           {
        // 解析参数
        const auto seconds_ptr = ev.param<int64_t>("seconds");
        const auto buzzer_state_ptr = ev.param<int64_t>("buzzer_state");

        // 检查必需参数是否存在
        if (seconds_ptr == nullptr) {
            ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Missing
required argument: seconds (countdown duration in seconds)");
            return;
        }

        // 获取参数值
        int64_t seconds = *seconds_ptr;
        int64_t buzzer_state = buzzer_state_ptr ? *buzzer_state_ptr : 1; // 默认为
1（开启）

        // 参数验证
        if (seconds < 1 || seconds > 3600) {
            ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Seconds must be
between 1 and 3600");
            return;
        }

        if (buzzer_state != 0 && buzzer_state != 1) {
            ai_vox::Engine::GetInstance().SendMcpCallError(ev.id, "Buzzer state
must be 0 (off) or 1 (on)");
            return;
        }

        // 设置新的闹钟
        current_alarm.active = true;
        current_alarm.start_time = millis();
        current_alarm.duration_ms = seconds * 1000;  // 转换为毫秒
        current_alarm.buzzer_state = buzzer_state;
        current_alarm.event_id = ev.id;

        printf("Starting alarm timer for %lld seconds\n", (long long)seconds);
```

```
        // 立即发送开始倒计时的确认响应
        DynamicJsonDocument doc(256);
        doc["status"] = "timer_started";
        doc["seconds"] = seconds;
        doc["buzzer_state"] = buzzer_state;

        String jsonString;
        serializeJson(doc, jsonString);

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

/**
 * @brief 检查并处理定时器到期事件
 */
void check_and_process_alarm() {
    if (current_alarm.active) {
        unsigned long elapsed_time = millis() - current_alarm.start_time;

        if (elapsed_time >= current_alarm.duration_ms) {
            // 闹钟时间到，触发蜂鸣器
            digitalWrite(BUZZER_PIN, current_alarm.buzzer_state);

            const char* state_str = (current_alarm.buzzer_state == 1) ? "ON" :
"OFF";
            printf("Alarm triggered! Buzzer turned %s (GPIO %d)\n", state_str,
BUZZER_PIN);

            // 准备响应数据
            DynamicJsonDocument response_doc(256);
            response_doc["status"] = "alarm_triggered";
            response_doc["buzzer_state"] = current_alarm.buzzer_state;
            response_doc["duration_seconds"] = current_alarm.duration_ms / 1000;

            String response_json;
            serializeJson(response_doc, response_json);

            // 发送异步响应

ai_vox::Engine::GetInstance().SendMcpCallResponse(current_alarm.event_id,
response_json.c_str());

            // 重置闹钟状态
            current_alarm.active = false;
        }
    }
}

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

```
    delay(500); // 等待串口初始化

    pinMode(BUZZER_PIN, OUTPUT);    // 初始化蜂鸣器引脚
    digitalWrite(BUZZER_PIN, LOW);  // 初始状态关闭蜂鸣器

    // 注册MCP工具 - 蜂鸣器控制功能
    mcp_tool_buzzer_control();

    // 注册MCP工具 - 定时闹钟功能
    mcp_tool_alarm_timer();

    // 初始化设备服务，包括硬件和AI引擎，必备步骤
    InitializeDevice();
}

void loop()
{
    // 检查并处理定时器
    check_and_process_alarm();

    // 处理设备服务主循环事件，  必备步骤
    ProcessMainLoop();
}
```

语音交互使用流程

> **注意：** 请先在小智AI后台，清空历史记忆，防止出现不同程序间记忆冲突的问题。

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“触发报警” 或 “关闭报警”。
3. 小智AI识别到用户输入的意图指令，并调用相应的MCP工具进行触发报警或关闭报警动作。从屏幕日志中可以看到“% user.buzzer.control”的MCP工具调用日志。
4. 用户通过麦克风对AI-VOX3说出“设置闹钟，10秒后提醒我”。
5. 小智AI识别到用户输入的意图指令，并调用相应的MCP工具进行设置闹钟动作。从屏幕日志中可以看到“% user.alarm.timer”的MCP工具调用日志，10秒后触发的闹钟会通过蜂鸣器发出提示声。