通讯:

单工/半双工/全双工/异步同步

单工，即数据传输只在一个方向上传输，只能你给我发送或者我给你发送，方向是固定的，不能实现双向通信，如：室外天线电视、调频广播等。

半双工比单工先进一点，传输方向可以切换，允许数据在两个方向上传输，但是某个时刻，只允许数据在一个方向上传输，可以基本双向通信，如：对讲机，IIC通信。

全双工，允许数据同时在两个方向传输。发送和接收完全独立，在发送的同时可以接收信号，或者在接收的同时可以发送。它要求发送和接收设备都要有独立的发送和接收能力，如：电话通信，SPI通信，串口通信。

串行通信可以分为两种类型，一种叫同步通信，另一种叫异步通信。

简单的说，就是同步通信需要时钟信号，而异步通信不需要时钟信号。

- 同步：发送方发出数据后，等接收方发回响应以后才发下一个数据包的通讯方式。
- 异步：发送方发出数据后，不等接收方发回响应，接着发送下个数据包的通讯方式。

SPI和IIC为同步通信，UART为异步通信，而USART为同步&异步通信。

- USART：通用同步和异步收发器
- UART：通用异步收发器

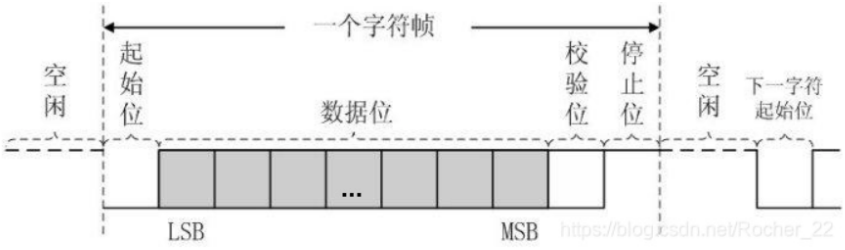
即USART支持同步和异步收发，而UART只支持异步收发。

如STM32的串口工作在同步模式时，即智能卡模式时，就需要连接同步时钟引脚。

串口通讯

奇偶检验等N、O、E、M、S 五种串口检验位类型

校验位：串口通信中的**检错方式**。串口在接收数据时，如果无检验位，则只要检测到串口出现了数据，数据总能接收到；而采用了某种检验后，只有校验码正确的数据才能被正确的接收。



串口通信校验方式

串口通信过程中有五种校验方式：**奇校验(ODD)**、**偶校验(EVEN)**，**1校验(MARK)** 、 **0校验(SPACE)**，**无校验(NONE)**。

奇偶校验位

串口会设置校验位（数据位后面的一位），用一个值确保传输的数据有偶个或者奇个逻辑高位（即1的个数为偶个或者奇个）。 *（例如，如果数据是011，那么对于偶校验，校验位为0，保证逻辑高的位数是偶数个。如果是奇校验，校验位为1，这样就有3个逻辑高位。）*

1. **奇校验(ODD)**：每个字节传送整个过程中bit为1的个数是奇数个（校验位调整个数）。
2. **偶校验(EVEN)**：每个字节传送整个过程中bit为1的个数是偶数个（校验位调整个数）。

固定校验位(Stick)

串口简单设置校验位，置位逻辑高(1)或者逻辑低(0)校验。 高位和低位不是真正的检查数据，但这样使得接收设备能够知道一个位的状态，有机会判断是否有噪声干扰了通信或者是否传输和接收数据是否不同步。

3. **1校验(MARK)**：校验位总为1。
4. **0校验(SPACE)**：校验位总为0。

无校验位

5. **无校验(NONE)**： 没有校验位。

一、 检验位

在串行通讯所发送数据的最后一位，用来**粗略**的检验数据在传输过程中是否有出错。

二、检验位的五种类型

1. N（None [没有]）

【无校验】 不加校验位，可以少传输一位数据

2. O（Odd [单、奇、奇怪]）

【奇校验】 两种理解方式

a. 要传输的数据中（不包含校验位）有**奇数个**‘1’那么校验位为‘0’， 反之为‘1’

例： 数据‘1111 000’ 偶数个‘1’ 所以添加校验位为‘1’ 整体为‘1111 0000 1’

b. 所有位数中（数据+包含校验位）有**奇数个**‘1’

