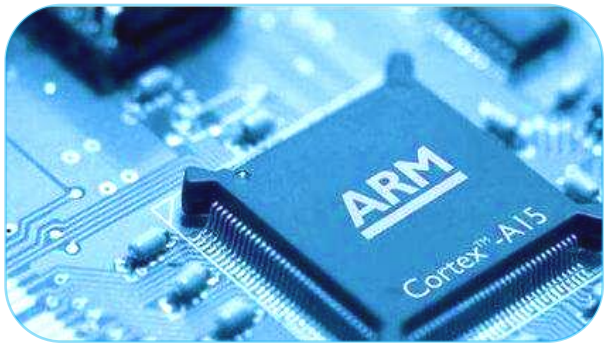
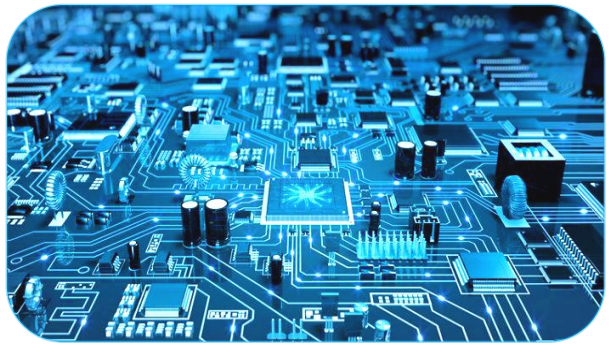
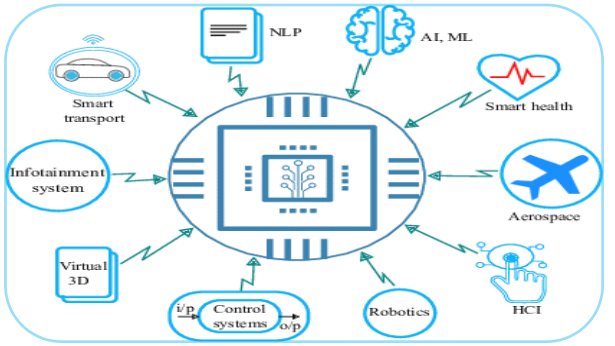
《嵌入式系统课程设计》之

系统定时器与USART







课程回顾

- 启动文件介绍
- RCC时钟管理
- STM32中断介绍
- EXTI-外部中断实验

启动文件简介

启动文件由汇编编写，是系统上电复位后第一个执行的程序。

主要完成以下任务：

- 初始化堆栈指针 SP=_initial_sp
- 初始化 PC 指针=Reset_Handler
- 初始化中断向量表
- 配置系统时钟
- 调用 C 库函数_main 初始化用户堆栈，从而最终调用 main 函数去到 C 的世界

RCC简介

RCC（**Reset and Clock Control**）模块负责控制时钟系统和复位系统的配置和管理，包括：

- **系统时钟源选择和配置**： RCC可以选择并配置系统的时钟源，如外部高速时钟（HSE）、内部高速时钟（HSI）、PLL（相位锁定环）等。通过配置这些时钟源，RCC可以为系统和外设提供合适的时钟频率。
- **外设时钟的使能和管理**： RCC能够控制每个外设模块的时钟，使能或关闭对应的外设时钟，以节省功耗。例如，可以通过RCC开启或关闭GPIO、USART、ADC、SPI等外设的时钟。
- **系统复位管理**： RCC包含多个复位控制功能，包括系统复位、外设复位、备份域复位等。复位可以通过软件手动触发，或在系统上电时自动触发。
- **时钟分频设置**： RCC允许设置时钟分频器，以便为不同的外设或系统部分提供不同的时钟频率。通过合理配置分频器，可以让时钟频率适配外设的工作要求。
- **低功耗模式支持**： RCC还支持低功耗模式的时钟管理，比如在待机模式或睡眠模式下，关闭不必要的时钟，以降低系统功耗。

