第一章 两个 UART 实验

1. 学习目的及目标

- 串口通信的原理
- 学习 ESP32 的 UART 功能的配置
- 掌握 UART 收发测试程序

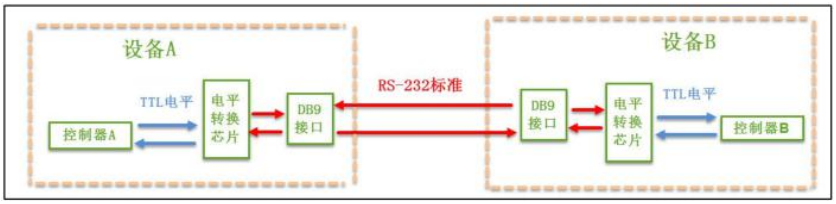
2. 串口通讯协议简介

串口通讯(Serial Communication)是一种设备间非常常用的串行通讯方式，因为它简单便捷，大部分电子设备都支持该通讯方式，电子工程师在调试设备时也经常使用该通讯方式输出调试信息，ESP32 自有一个串口用于程序下载和 log 打印，就是这个道理。

在计算机科学里，大部分复杂的问题都可以通过分层来简化。如芯片被分为内核层和片上外设；对于通讯协议，我们也以分层的方式来理解，最基本的是把它分为物理层和协议层。物理层规定通讯系统中具有机械、电子功能部分的特性，确保原始数据在物理媒体的传输。协议层主要规定通讯逻辑，统一收发双方的数据打包、解包标准。简单来说物理层规定我们用嘴巴还是用肢体来交流，协议层则规定我们用中文还是英文来交流。

2.1. 物理层

串口通讯的物理层有很多标准及变种，我们主要讲解 RS-232 标准 ，RS-232 标准主要规定了信号的用途、通讯接口以及信号的电平标准。使用 RS-232 标准的串口设备间常见的通讯结构如下。



在上面的通讯方式中，两个通讯设备的“DB9 接口”之间通过串口信号线建立起连接，串口信号线中使用“RS-232 标准”传输数据信号。由于 RS-232 电平标准的信号不能直接被控制器直接识别，所以这些信号会经过一个“电平转换芯片”转换成控制器能识别的“TTL 校准”的电平信号，才能实现通讯。

2.2. 协议层

串口通讯的数据包由发送设备通过自身的 TXD 接口传输到接收设备的 RXD 接口。在串口通讯的协议层中，规定了数据包的内容，它由启始位、主体数据、校验位以及停止位组成，通讯双方的数据包格式要约定一致才能正常收发数据，其组成如下：

- 波特率

本章中主要讲解的是串口异步通讯，异步通讯中由于没有时钟信号(如前面讲解的 DB9 接口中是没有时钟信号的)，所以两个通讯设备之间需要约定好波特率，即每个码元的长度，以便对信号进行解码，上图中用虚线分开的每一格就是代表一个码元。常见的波特率为 4800、9600、115200 等。

- 通讯的起始和停止信号

串口通讯的一个数据包从起始信号开始，直到停止信号结束。数据包的起始信号由一个逻辑 0 的数据位表示，而数据包的停止信号可由 0.5、1、1.5 或 2 个逻辑 1 的数据位表示，只要双方约定一致即可。

- 有效数据

在数据包的起始位之后紧接着的就是要传输的主体数据内容，也称为有效数据，有效数据的长度常被约定为 5、6、7 或 8 位长。

- 数据校验

在有效数据之后，有一个可选的数据校验位。由于数据通信相对更容易受到外部干扰导致传输数据出现偏差，可以在传输过程加上校验位来解决这个问题。校验方法有奇校验(odd)、偶校验(even)、0 校验(space)、1 校验(mark)以及无校验(noparity)。在无校验的情况下，数据包中不包含校验位。

3. 硬件设计及原理

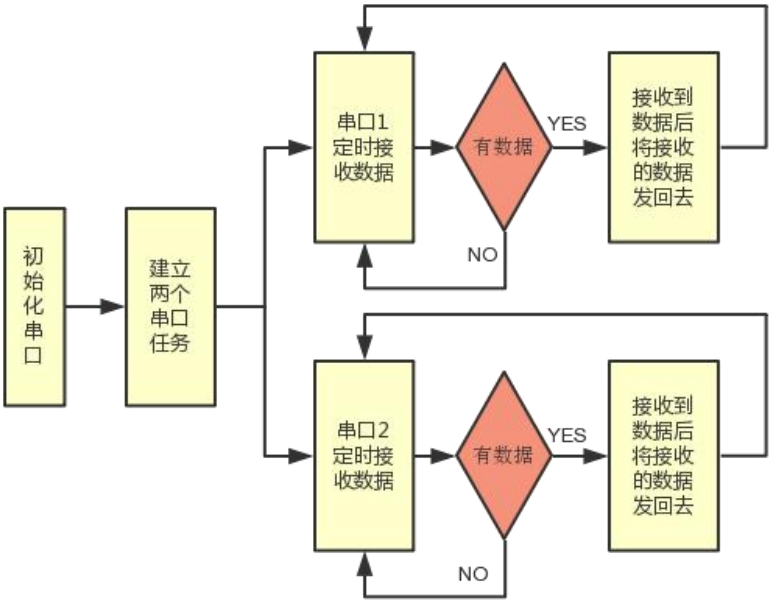
本实验板使用了 ESP32 的 UART1 和 UART2，下表是我们的程序 IO 的映射。

