SDIO
读写 TF 卡

淘宝店铺：

PC 端：

http://n-xytrt8gqu585po94mwj5atokcyd4.taobao.com/index.htm

手机端：

https://shop.m.taobao.com/shop/shop_index.htm?sellerId=755668508&shopId=104493595&inShopPageId=423890608&pathInfo=shop/index2

资料下载地址：

链接：https://pan.baidu.com/s/1kCjD8yktZECSGmHomx_veg?pwd=q8er
提取码：q8er

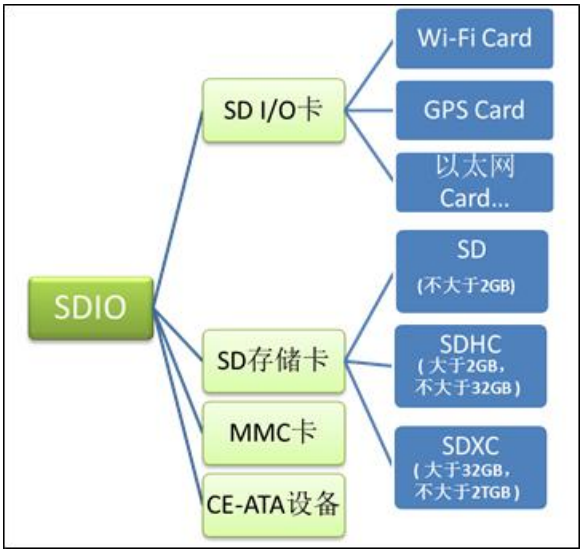
源码下载地址：

https://gitee.com/vi-iot/esp32-board.git

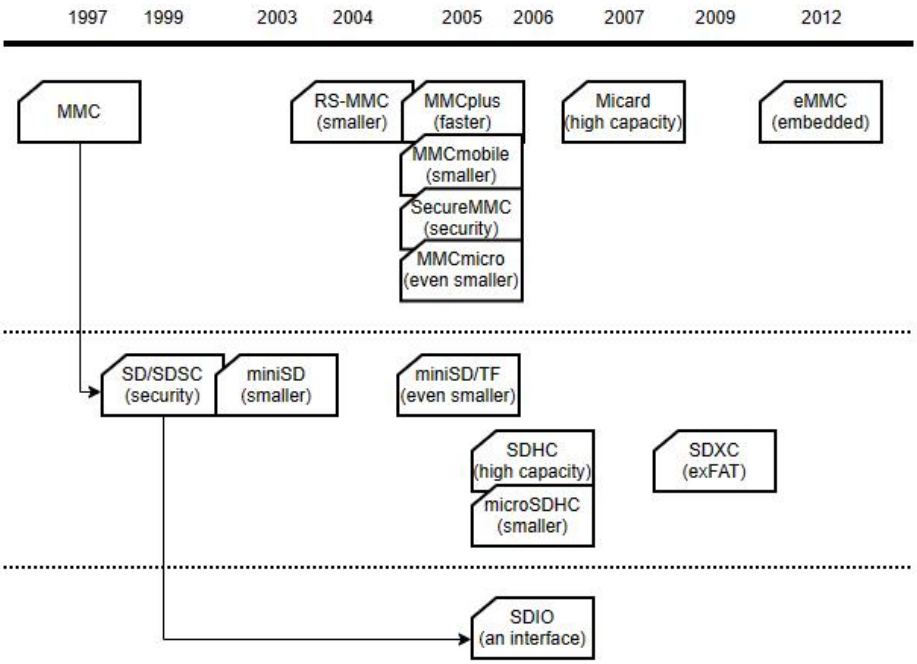
一、SDIO 介绍

SDIO，全称：Secure Digital Input and Output，即安全数字输入输出接口。它是在 SD 卡接口的基础上发展而来，它可以兼容之前的 SD 卡，并可以连接 SDIO 接口设备，比如：蓝牙、WIFI、照相机等。SDIO 和 SD 卡规范间的一个重要区别是增加了低速标准。低速卡的目标应用是以最小的硬件开支支持低速 I/O 能力。低速卡支持类似调制解调器、条码扫描仪和 GPS 接收器等应用。

以下是 SDIO 的兼容性



这里也提一下 MMC 卡，MMC(Multi-Media Card，多媒体卡)由西门子公司 Siemens 和 SanDisk 于 1997 年推出。由于它的封装技术较为先进，7 针引脚，体积小、重量轻、非常符合移动存储的需要。MMC 支持 1bit 模式，20MHz 时钟，采用总线结构。 SD 卡是基于 MMC 卡发展而来的，具体的关系和历史发展大家可以看下图