RCC时钟配置

在STM32时钟管理部分，RCC模块的负责系统时钟的选择、生成、分配和分配给外设

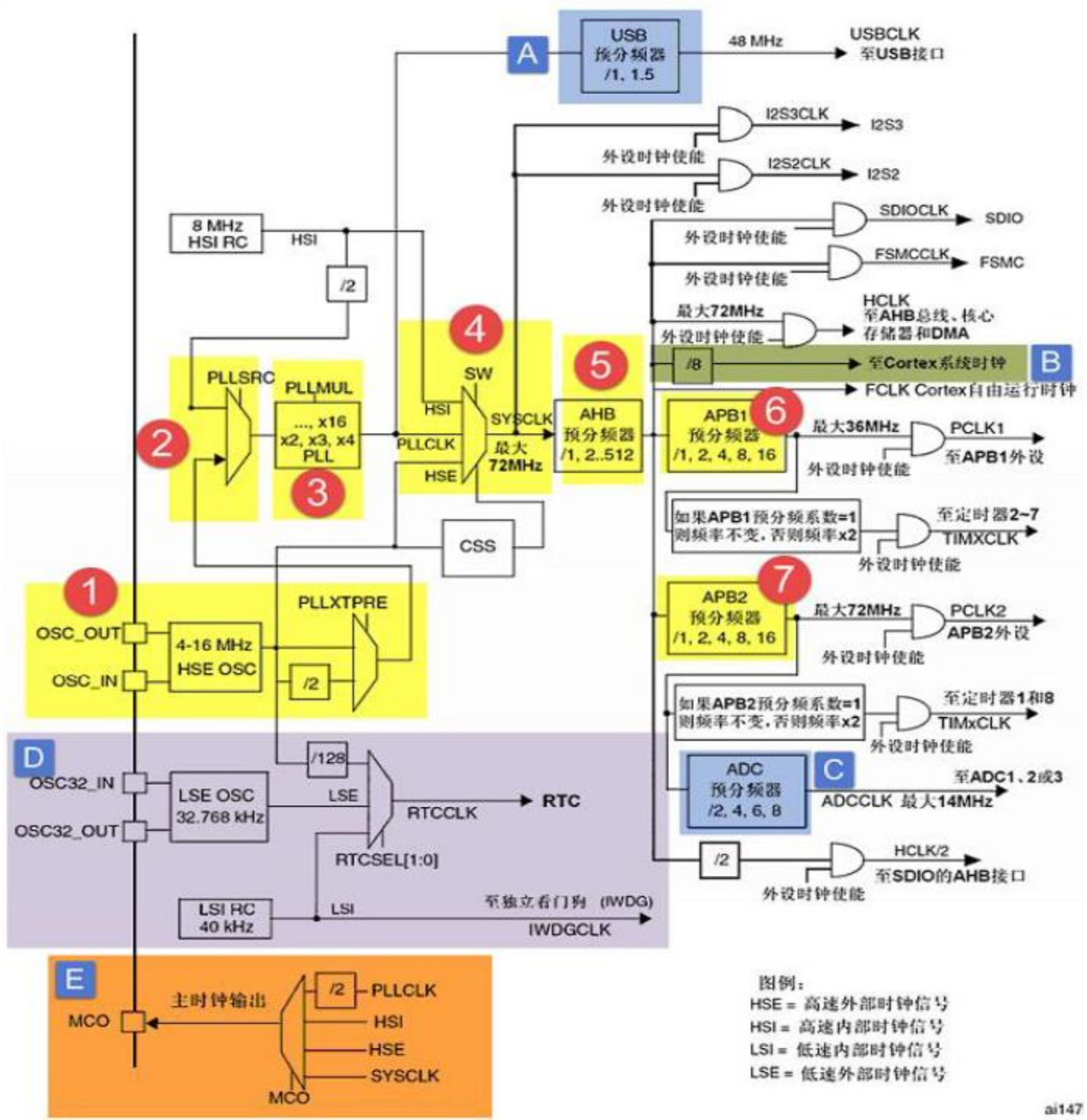
- 设置系统时钟SYSCLK
- 设置AHB 分频因子（决定HCLK 等于多少）
- 设置APB2 分频因子（决定PCLK2 等于多少）
- 设置APB1 分频因子（决定PCLK1 等于多少）
- 设置各个外设的分频因子
- 控制AHB、APB2 和APB1 这三条总线时钟的开启、控制每个外设的时钟的开启

常用配置：

PCLK2 =HCLK = SYSCLK=PLLCLK = 72M

PCLK1=HCLK/2 = 36M

RCC时钟框图


A
USB
预分频器
/1, 1.5
48 MHz
USBCLK
至USB接口
I2S3CLK
I2S3
外设时钟使能
I2S2CLK
I2S2
外设时钟使能
SDIOCLK
SDIO
外设时钟使能
FSMCCLK
FSMC
外设时钟使能
8 MHz
HSI RC
HSI
/2
4
SW
HSI
SYSCLK
PLLCLK
HSE
最大
72MHz
5
AHB
预分频器
/1, 2..512
最大72MHz
外设时钟使能
HCLK
至AHB总线、核心
存储器和DMA
/8
至Cortex系统时钟
B
FCLK Cortex自由运行时钟
PLLSRC
PLLMUL
..., x16
x2, x3, x4
PLL
2
3
APB1
预分频器
/1, 2, 4, 8, 16
6
最大36MHz
PCLK1
至APB1外设
外设时钟使能
如果APB1预分频系数=1
则频率不变，否则频率x2
至定时器2~7
TIMXCLK
外设时钟使能
CSS
1
OSC_OUT
OSC_IN
4-16 MHz
HSE OSC
PLLXTPRE
/2
APB2
预分频器
/1, 2, 4, 8, 16
7
最大72MHz
PCLK2
APB2外设
外设时钟使能
如果APB2预分频系数=1
则频率不变，否则频率x2
至定时器1和8
TIMxCLK
外设时钟使能
D
/128
OSC32_IN
OSC32_OUT
LSE OSC
32.768 kHz
LSE
RTCCLK
RTC
RTCSEL[1:0]
ADC
预分频器
/2, 4, 6, 8
C
至ADC1、2或3
ADCCLK 最大14MHz
/2
HCLK/2
至SDIO的AHB接口
外设时钟使能
LSI RC
40 kHz
LSI
至独立看门狗 (IWDG)
IWDGCLK
E
主时钟输出
MCO
/2
PLLCLK
HSI
HSE
SYSCLK
MCO
图例：
HSE = 高速外部时钟信号
HSI = 高速内部时钟信号
LSI = 低速内部时钟信号
LSE = 低速外部时钟信号
ai147

中断源与中断屏蔽

中断源

中断源：能引发中断的事件。通常，中断源都与外设有关

- 在生活中，门铃的铃声是一个中断源，它由门铃这个外设发出。
- 在嵌入式系统中，常见的中断源有按键按下和释放、定时器溢出、串口收到数据等，与此相关的外设有键盘、定时器和串口等。