例：数据‘1111 0000’ 已有的‘1’为偶数个，为了确保所有位数中‘1’的个数为奇数，就要让校验位为‘1’ 整体为 ‘1111 0000 1’

例：数据‘1110 0000’ 已经有奇数个‘1’ 所以校验位为‘0’ 整体为‘1110 0000 0’

3. E (Even 偶、双、平均)

【偶校验】 两种理解方式

a. 要传输的数据中（不包含校验位）有**偶数个**‘1’那么校验位为‘0’， 反之为‘1’

例： 数据‘1111 000’ 偶数个‘1’ 所以添加校验位为‘0’ 整体为‘1111 0000 0’

b. 所有位数中（数据+包含校验位）有**偶数个**‘1’

例：数据‘1111 0000’ 已经有偶数个‘1’ 所以校验位为‘0’ 整体为‘1111 0000 0’

例：数据‘1110 0000’ 已有的‘1’为奇数个，为了确保所有位数中‘1’的个数为偶数，就要让校验位为‘1’ 整体为 ‘1110 0000 1’

4. M（Mark 标记、符合）

【固定1】检验位固定为1

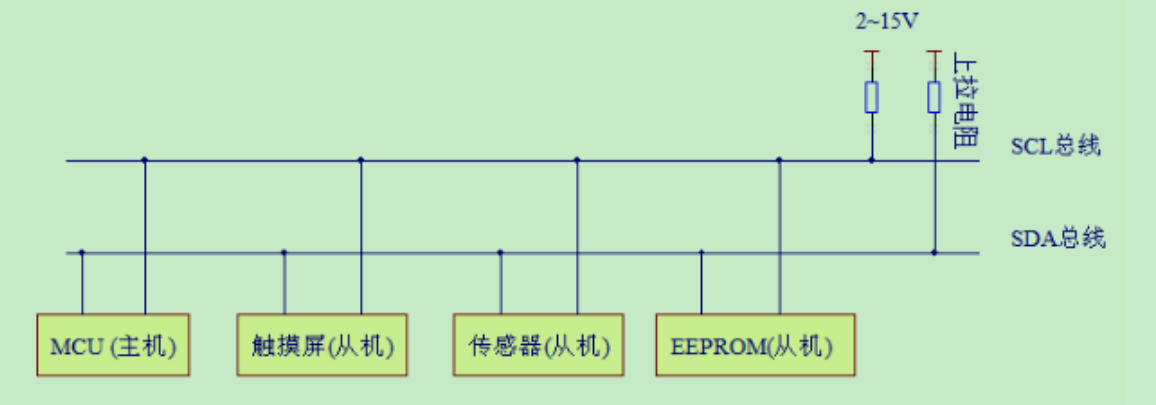
5. S（Space 空间、空地）

【固定0】校验位固定为0

IIC

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。

I2c物理层



它的物理层有如下特点：

(1) 它是一个支持设备的总线。“总线”指多个设备共用的信号线。在一个 I2C 通讯总线中，可连接多个 I2C 通讯设备，支持多个通讯主机及多个通讯从机。

(2) 一个 I2C 总线只使用两条总线线路，一条双向串行数据线 (SDA)，一条串行时钟线 (SCL)。数据线即用来表示数据，时钟线用于数据收发同步。

(3) 每个连接到总线的设备都有一个独立的地址，主机可以利用这个地址进行不同设备之间的访问。

(4) 总线通过上拉电阻接到电源。当 I2C 设备空闲时，会输出高阻态，而当所有设备都空闲，都输出高阻态时，由上拉电阻把总线拉成高电平。

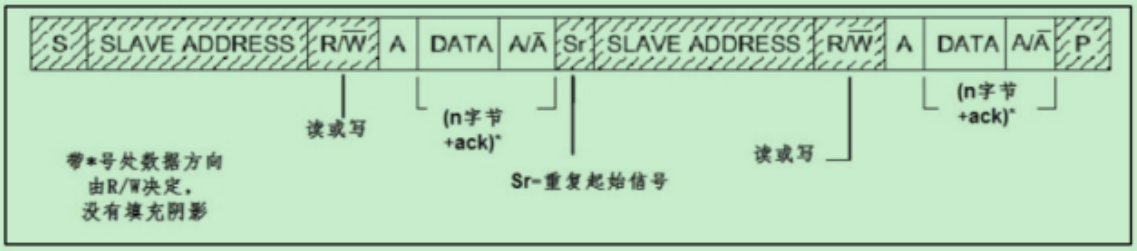
(5) 多个主机同时使用总线时，为了防止数据冲突，会利用仲裁方式决定由哪个设备占用总线。