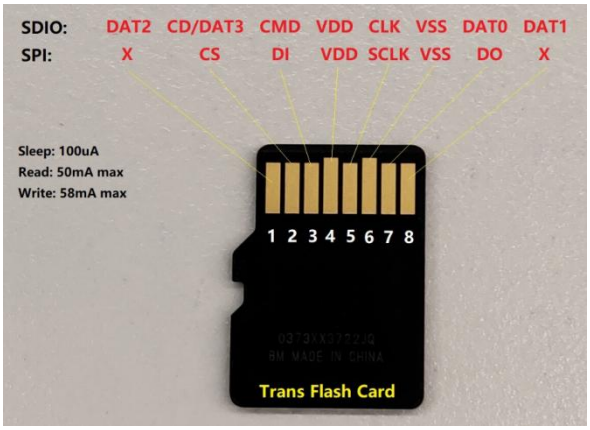
发展到今天 MMC 卡基本退出历史，取而代之的是 eMMC 卡主要应用于嵌入式领域，eMMC 卡是 bga 封装，一般焊接在板子上。

在配套教程的开发板上，预留了一个 MicroSD/TF 卡座，在 esp-idf 中驱动 SD 卡/MicroSD 卡/eMMC 卡的组件称为 sdmmc 模块，这个模块对常用的存储卡操作进行了封装兼容，并向上提供了友好的接口，我们无须关注具体的驱动实现以及协议内容，即可方便的操作 MicroSD 卡

ESP32 的 SDMMC 主机外设共有两个卡槽，用于插入 SD 卡、连接 SDIO 设备或连接 eMMC 芯片，每个卡槽均可单独使用。

信号	卡槽 0	卡槽 1
CMD	GPIO11	GPIO15
CLK	GPIO6	GPIO14
D0	GPIO7	GPIO2
D1	GPIO8	GPIO4
D2	GPIO9	GPIO12
D3	GPIO10	GPIO13
D4	GPIO16	
D5	GPIO17	
D6	GPIO5	
D7	GPIO18	
CD	来自 GPIO 交换矩阵的任意输入	来自 GPIO 交换矩阵的任意输入
WP	来自 GPIO 交换矩阵的任意输入	来自 GPIO 交换矩阵的任意输入

由于卡槽 0 引脚与官方的 ESP32 模组 Flash 引脚有冲突，因此我们一般使用卡槽 1。MicroSD 卡，对于 MicroSD 卡的接口，CD 插入检测与 D3 引脚公用，没有 WP 写保护引脚。具体 MicroSD 卡接口如下图所示。