中断请求标志位：每个中断源都有它对应的中断标志位

- 一旦该中断发生，它的中断标志位就会被置位。如果中断标志位被清除，那么它所对应的中断便不会再被响应。因此，一般在中断服务程序最后要将对应的中断标志位清零。

中断处理过程

中断屏蔽

可以通过设置相应的中断屏蔽位，禁止CPU响应某个中断，从而实现中断屏蔽。

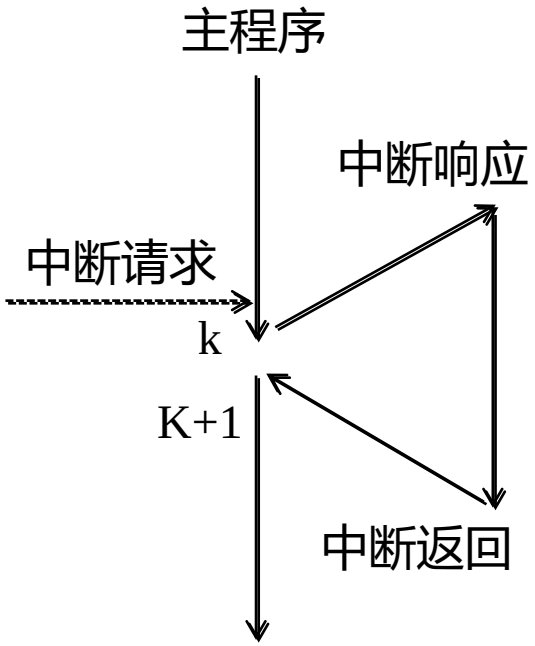
可屏蔽中断

- 在生活中，门铃的铃声是一个可屏蔽中断。

不可屏蔽中断

- 在计算机系统中，电源故障、内存出错、总线出错等是不可屏蔽中断。

中断源与中断屏蔽



- **中断响应（硬件自动实现）**

✓ 保护现场

✓ 找到该中断对应的中断服务程序的地址——中断向量表

- **执行中断服务程序（用户编程）**

中断服务程序通常是由用户使用C语言编写的特殊函数，用来实现对该中断真正的处理操作，具有以下特点：

✓ 中断服务程序既没有参数，也没有返回值，更不由用户调用，而是当某个事件产生一个中断时由硬件自动调用。

✓ 在中断服务程序中修改、在其他程序中访问的变量，在其定义和声明时要在前面加上volatile修饰词。

✓ 中断服务程序要求应当尽量的简短。

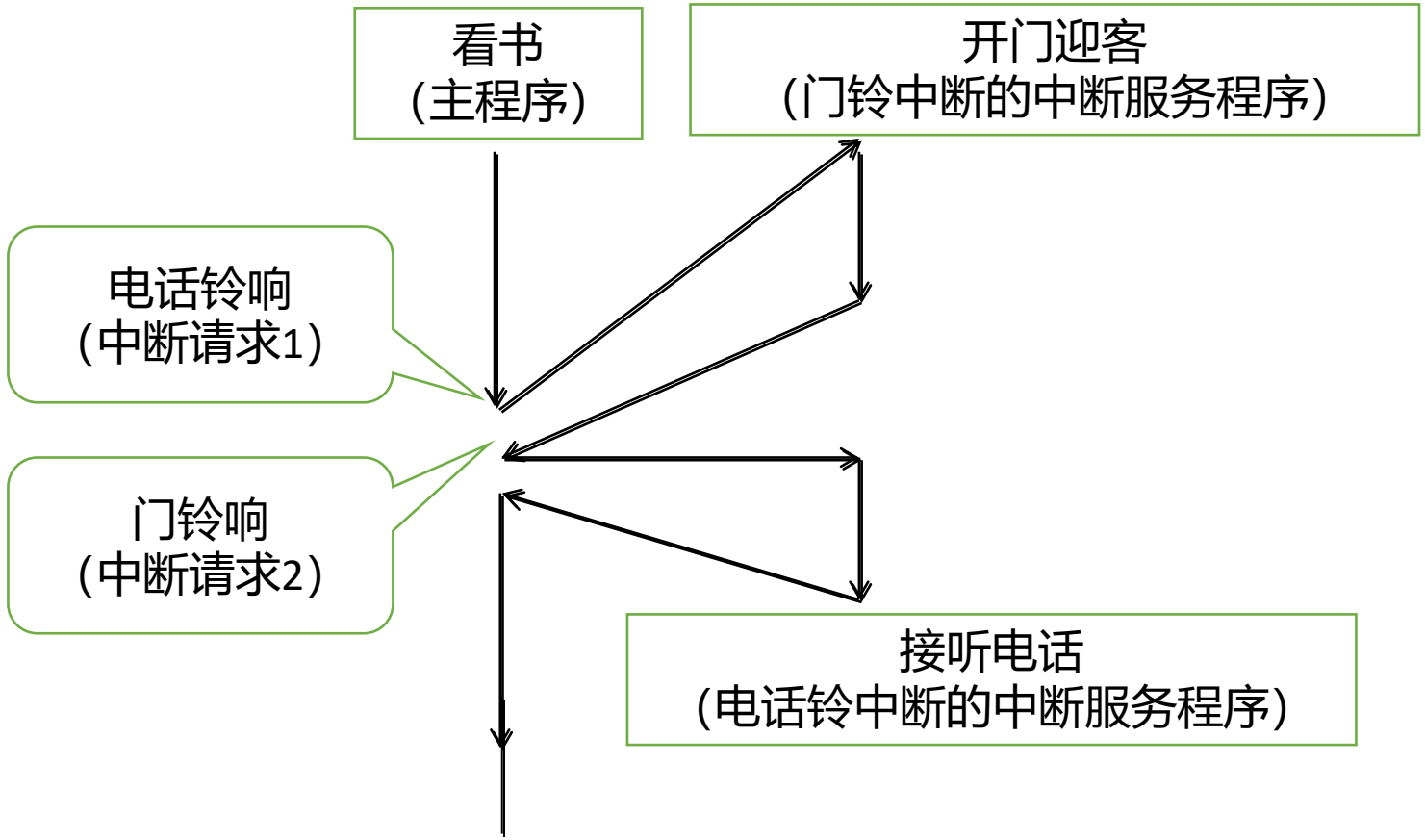