(6) 具有三种传输模式：标准模式传输速率为 100kbit/s，快速模式为 400kbit/s，高速模式下可达 3.4Mbit/s，但目前大多 I^2C 设备尚不支持高速模式。

(7) 连接到相同总线的 IC 数量受到总线的最大电容 400pF 限制。

I2c协议层

I2C 的协议定义了通讯的起始和停止信号、数据有效性、响应、仲裁、时钟同步和地址广播等环节。



图例：▨ 数据由主机传输至从机　S： 传输开始信号

SLAVE_ADDRESS: 从机地址

□ 数据由从机传输至主机　R/$\overline{W}$： 传输方向选择位，1 为读，0 为写

A/$\overline{A}$ ： 应答(ACK)或非应答(NACK)信号

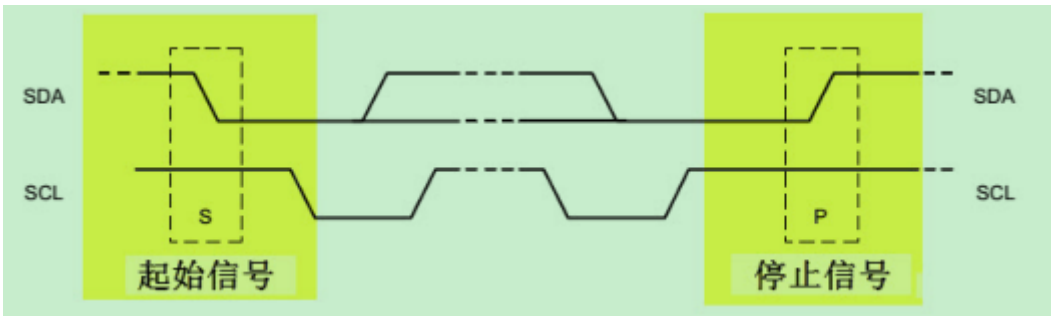
P ： 停止传输信号

这些图表示的是主机和从机通讯时，SDA 线的数据包序列。

其中 S 表示由主机的 I2C 接口产生的传输起始信号 (S)，这时连接到 I2C 总线上的所有从机都会接收到这个信号。
起始信号产生后，所有从机就开始等待主机紧接下来广播的从机地址信号 (SLAVE_ADDRESS)。在 I2C 总线上，每个设备的地址都是唯一的，当主机广播的地址与某个设备地址相同时，这个设备就被选中了，没被选中的设备将会忽略之后的数据信号。根据 I2C 协议，这个从机地址可以是7 位或 10 位。
在地址位之后，是传输方向的选择位，该位为 0 时，表示后面的数据传输方向是由主机传输至从机，即主机向从机写数据。该位为 1 时，则相反，即主机由从机读数据。从机接收到匹配的地址后，主机或从机会返回一个应答 (ACK) 或非应答 (NACK) 信号，只有接收到应答信号后，主机才能继续发送或接收数据。

1.通讯的起始和停止信号

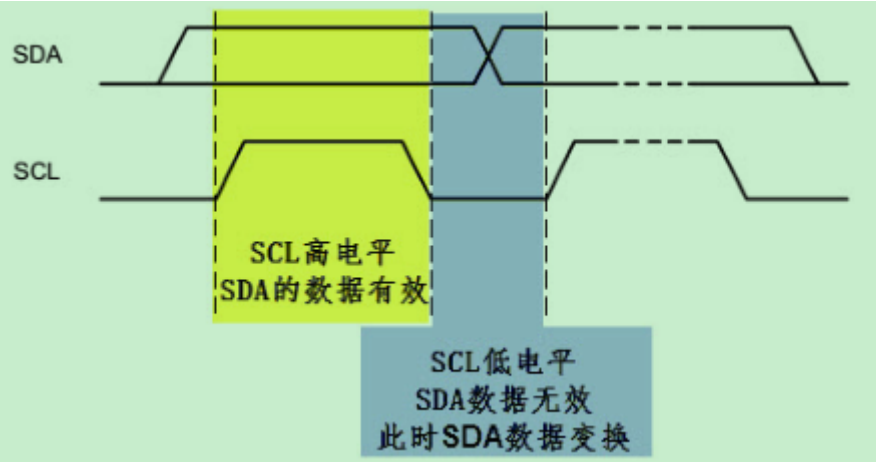
当 SCL 线是高电平时 SDA 线从高电平向低电平切换，这个情况表示通讯的起始。当 SCL 是高电平时 SDA 线由低电平向高电平切换，表示通讯的停止。起始和停止信号一般由主机产生。



