ESP32C3 简单 IO 及串口

代码分析 ：

1. **GPIO 初始化**

接着最简单也是最重要的 IO 初始化及输出

```
gpio_config_t io_conf;//创建 io 配置结构体
io_conf.intr_type = GPIO_INTR_DISABLE;  //禁用中断
io_conf.mode = GPIO_MODE_OUTPUT;    //设置为输出模式
io_conf.pin_bit_mask = GPIO_OUTPUT_PIN_SEL;
//io 脚位掩码，用位运算 左移到需要改变的位进行操作
//#define GPIO_OUTPUT_PIN_SEL  ((1ULL<<GPIO_OUTPUT_IO_0/*18*/) |  \
//(1ULL<<GPIO_OUTPUT_IO_1/*19*/))
//该宏便是先将 1 强转类型为无符号长整型左移再或运算使得 18,19 一起配置为输出
io_conf.pull_down_en = 0;   //下拉模式使能 关闭
io_conf.pull_up_en = 0;     //上拉模式使能 打开
gpio_config(&io_conf);      //gpio 配置
```

2. **输出控制**

输出已经配置好了

```
 gpio_set_level(GPIO_OUTPUT_IO_0, 0);    //18 脚设置为低电平
 gpio_set_level(GPIO_OUTPUT_IO_1, 1);    //19 脚设置为高电平
```

3. **接收初始化**

输入配置

```
  //interrupt of rising edge
    io_conf.intr_type = GPIO_INTR_POSEDGE;  //设置上升沿中断
    //bit mask of the pins, use GPIO4/5 here
    io_conf.pin_bit_mask = GPIO_INPUT_PIN_SEL;  //io 脚位掩码，用位运
算 左移到需要改变的位进行操作
    //set as input mode
    io_conf.mode = GPIO_MODE_INPUT; //设置为输入模式
```

```
  io_conf.pull_up_en = 1; //上拉模式使能 打开
    gpio_config(&io_conf);
```

接收有中断接收和读取 IO 电压接收

中断接收优点 ： 相对于实时读取

读取 IO 优点 ： 简单易操作

4. 获取输入电平

读取 IO ：

```
Int IO0_level = gpio_get_level( GPIO_INPUT_IO_0);   /*返回值为读取的
IO_0 的高低电平*/
```

```
static void IRAM_ATTR gpio_isr_handler(void* arg)
{
    uint32_t gpio_num = (uint32_t) arg;
    xQueueSendFromISR(gpio_evt_queue, &gpio_num, NULL);
}

static void gpio_task_example(void* arg)
{
    uint32_t io_num;
    for(;;)
    {
        if(xQueueReceive(gpio_evt_queue, &io_num, portMAX_DELAY))
        {
            printf("GPIO[%d] intr, val: %d\n", io_num, gpio_get_level(
io_num));
        }
    }
}
```

具体的 demo 在已下载好的 esp-idf\examples\peripherals\gpio\gpio\generic_gpio

实际操作 ：

5. **指定芯片**

idf.py fullclean　　会清除之前的编译

编译前应注意应先设置目标芯片 设置指令 ：

```
idf.py  set-target  esp32c3
```

否则将报错 如下 ：

```
A fatal error occurred: This chip is ESP32-C3 not ESP32. Wrong --chip argument?
CMake Error at run_serial_tool.cmake:50 (message):
  D:/esp-idf4.2/.espressif/python_env/idf4.3_py3.9_env/Scripts/python.exe
  D:/Esp32_idf/esp-idf/components/esptool_py/esptool/esptool.py --chip esp32
  failed
```