- **中断返回（硬件自动实现）**

恢复现场

中断优先级与中断嵌套

中断优先级：

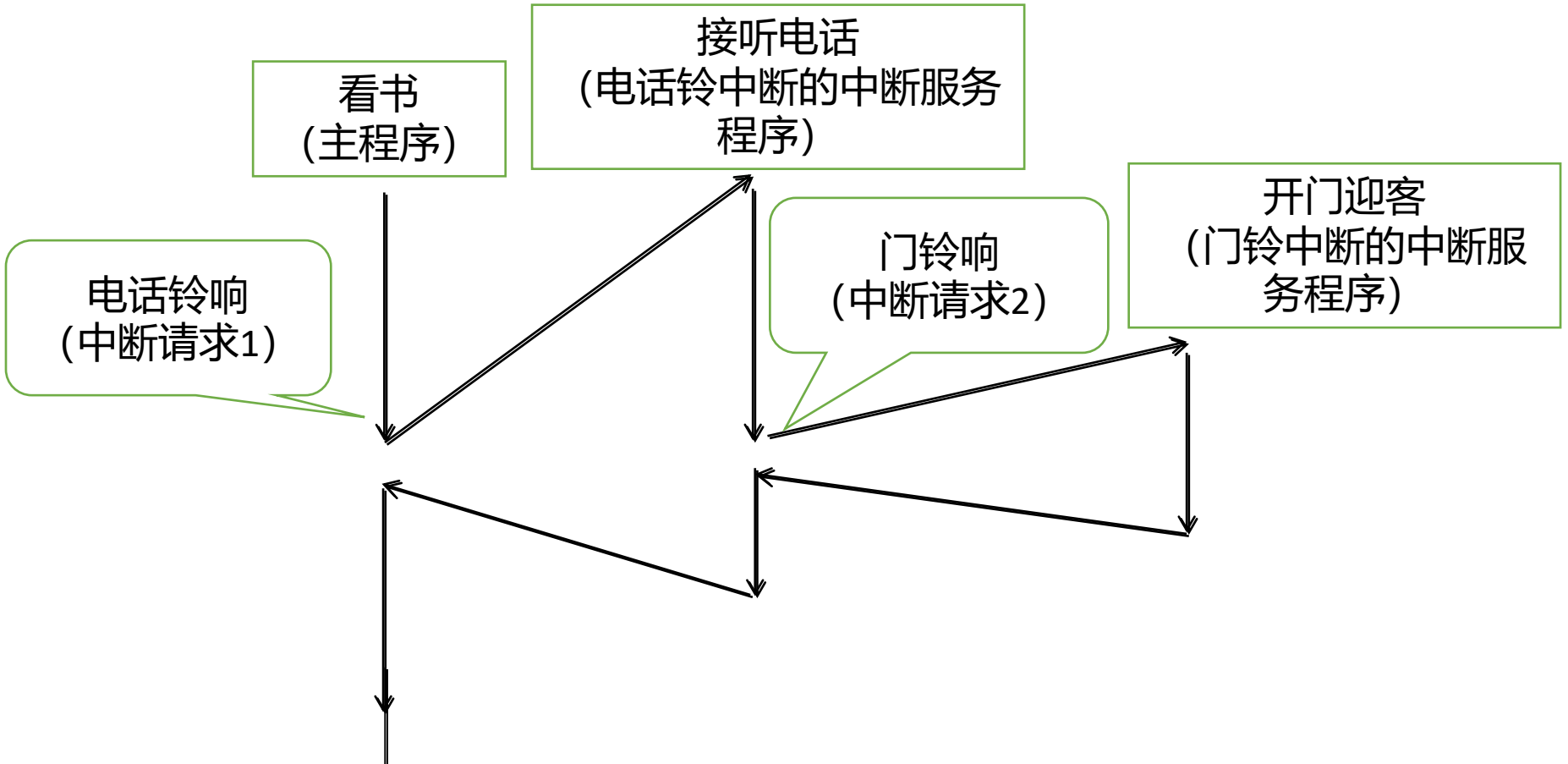
嵌入式系统中的中断往往不止一个，那么，对于多个同时发生的中断或者嵌套发生的中断，CPU又该如何处理？应该先响应哪一个中断？答案就是**中断优先级**



中断优先级与中断嵌套

中断嵌套：

在嵌入式系统中，中断嵌套是指当系统正在执行一个中断服务时，又有新的中断事件发生而产生了新的中断请求



嵌套向量中断控制器NVIC

NVIC优先级定义

- NVIC 有中断优先级寄存器NVIC_IPRx来配置外部中断的优先级
- IPR 宽度为8bit，原则上每个外部中断可配置的优先级为0~255
- 数值越小，优先级越高
- F103中，只使用了高4位

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
用于表达优先级				保留			

STM32F103中断系统

STM32F103中断优先级

1.抢占优先级（Preempting Priority）

高抢占式优先级的中断事件会打断当前的主程序/中断程序运行，俗称中断嵌套。

2. 何为响应优先级(subpriority)