2.数据有效性

I2C 使用 SDA 信号线来传输数据，使用 SCL 信号线进行数据同步。

SDA 数据线在 SCL 的每个时钟周期传输一位数据.传输时，SCL 为高电平的时候 SDA 表示的数据有效,此时SDA的电平高低分别表示数据1/数据0。当 SCL 为低电平时，SDA的数据无效，一般在这个时候 SDA 进行电平切换，为下一次表示数据做好准备。每次数据传输都以字节为单位，每次传输的字节数不受限制。：



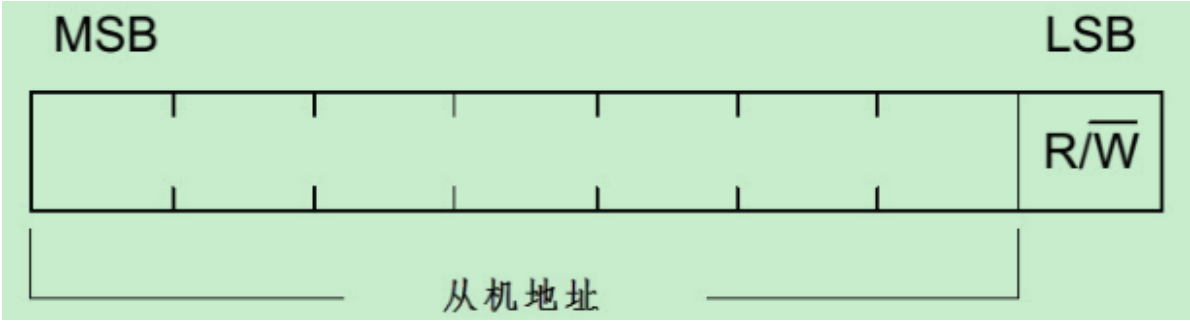
3.地址及数据方向

主机发起通讯时，通过 SDA 信号线发送设备地址来查找从机。

I2C 协议规定设备地址可以是 7 位或 10 位，紧跟设备地址的一个数据位用来表示数据传输方向，它是数据方向位 （R或者/W），第 8位或第 11 位。数据方向位为“1”时表示主机由从机读数据，该位为“0”时表示主机向从机写数据。

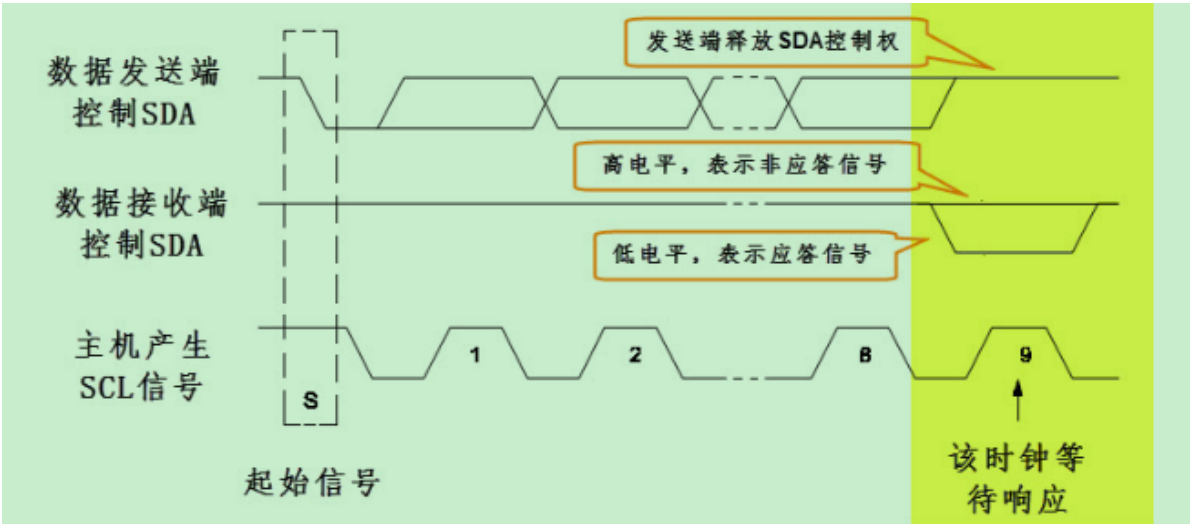
读数据方向时，主机会释放对 SDA 信号线的控制，由从机控制 SDA 信号线.主机接收信号.