6. **烧入已经生成的文件**

```
 D:\Esp32_idf\My_pro\generic_gpio_ok 的目录

2021/09/01  14:25    <DIR>          .
2021/09/01  14:25    <DIR>          ..
2021/09/02  15:45    <DIR>          build
2021/08/24  09:50               236 CMakeLists.txt
2021/08/24  09:50               635 example_test.py
2021/08/24  18:17    <DIR>          main
2021/08/24  09:50               183 Makefile
2021/08/24  09:50               414 README.md
2021/08/25  10:31               138 readme.txt
2021/09/01  14:25            35,165 sdkconfig
2021/08/25  09:18            38,075 sdkconfig.old
               7 个文件         74,846 字节
               4 个目录 270,109,184,000 可用字节

D:\Esp32_idf\My_pro\generic_gpio_ok>
```

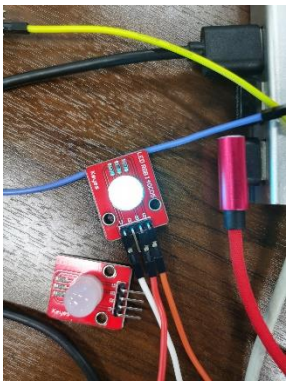
```
D:\Esp32_idf\My_pro\generic_gpio_ok>idf.py build
Executing action: all (aliases: build)
Running cmake in directory d:\esp32_idf\my_pro\generic_gpio_ok\build
Executing "cmake -G Ninja -DPYTHON_DEPS_CHECKED=1 -DESP_PLATFORM=1 -DIDF_TARGET=e
\my_pro\generic_gpio_ok"...
-- Found Git: D:/esp-idf4.2/.espressif/tools/idf-git/2.30.1/cmd/git.exe (found ve
```

编译，idf.py -p COM5 flash 烧入

```
or run 'idf.py -p (PORT) flash'

D:\Esp32_idf\My_pro\generic_gpio_ok>idf.py -p COM5 flash
Executing action: flash
Running ninja in directory d:\esp32_idf\my_pro\generic_gpio_ok\build
Executing "ninja flash"...
[1/4] Performing build step for 'bootloader'
ninja: no work to do.
[1/2] cmd.exe /C "cd /D D:\Esp32_idf\esp-idf\components\e..._idf/esp-idf
```

7. 硬件连接



接线 ： 18 脚接 rgb 灯的 G 段显绿色 19 接 rgb 灯的 B 段

显示为青色，这里的 18 可以和 19 调换，因为点亮的时间与强度一致

1. Printf 函数测试

第一个 demo “hello_word”

即使用 printf 函数 向与电脑连接的串口发送数据

好处 ： 方便快捷

考虑到线程安全，在 FREERTOS 中尽量使用 ESP_LOGX 来输出调试信息和打印消息

```
    /* Print chip information */
    esp_chip_info_t chip_info;   //芯片信息结构体
    esp_chip_info(&chip_info);   //获取芯片信息函数,只需将结构体地址传入
```

```
    printf("This is %s chip with %d CPU core(s), WiFi%s%s, ",
            CONFIG_IDF_TARGET,
            chip_info.cores, /*芯片名称*/
            (chip_info.features & CHIP_FEATURE_BT) ? "/BT" : "",
            (chip_info.features & CHIP_FEATURE_BLE) ? "/BLE" : "");
```

2. 两个串口数据收发代码讲解

接着

```
    uart_config_t uart_config =
    {
        .baud_rate = ECHO_UART_BAUD_RATE,    /*串口波特率 */
        .data_bits = UART_DATA_8_BITS,       /*串口八位数据*/
        .parity    = UART_PARITY_DISABLE,    /*禁用串口奇偶校验*/
        .stop_bits = UART_STOP_BITS_1,       /*串口停止位 1*/
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,  /*串口数据流控制禁用*/
        .source_clk = UART_SCLK_APB,     /*时钟源选择来自于 APB 的时钟*/
    };  //串口配置结构体
    int intr_alloc_flags = 0;
```