在抢占式优先级相同的情况下，高响应优先级的中断优先被响应。

3. 判断中断是否会被响应的依据

首先是抢占式优先级，其次是响应优先级；抢占式优先级决定是否会有中断嵌套。

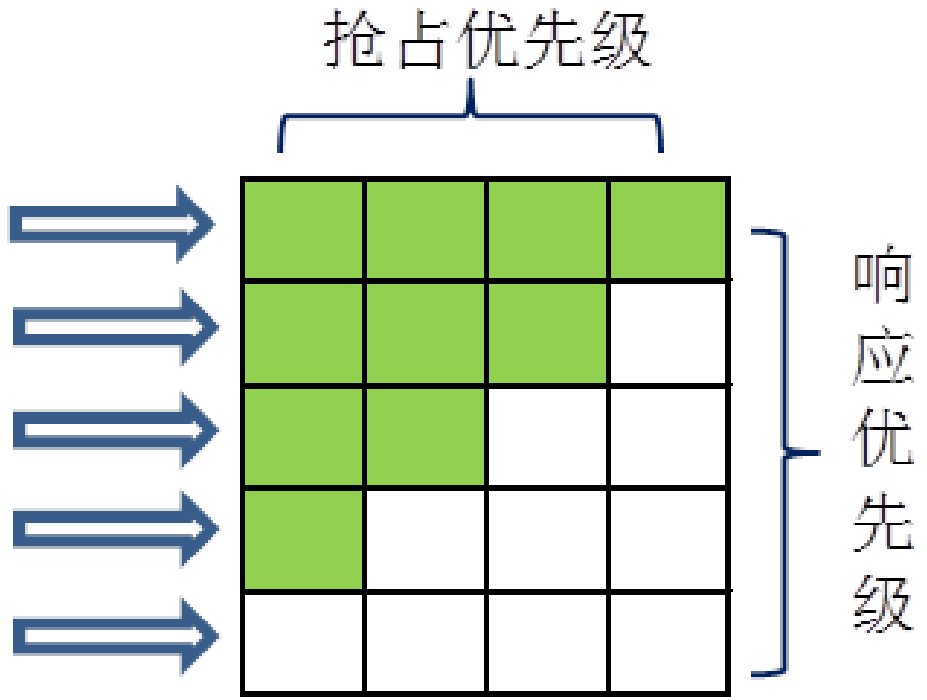
4. 优先级冲突的处理

具有高抢占式优先级的中断可以在具有低抢占式优先级的中断处理过程中被响应，即中断的嵌套，或者说高抢占式优先级的中断可以嵌套低抢占式优先级的中断。

STM32F103中断系统

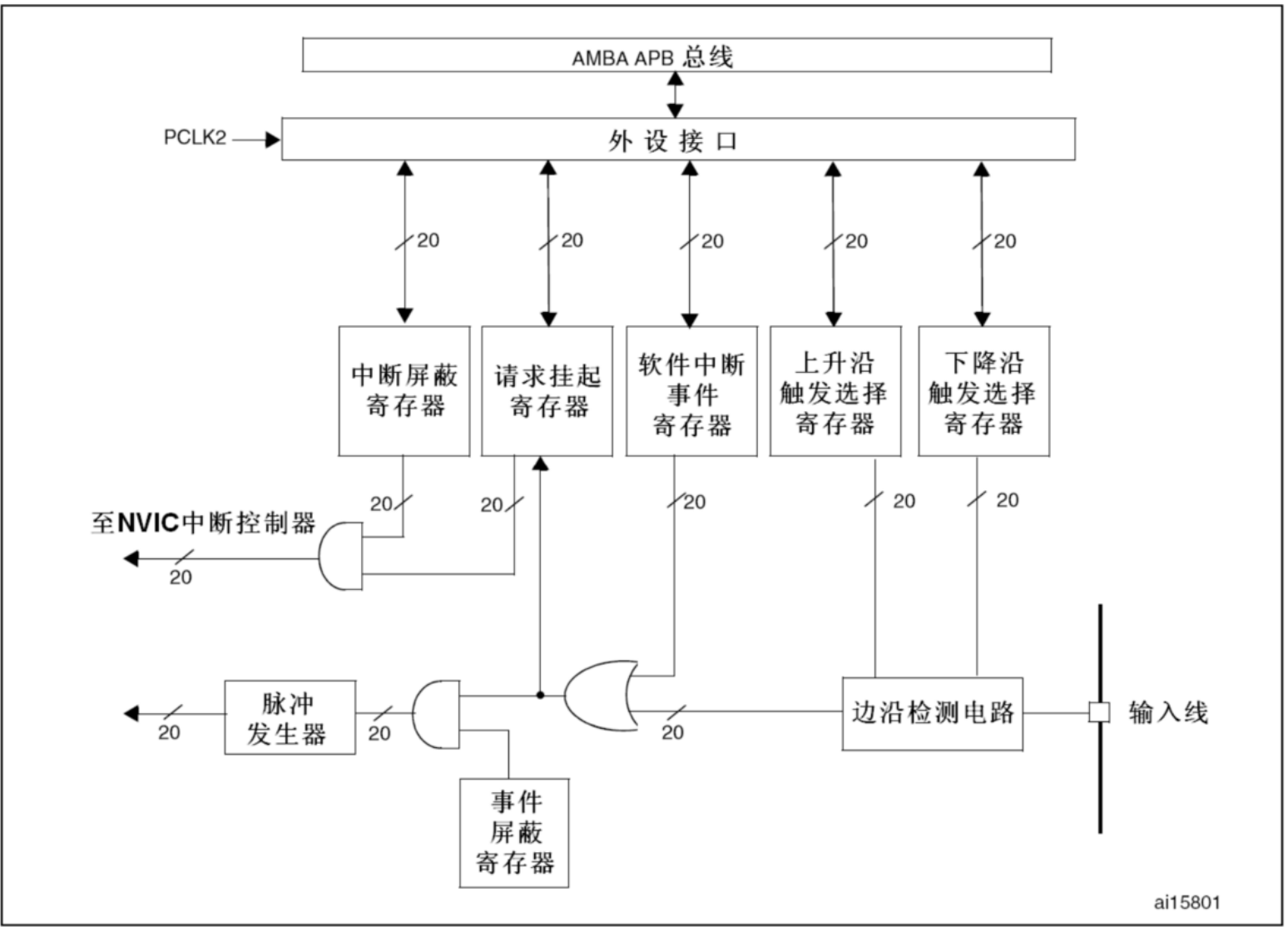
STM32F103中断优先级

优先级组别	抢占式优先级		响应式优先级	
	位数	级数	位数	级数
4组	4	16	0	0
3组	3	8	1	2
2组	2	4	2	4
1组	1	2	3	8
0组	0	0	4	16