写数据方向时，SDA 由主机控制，从机接收信号。



4.响应

传输时主机产生时钟，在第 9 个时钟时，数据发送端会释放 SDA 的控制权，由数据接收端控制SDA，若 SDA 为高电平，表示非应答信号 (NACK)，低电平表示应答信号 (ACK)。



32.768KHZ

实时时钟晶振为什么选择是32768Hz的晶振，在百度上搜索的话大部分的答案都是说2的15次方是32768，使用这个频率的晶振，人们可以很容易的通过分频电路得到1Hz的计时脉冲。但是话有说回来了，2的整数次方很多为什么偏偏选择15呢?

2的15次方正好等于32768，反过来讲，如果要把32.768K的时钟频率经过15次分频的话，得到的频率正好是1Hz。

以下是关于时钟晶振频率选择所需要考虑的几点：

1.频度越高计时精度越高，误差越小。

2.由于各种原因，每个晶振的实际频率与其标称频率之间也存在偏差。

3.晶振的工作环境对晶振的频率也有影响，用晶振的频率稳定度来表示不同晶振受环境影响的大小，其单位是ppm（百万分之一）

4.通常工作频率越高，单片机等数字电路的功耗越大，32.768KHz这个频率比较低，对降低电路功耗有利。

5.晶振的 32.768KHZ是大于20Khz(人耳听力上限)的第一个2的整幂数。

FLASH

内部FLASH

理论

1.解锁

内部 FLASH 空间主要存储的是应用程序，是非常关键的数据，为了防止误操作修改了这些内容，芯片复位后默认会给控制寄存器 FLASH_CR 上锁，这个时候不允许设置 FLASH 的控制寄存器，从而不能修改 FLASH 中的内容。
所以对 FLASH 写入数据前，需要先给它解锁。解锁的操作步骤如下：
(1) 往 FPEC 键寄存器 FLASH_KEYR 中写入 KEY1 = 0x45670123
(2) 再往 FPEC 键寄存器 FLASH_KEYR 中写入 KEY2 = 0xCDEF89AB

2.页擦除

在写入新的数据前，需要先擦除存储区域，STM32 提供了页（扇区）擦除指令和整个 FLASH 擦除 (批量擦除) 的指令，批量擦除指令仅针对主存储区。 页擦除的过程如下：
(1) 检查 FLASH_SR 寄存器中的“忙碌寄存器位 BSY”，以确认当前未执行任何 Flash 操作；
(2) 在 FLASH_CR 寄存器中，将“激活页擦除寄存器位 PER ”置 1，
(3) 用 FLASH_AR 寄存器选择要擦除的页；
(4) 将 FLASH_CR 寄存器中的“开始擦除寄存器位 STRT ”置 1，开始擦除；
(5) 等待 BSY 位被清零时，表示擦除完成。

3.写入数据

擦除完毕后即可写入数据，写入数据的过程并不是仅仅使用指针向地址赋值，赋值前还还需要配置一系列的寄存器，步骤如下：
(1) 检查 FLASH_SR 中的 BSY 位，以确认当前未执行任何其它的内部 Flash 操作；
(2) 将 FLASH_CR 寄存器中的“激活编程寄存器位 PG”置 1；
(3) 向指定的 FLASH 存储器地址执行数据写入操作，每次只能以 16 位的方式写入；
(4) 等待 BSY 位被清零时，表示写入完成。