配置第二个串口尝试使用两个串口环路数据测试

下面串口 demo 路径为 ： esp-idf\examples\peripherals\uart\uart_echo

```

//串口驱动下载函数 ( TX 环形缓冲区, RX 环形缓冲区, 事件队列句柄和大小, 分配中断的标志)
    ESP_ERROR_CHECK(uart_driver_install(ECHO_UART_PORT_NUM, BUF_SIZE * 2, 0, 0, NULL, intr_alloc_flags));
                    //串口配置函数 (  串口号, 串口配置结构体)
    ESP_ERROR_CHECK(uart_param_config(ECHO_UART_PORT_NUM, &uart_config));
                    //设置通信 IO 函数 ( 波特率, TXIO, RXIO, RTSIO, CTSIO)
    ESP_ERROR_CHECK(uart_set_pin(ECHO_UART_PORT_NUM, ECHO_TEST_TXD, ECHO_TEST_RXD, ECHO_TEST_RTS, ECHO_TEST_CTS));
```

```
// Configure a temporary buffer for the incoming data
    uint8_t *data = (uint8_t *) malloc(BUF_SIZE);  //创建一个大小为 BUF_SIZE 的指针
```

```
    while (1)
    {
        bzero ( data, BUF_SIZE); //清空指针内残留
        // Read data from the UART   从串口读取数据
```

```
        int len = uart_read_bytes(ECHO_UART_PORT_NUM, data, BUF_SIZE, 2
0 / portTICK_RATE_MS);

        // if ( *data != 0)  //如果不是默认值则认为该数据为正确值
        {
            printf ( "%s\n", data);
        }


        // Write data back to the UART
        uart_write_bytes( ECHO_UART_PORT_NUM, (const char *) data, len)
;
    }
```

循环读写，此读函数不阻塞

3. **指定芯片**

idf.py fullclean　　会清除之前的编译

编译前应注意应先设置目标芯片 设置指令 ：

`idf.py set-target esp32c3`

否则将报错 如下 ：

```
A fatal error occurred: This chip is ESP32-C3 not ESP32. Wrong --chip argument?
CMake Error at run_serial_tool.cmake:50 (message):
  D:/esp-idf4.2/.espressif/python_env/idf4.3_py3.9_env/Scripts/python.exe
  D:/Esp32_idf/esp-idf/components/esptool_py/esptool/esptool.py --chip esp32
  failed
```