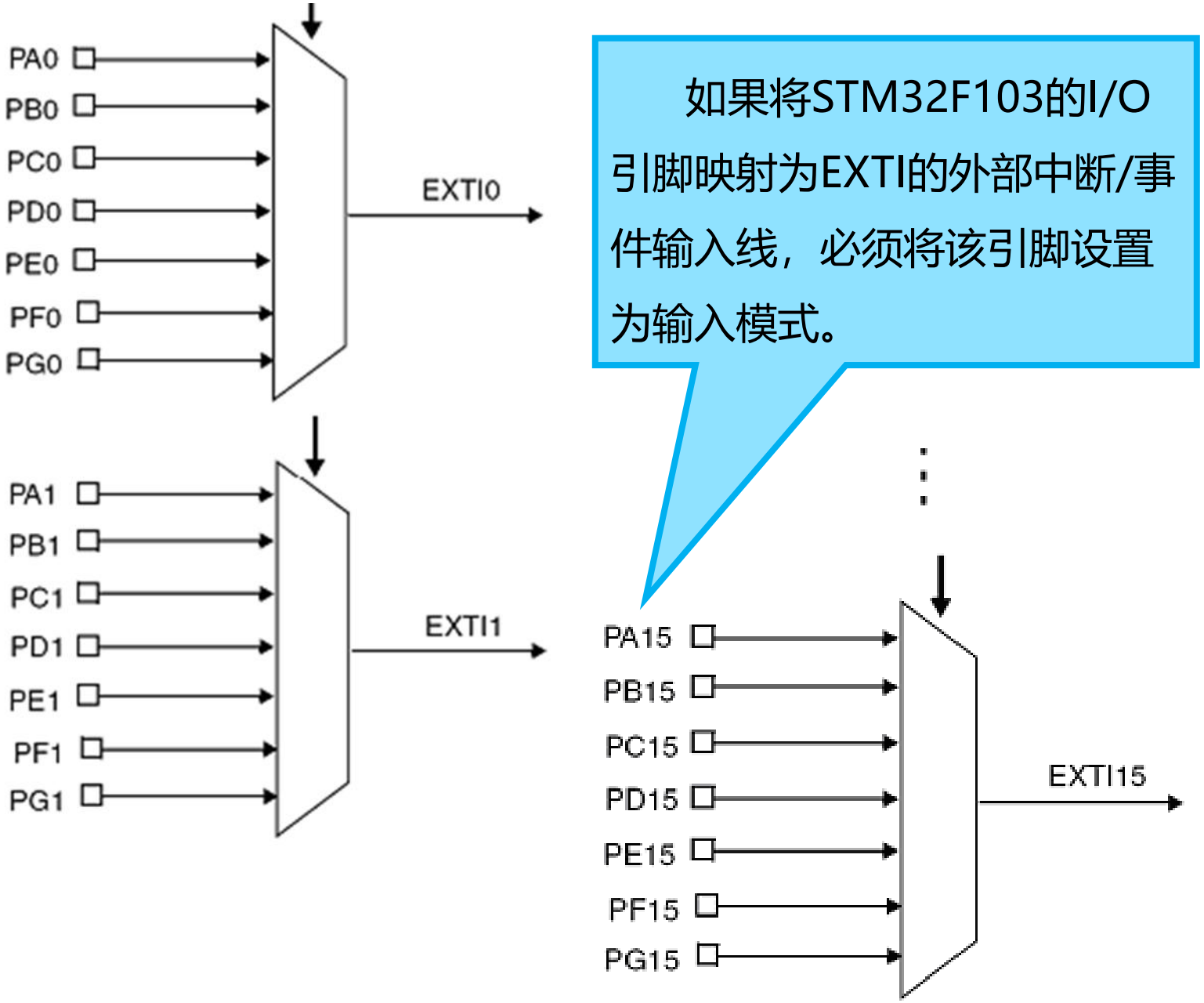


STM32F103外部中断/事件控制器EXTI

EXTI内部结构



STM32F103外部中断/事件控制器EXT1内部信号线上画有一条斜线，旁边标有20，表示这样的线路共有20套


PA0
PB0
PC0
PD0
PE0
PF0
PG0
EXTI0
PA1
PB1
PC1
PD1
PE1
PF1
PG1
EXTI1
PA15
PB15
PC15
PD15
PE15
PF15
PG15
EXTI15
如果将STM32F103的I/O引脚映射为EXTI的外部中断/事件输入线，必须将该引脚设置为输入模式。