MicroSD 可以使用 SPI 和 SDIO 模式，本例程使用 SDIO 模式，可同时传输 4bit 数据。

二、ESP32 读写 MicroSD 卡

由于 esp-idf 中对 SD 卡的操封装的相当好，因此初始化、读写 SD 卡等操作代码量不多，并且加入了 VFS 虚拟文件系统管理，使得我们对 SD 卡的操作可以使用 C 语言标准文件读写 api，比如 fopen、fwrite 等操作。我们先看下代码。代码位于 esp32-board/sdcard

```
void app_main(void)
{
    esp_err_t ret;
    esp_vfs_fat_sdmmc_mount_config_t mount_config = {
        .format_if_mount_failed = true,     //挂载失败是否执行格式化
        .max_files = 5,                     //最大可打开文件数
        .allocation_unit_size = 16 * 1024   //执行格式化时的分配单元大小（分配单元越大，读写越快）
    };
    sdmmc_card_t *card;
    const char mount_point[] = MOUNT_POINT;
    ESP_LOGI(TAG, "Initializing SD card");

    ESP_LOGI(TAG, "Using SDMMC peripheral");
    //默认配置，速度 20MHz,使用卡槽 1
    sdmmc_host_t host = SDMMC_HOST_DEFAULT();
    //默认的 IO 管脚配置，
    sdmmc_slot_config_t slot_config = SDMMC_SLOT_CONFIG_DEFAULT();
    //4 位数据
    slot_config.width = 4;
    //不使用通过 IO 矩阵进行映射的管脚，只使用默认支持的 SDMMC 管脚，可以获得最大性能
    #if 0
    slot_config.clk = CONFIG_EXAMPLE_PIN_CLK;
    slot_config.cmd = CONFIG_EXAMPLE_PIN_CMD;
    slot_config.d0 = CONFIG_EXAMPLE_PIN_D0;
    slot_config.d1 = CONFIG_EXAMPLE_PIN_D1;
    slot_config.d2 = CONFIG_EXAMPLE_PIN_D2;
```

```
    slot_config.d3 = CONFIG_EXAMPLE_PIN_D3;
    #endif
    // Enable internal pullups on enabled pins. The internal pullups
    // are insufficient however, please make sure 10k external pullups are
    // connected on the bus. This is for debug / example purpose only.
    //管脚启用内部上拉
    slot_config.flags |= SDMMC_SLOT_FLAG_INTERNAL_PULLUP;
    ESP_LOGI(TAG, "Mounting filesystem");
    ret = esp_vfs_fat_sdmmc_mount(mount_point, &host, &slot_config,
&mount_config, &card);
    if (ret != ESP_OK) {
        if (ret == ESP_FAIL) {
            ESP_LOGE(TAG, "Failed to mount filesystem. ");
        } else {
            ESP_LOGE(TAG, "Failed to initialize the card (%s). "
                     "Make sure SD card lines have pull-up resistors in
place.", esp_err_to_name(ret));
        }
        return;
    }
    ESP_LOGI(TAG, "Filesystem mounted");
    // Card has been initialized, print its properties
    sdmmc_card_print_info(stdout, card);
    // Use POSIX and C standard library functions to work with files:
    // First create a file.
    const char *file_hello = MOUNT_POINT"/hello.txt";
    char data[EXAMPLE_MAX_CHAR_SIZE];
    snprintf(data, EXAMPLE_MAX_CHAR_SIZE, "%s %s!\n", "Hello",
card->cid.name);
    ret = s_example_write_file(file_hello, data);
    if (ret != ESP_OK) {
        return;
    }
    const char *file_foo = MOUNT_POINT"/foo.txt";
    // Check if destination file exists before renaming
    struct stat st;
    if (stat(file_foo, &st) == 0) {
        // Delete it if it exists
        unlink(file_foo);
    }
    // Rename original file
    ESP_LOGI(TAG, "Renaming file %s to %s", file_hello, file_foo);
    if (rename(file_hello, file_foo) != 0) {
        ESP_LOGE(TAG, "Rename failed");
```

```
        return;
    }
    ret = s_example_read_file(file_foo);
    if (ret != ESP_OK) {
        return;
    }

    const char *file_nihao = MOUNT_POINT"/nihao.txt";
    memset(data, 0, EXAMPLE_MAX_CHAR_SIZE);
    snprintf(data, EXAMPLE_MAX_CHAR_SIZE, "%s %s!\n", "Nihao",
card->cid.name);
    ret = s_example_write_file(file_nihao, data);
    if (ret != ESP_OK) {
        return;
    }
    //Open file for reading
    ret = s_example_read_file(file_nihao);
    if (ret != ESP_OK) {
        return;
    }
    // All done, unmount partition and disable SDMMC peripheral
    esp_vfs_fat_sdcard_unmount(mount_point, card);
    ESP_LOGI(TAG, "Card unmounted");
}
```

初始化的时候需要填充 esp_vfs_fat_sdmmc_mount_config_t、sdmmc_slot_config_t 这两个结构体，然后通过 esp_vfs_fat_sdmmc_mount 函数把 SD 卡挂载到对应的挂载点上（函数的第一个参数）。具体的内容大家可以自行查看定义，并不复杂，这里有几点要注意一下，slot_config.flags |= SDMMC_SLOT_FLAG_INTERNAL_PULLUP 这句是对 SDIO 的管脚启用内部上拉电阻，严格来说我们在设计电路的时候要对 SDIO 四个 DATA 管脚都接一个 10K-90K 的上拉电阻，而 ESP32 内部给我们提供了上拉的功能，是不是就意味着我们不用外接上拉呢？其实这里官方也给了注释，启用这个内部上拉，功能有待验证，最稳的还是接个 10K 上拉，但在实际应用中我发现了一个问题，如果 DATA2（GPIO12）引脚接了上拉，则 ESP32 在上电的时候认为 Flash 是 1.8V 供电的，而板子上全都用 3.3V 供电，这样我们程序起不来。官方的解决方法是烧录 efuse 的存储信息，让 ESP32 强制以 3.3V 工作。还有一个地方是 DATA0（GPIO2）引脚，如果我接了 10K 上拉，则无法烧录程序，这个和系统的启动模式有关。因此保险起见，我把 DATA0 和 DATA2 上的上拉都 NC 了，启用了内部上拉，实际测试工作正常。大家如果有需要可以自行焊接这两个电阻。

在挂载成功后，sdmmc_card_print_info(stdout, card)函数可以打印 SD 卡信息，之后我们就可以使用 fopen、fwrite、fread 标准 C 库函数进行操作了。