UART1	功能	映射 ESP32 的引脚
TXD	发送	IO5
RXD	接收	IO4
UART2	**功能**	**映射 ESP32 的引脚**
TXD	发送	IO12
RXD	接收	IO13

若您使用的实验板 UART 的连接方式或引脚不一样，只需根据我们的工程修改引脚即可，程序的控制原理相同。

4. 软件设计

4.1. 代码逻辑



4.2. ESP32 的 UART 接口介绍

➢ UART 配置函数：uart_param_config();

函数原型	esp_err_t uart_param_config (uart_port_t uart_num, const uart_config_t *uart_config)
函数功能	UART 配置函数
参数	[in] uart_num:串口号，取值 ● UART_NUM_0 = 0x0, /*串口 0，下载程序端口*/ ● UART_NUM_1 = 0x1, /*串口 1*/ ● UART_NUM_2 = 0x2,/*串口 2*/ [in] uart_config:串口参数配置 typedef struct { int baud_rate; /*波特率*/ uart_word_length_t data_bits; /*数据位*/ uart_parity_t parity; /*校验模式*/ uart_stop_bits_t stop_bits; /*停止位*/ uart_hw_flowcontrol_t flow_ctrl; /*硬件流控使能位*/ } uart_config_t;
返回值	ESP_OK：成功 ESP_ERR_INVALID_ARG ： 参数错误

➢ UART 的 IO 映射设置函数：uart_set_pin();

函数原型	esp_err_t uart_set_pin (uart_port_t uart_num, int tx_io_num, int rx_io_num, int rts_io_num, int cts_io_num)
函数功能	UART 的 IO 映射函数
参数	[in] uart_num:串口号，取值 [in] tx_io_num:发送引脚 [in] rx_io_num:接收引脚 [in] rts_io_num:rts 流控引脚 [in] cts_io_num:cts 流控引脚
返回值	ESP_OK：成功 ESP_ERR_INVALID_ARG ： 参数错误

➢ UART 功能安装使能函数：uart_driver_install();

函数原型	esp_err_t uart_driver_install (uart_port_t uart_num, int rx_buffer_size,

	int tx_buffer_size, int queue_size, QueueHandle_t* uart_queue, int intr_alloc_flags)
函数功能	UART 功能安装使能函数
参数	[in] uart_num:串口号 [in] rx_buffer_size:接收缓存大小 [in] tx_buffer_size:发送缓存大小 [in] queue_size:队列大小 [in] uart_queue:串口队列指针 [in] intr_alloc_flags:分配中断标记
返回值	ESP_OK：成功 ESP_ERR_INVALID_ARG ： 参数错误

➢ UART 发送函数：uart_write_bytes();

函数原型	int uart_write_bytes (uart_port_t uart_num, const char* src, size_t size)
函数功能	UART 发送函数函数
参数	[in] uart_num:串口号 [in] src:发送数据指针 [in]size:发送数据大小
返回值	(-1) ：参数错误 (>=0)：数据已放到发送缓存

➢ UART 读取函数：uart_read_bytes();

函数原型	int uart_read_bytes (uart_port_t uart_num, uint8_t* buf, uint32_t length, TickType_t ticks_to_wait)
函数功能	UART 读取函数
参数	[in] uart_num:串口号 [in] buf:接收数据指针 [in]length:接收数据最大大小 [in]ticks_to_wait:等待时间
返回值	(-1) ：参数错误 (>=0)：数据已放到发送缓存

更多更详细接口请参考官方指南。

4.3. 串口收发代码编写

加载串口相关的头文件、定义串口 IO 映射引脚、定义串口缓存等。

```
#include <stdio.h>
#include "esp_system.h"
#include "esp_spi_flash.h"
#include "esp_wifi.h"
#include "esp_event_loop.h"
#include "esp_log.h"
#include "esp_err.h"
#include "nvs_flash.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/ledc.h"
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/uart.h"
#include "driver/gpio.h"
#include "string.h"
//UART1
#define RX1_BUF_SIZE        (1024)
#define TX1_BUF_SIZE        (512)
#define TXD1_PIN            (GPIO_NUM_5)
#define RXD1_PIN            (GPIO_NUM_4)
//UART2
#define RX2_BUF_SIZE        (1024)
#define TX2_BUF_SIZE        (512)
#define TXD2_PIN            (GPIO_NUM_12)
#define RXD2_PIN            (GPIO_NUM_13)
```