STM32F103外部中断/事件控制器EXTI

APB外设接口：

APB外设模块接口是STM32F103微控制器每个功能模块都有的部分，CPU通过这样的接口访问各个功能模块。

尤其需要注意的是，如果使用STM32F103引脚的外部中断/事件映射功能，**必须打开APB2总线上该引脚对应端口的时钟以及AFIO功能时钟。**

边沿检测器：

EXTI中的边沿检测器共有20个，用来连接20个外部中断/事件输入线，是EXTI的主体部分。

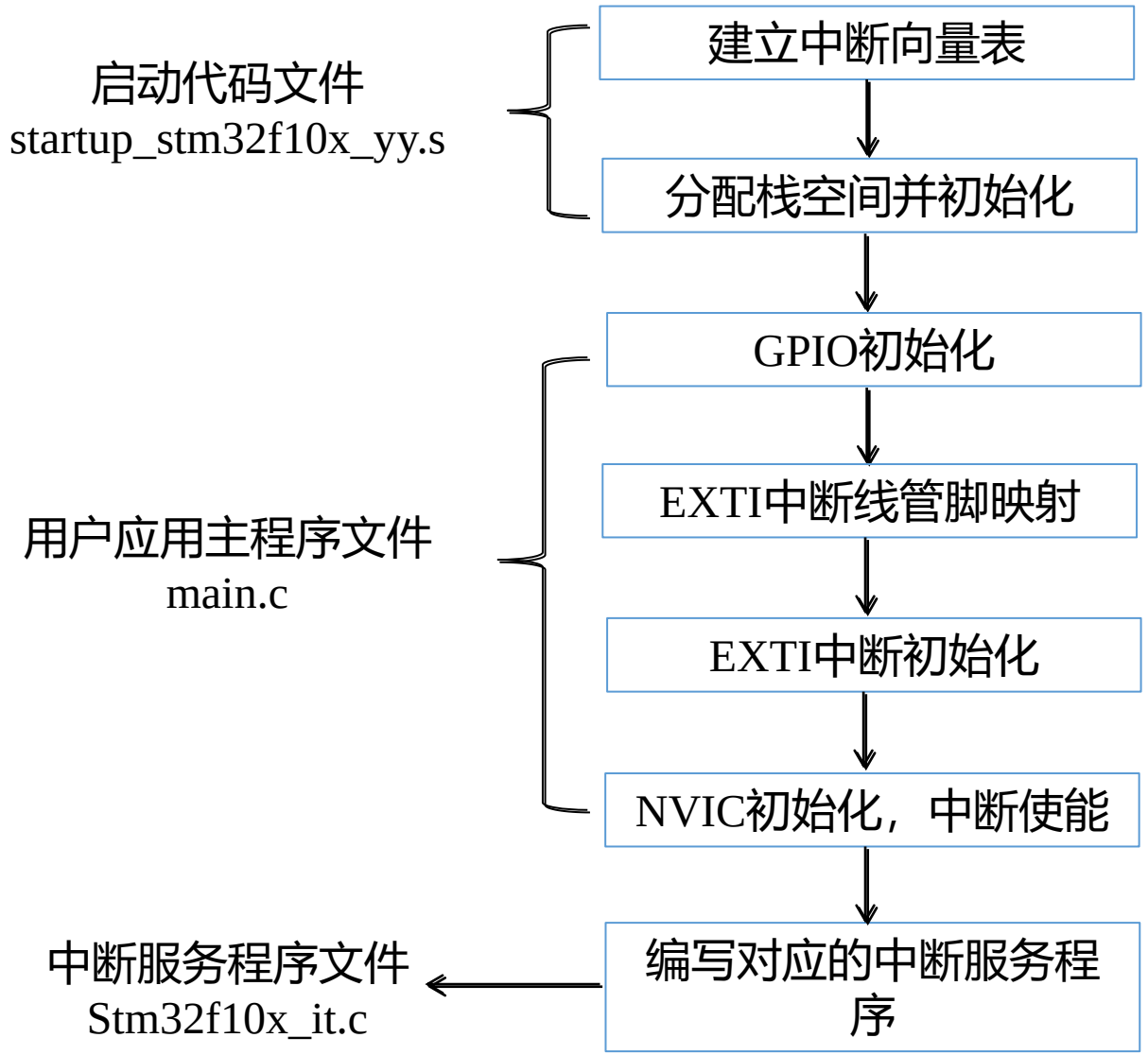
每个边沿检测器由边沿检测电路、控制寄存器、门电路和脉冲发生器等部分组成

嵌套向量中断控制器NVIC

NVIC库函数

NVIC库函数	描述
void NVIC_EnableIRQ(IRQn_Type IRQn)	使能中断
void NVIC_DisableIRQ(IRQn_Type IRQn)	失能中断
void NVIC_SetPendingIRQ(IRQn_Type IRQn)	设置中断悬起位
void NVIC_ClearPendingIRQ(IRQn_Type IRQn)	清除中断悬起位
uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn)	获取悬起中断编号
void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)	设置中断优先级
uint32_t NVIC_GetPriority(IRQn_Type IRQn)	获取中断优先级
void NVIC_SystemReset(void)	系统复位

EXTI常用流程



外部中断建立过程

作业2-外部中断控制实验

▸结合“实验参考20241017.pdf”内容进行实验，实现：

1、参照实验参考20241017.pdf P154完成中断对LED等的控制实验；

2、对代码进行修改，分别实现上升沿、下降沿或双边沿触发；

3、修改中断服务函数为占用时间较长的任务（如某一色的灯亮多少秒之后再熄灭），之后修改按键1和按键2为不同的抢占优先级和响应优先级，验证是否会出现中断抢占。

4、结合之前交通灯代码，用按键中断方式实现交通灯运行/暂停控制。

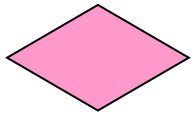
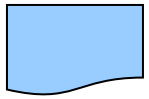
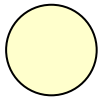
报告请统一命名为“嵌入式系统设计实验-实验2-第Y组”
在10月23日23:00前发送到1043934501@qq.com
本次作业需要提交：报告（报告最后部分注明小组内各自完成的工作内容）、四个子任务的源码、四个任务的演示视频（视频中由队员对自己完成部分内容进行讲解）