应用例子

外接FLASh

DMA的基本定义
DMA，全称Direct Memory Access，即直接存储器访问。

DMA传输将数据从一个地址空间复制到另一个地址空间，
提供在外设和存储器之间或者存储器和存储器之间的高速数据传输。
当CPU初始化这个传输动作，传输动作本身是由DMA控制器来实现和完成的。
DMA传输方式无需CPU直接控制传输，也没有中断处理方式那样保留现场和恢复现场过程，
通过硬件为RAM和IO设备开辟一条直接传输数据的通道，使得CPU的效率大大提高。

A= 110101x1
B= 0000 0010
~B= 1111 1101
A&=~B---->1101 0101 //把B中为1的位对应于A的同样位上置0，A的其他位不变

A= 1101 0111
B= 0000 0010
A&=B----> 0000 0010 //把A中其他位置0,

A |= B
A=0010 110x,
B=0000 0001，则执行结果为A=00101101，
也就是说A |= B是给B中为1的位对应于A的同样位上置1，A的其他位不变

启动文件由汇编(xxx.s)编写，是系统上电复位后第一个执行的程序。

主要做了以下工作：

1.初始化堆栈指针

```
; Stack Configuration
; Stack Size (in Bytes) <0x0-0xFFFFFFFF:8>

Stack_Size      EQU     0x00000200

                AREA    STACK, NOINIT, READWRITE, ALIGN=3
Stack_Mem       SPACE   Stack_Size
__initial_sp
```

1. stack栈，开辟栈的大小为 0X00000400（1KB），名字为 STACK，NOINIT 即不初始化，可读可写，8（2^3）字节对齐
2. 栈的作用是用于局部变量，函数调用，函数形参等的开销，栈的大小不能超过内部 SRAM 的大小。如果编写的程序比较大，定义的局部变量很多，那么就需要修改栈的大小。硬 fault 的时候，这时你就要考虑下是不是栈不够大，溢出了。

```
; Heap Configuration
;  Heap Size (in Bytes) <0x0-0xFFFFFFFF:8>

Heap_Size       EQU     0x00000200

                AREA    HEAP, NOINIT, READWRITE, ALIGN=3
__heap_base
Heap_Mem        SPACE   Heap_Size
__heap_limit
```

```
Heap堆
开辟堆的大小为 0X00000200（512 字节），名字为 HEAP，NOINIT 即不初始化，可读可写，
8（2^3）字节对齐。__heap_base 表示对的起始地址，__heap_limit 表示堆的结束地址。堆是由低向高生长的，跟栈的生长方向相反。
堆主要用来动态内存的分配，像 malloc() 函数申请的内存就在堆上面。
```

2.初始化PC指针 Reset_Handler

3.初始化中断向量表

4.配置系统时钟 SystemInit

5.调用C库函数__main初始化用户堆栈,从而最终调用main函数.

15.3.4 复位程序

```
AREA |.text|, CODE, READONLY
```

定义一个名称为.text 的代码段，可读。

```
Reset_Handler PROC
              EXPORT  Reset_Handler     [WEAK]
              IMPORT  SystemInit
              IMPORT  __main

              LDR     R0, =SystemInit
              BLX     R0
              LDR     R0, =__main
              BX      R0
              ENDP
```

复位子程序是系统上电后第一个执行的程序，调用 SystemInit 函数初始化系统时钟，然后调用 C 库函数 _mian，最终调用 main 函数去到 C 的世界。

WEAK：表示弱定义，如果外部文件优先定义了该标号则首先引用该标号，如果外部文件没有声明也不会出错。这里表示复位子程序可以由用户在其他文件重新实现，这里并不是唯一的。

IMPORT：表示该标号来自外部文件，跟 C 语言中的 EXTERN 关键字类似。这里表示 SystemInit 和 __main 这两个函数均来自外部的文件。

SystemInit() 是一个标准的库函数，在 system_stm32f10x.c 这个库文件总定义。主要作用是配置系统时钟，这里调用这个函数之后，单片机的系统时钟配被配置为 72M。

__main 是一个标准的 C 库函数，主要作用是初始化用户堆栈，并在函数的最后调用 main 函数去到 C 的世界。这就是为什么我们写的程序都有一个 main 函数的原因。

2024/2/20

SPI Flash 128Block

Sector 4KB(4096个地址) 1KB=1024字节

Block 64KB

页写入一次256个字节(Byte)