- 串口配置函数

```
void uart_init(void)
{
    //串口配置结构体
    uart_config_t uart1_config,uart2_config;
    //串口参数配置->uart1
    uart1_config.baud_rate = 115200;                        //波特率
    uart1_config.data_bits = UART_DATA_8_BITS;              //数据位
    uart1_config.parity = UART_PARITY_DISABLE;              //校验位
    uart1_config.stop_bits = UART_STOP_BITS_1;              //停止位
    uart1_config.flow_ctrl = UART_HW_FLOWCTRL_DISABLE; //硬件流控
    uart_param_config(UART_NUM_1, &uart1_config);      //设置串口
    //IO 映射-> T:IO4  R:IO5
    uart_set_pin(UART_NUM_1, TXD1_PIN, RXD1_PIN, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE);
```

```
    //注册串口服务即使能+设置缓存区大小
    uart_driver_install(UART_NUM_1, RX1_BUF_SIZE * 2, TX1_BUF_SIZE * 2, 0, NULL, 0);

    //串口参数配置->uart2
    uart2_config.baud_rate = 115200;                        //波特率
    uart2_config.data_bits = UART_DATA_8_BITS;              //数据位
    uart2_config.parity = UART_PARITY_DISABLE;              //校验位
    uart2_config.stop_bits = UART_STOP_BITS_1;              //停止位
    uart2_config.flow_ctrl = UART_HW_FLOWCTRL_DISABLE; //硬件流控
    uart_param_config(UART_NUM_2, &uart2_config);      //设置串口
    //IO 映射-> T:IO12  R:IO13
    uart_set_pin(UART_NUM_2, TXD2_PIN, RXD2_PIN, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE);
    //注册串口服务即使能+设置缓存区大小
    uart_driver_install(UART_NUM_2, RX2_BUF_SIZE * 2, TX2_BUF_SIZE * 2, 0, NULL, 0);
}
```

- 主函数：串口初始化、创建两个任务用于串口数据接收、测试串口发送数据等。

```
/*
 * 应用程序的函数入口
*/
void app_main()
{
    //串口初始化
    uart_init();
    //创建串口 1 接收任务
    xTaskCreate(uart1_rx_task, "uart1_rx_task", 1024*2, NULL, configMAX_PRIORITIES, NULL);
    //创建串口 2 接收任务
    xTaskCreate(uart2_rx_task, "uart2_rx_task", 1024*2, NULL, configMAX_PRIORITIES-1, NULL);
    //串口 1 数据发送测试
    uart_write_bytes(UART_NUM_1, "uart1 test OK ", strlen("uart1 test OK "));
    //串口 2 数据发送测试
    uart_write_bytes(UART_NUM_2, "uart2 test OK ", strlen("uart2 test OK "));
}
```

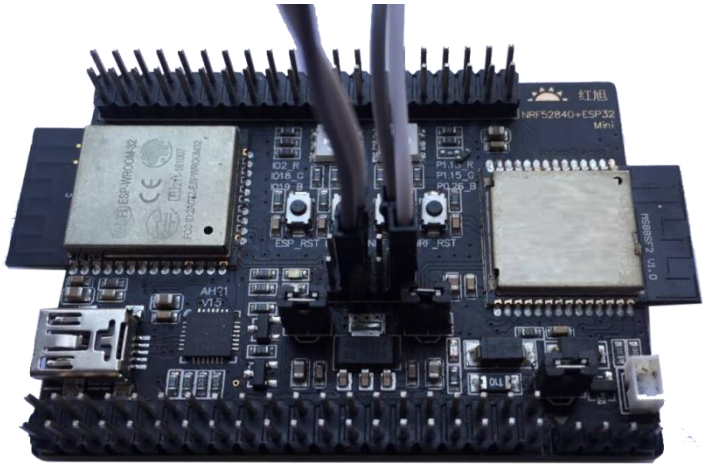
- 两个串口任务

```
/*
* 串口 1 接收任务
*/
void uart1_rx_task()
{
    uint8_t* data = (uint8_t*) malloc(RX1_BUF_SIZE+1);//分配内存，用于串口接收
    while (1) {
        //获取串口 1 接收的数据
```

```
        const int rxBytes = uart_read_bytes(UART_NUM_1, data, RX1_BUF_SIZE, 10 / portTICK_RATE_MS);
        if (rxBytes > 0) {
            data[rxBytes] = 0;//在串口接收的数据增加结束符
                //将接收到的数据发出去
                uart_write_bytes(UART_NUM_1, (char *)data, rxBytes);
        }
    }
    free(data);//释放申请的内存
}
/*
* 串口 2 接收任务：基本同上，省略
*/
```

4.4. 硬件连接

可按照 IO 映射表将串口 1 和串口 2 的 IO 接到 USB 转串口电路上，每次可接一个。如下图是串口 1 接线图。



4.5. 效果展示

5. UART 总结

- 乐鑫已经把串口部分的 API 封装的非常好，直接在任务重解析数据即可。
- 串口发送 32 字节，50ms 周期发送 1 小时无丢包
- 串口发送 32 字节，1ms 发送 5 分钟无死机
- 串口部分初步测试完成
- 源码地址：https://github.com/xiaolongba/wireless-tech