4. **查看端口号**

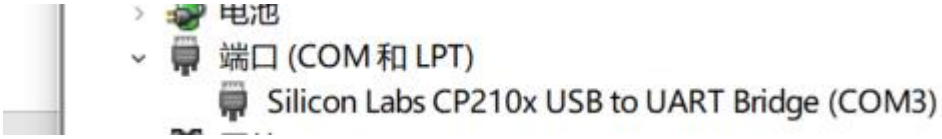
连接设备 ：

Windows 环境 打开设备管理器 找到

按下 windows 按键/鼠标单击左小角

键盘输入设备管理器

找到 COM 和 LPT 选项



当 COM 口未连接设备不会有该选项

5. **软件讲解**

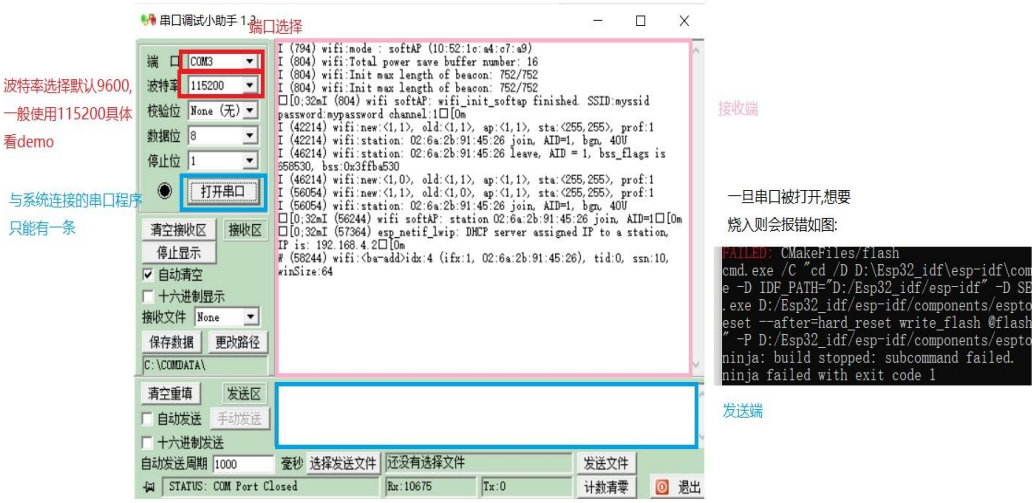
idf.py -p (PORT) monitor

这个 PORT 即端口号，如上则是 COM3 命令 : idf.py -p COM3 monitor

也可以使用简单的串口调试工具

链接：https://pan.baidu.com/s/1zD5JzLBzn72FNymSp4gSAw

提取码：1234

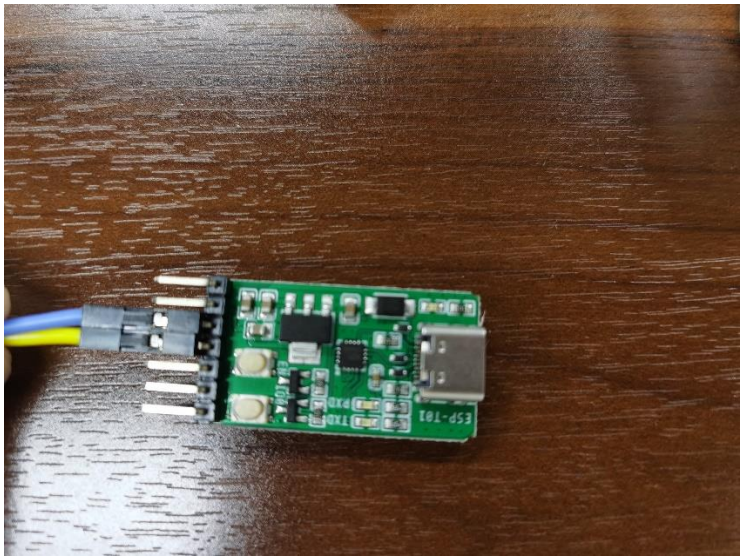


也可以运行以下命令，一次性执行构建、烧录和监视过程：

idf.py -p PORT flash monitor

6. 硬件连接

设备连接方式 ： 需要用到一个 TTL 转 USB 模块
比如此转换是 TTL 转 TYPE-C 母 在接跟 TYPE-C 公转 USB 公接入电脑

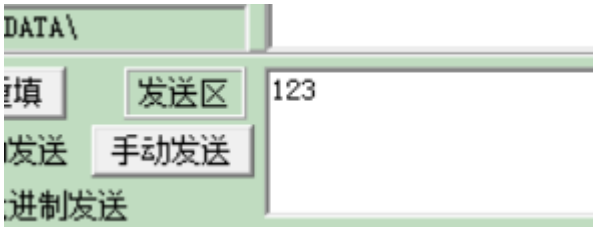


即将发送数据的 TX （ 4 脚） 连接到转换模块的 RX
接收数据的 RX（ 5 脚） 连接到转换模块的 TX （ 发送端）

实现原理 ：
假设 ESP32-C3 的 printf 的串口为 UART_1
ESP32-C3 新配置的串口为 UART_2

7. 功能实现

电脑通过串口 UART_2 发送数据到 ESP32-C3



ESP32-C3 使用 printf 函数 通过 UART1 打印给电脑 printf (“%s\n”, data);

并且通过串口打印到电脑的串口,即可在串口调试小助手上接收

```
uart_write_bytes( ECHO_UART_PORT_NUM, (const char *) data, len);
```