软件流程图

流程图

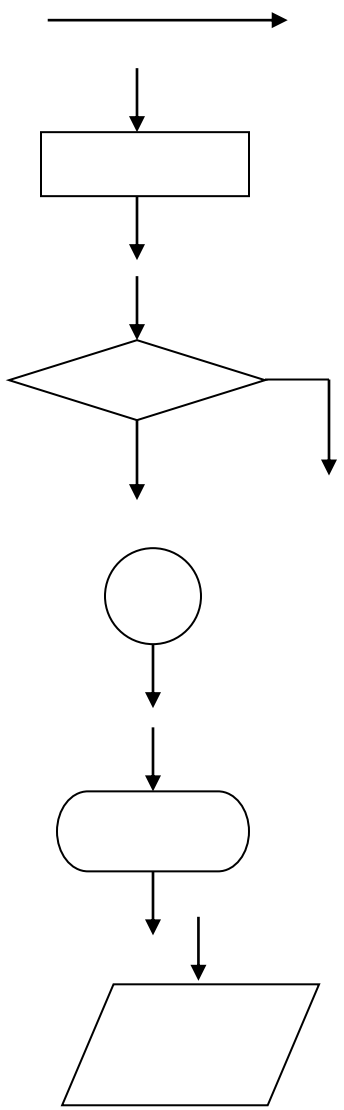
- 流程图是以简单的图标符号来表达问题解决步骤示意图
- 在实际工作中，常需要向别人介绍某项工作或程序的操作流程
- 若是稍复杂的工作流程，仅用文字难以表达清楚
- 应充分利用可视化技术，将复杂的工作流程用图形化的方式表达出来，不仅使你表达容易，而且让别人也更容易理解
- 流程图的绘制必须使用标准的流程图符号，并遵守流程图绘制的相关规定，才能绘制出正确而清楚的流程图。

流程图-部分常用符号

符 号	名称	含 义
	端点、中断	标准流程的开始与结束， 每一流程图只有一个起点
	处理	要执行的处理
	判断	决策或判断
	文档	以文件的方式输入/输出
	流向	表示执行的方向与顺序
	数据	表示数据的输入/输出
	联系	同一流程图中从一个进程到另一个进程的交叉引用

流程图-部分常用符号

参考：GB 1526—89



流程线：有向线段，指出流程控制方向。

处理框：框中指出要处理的内容。
通常有一个入口和一个出口。

判断框：表示分支情况。
四个顶点，通常上面表示入口，视需要用其余两个顶点表示出口。

连接框：连接因写不下而断开的流程线。

开始符
结束符：表示本段算法的开始或结束。

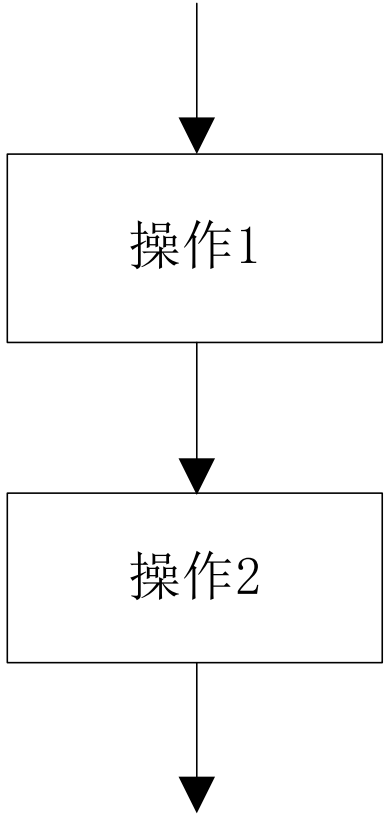
数据：输入,输出

流程图结构说明

顺序执行

含义：操作循序进行。

运用：当有需要顺序执行的操作发生时，绘制图形上下顺序就是程序执行顺序



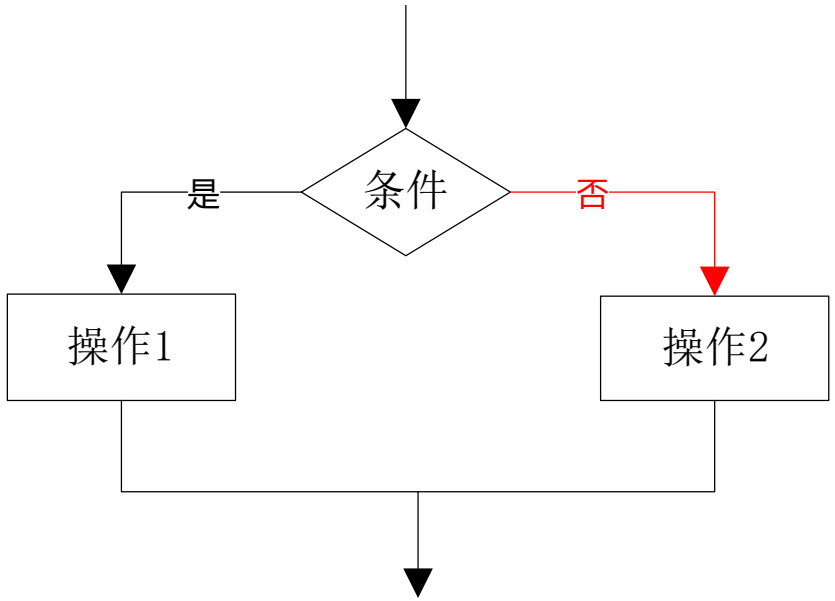
流程图结构说明

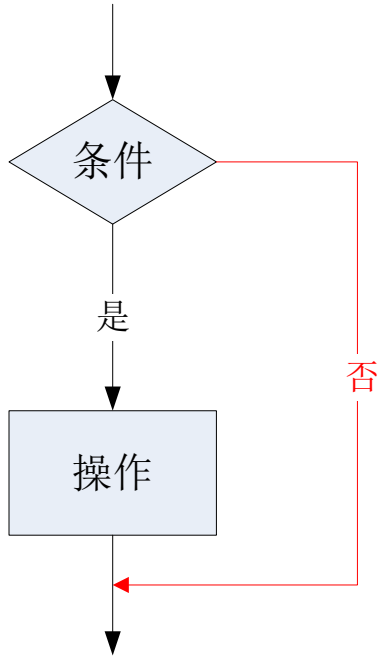
选择结构

二选一结构——基本结构

①含义：依据某些条件，分别执行不同操作。

②语法：IF(条件)——THEN DO 1/ELSE DO 2





流程图结构说明

选择结构

- 多重选择结构——二元选择结构变形

① 含义：依据某些条件，分别执行不同操作。

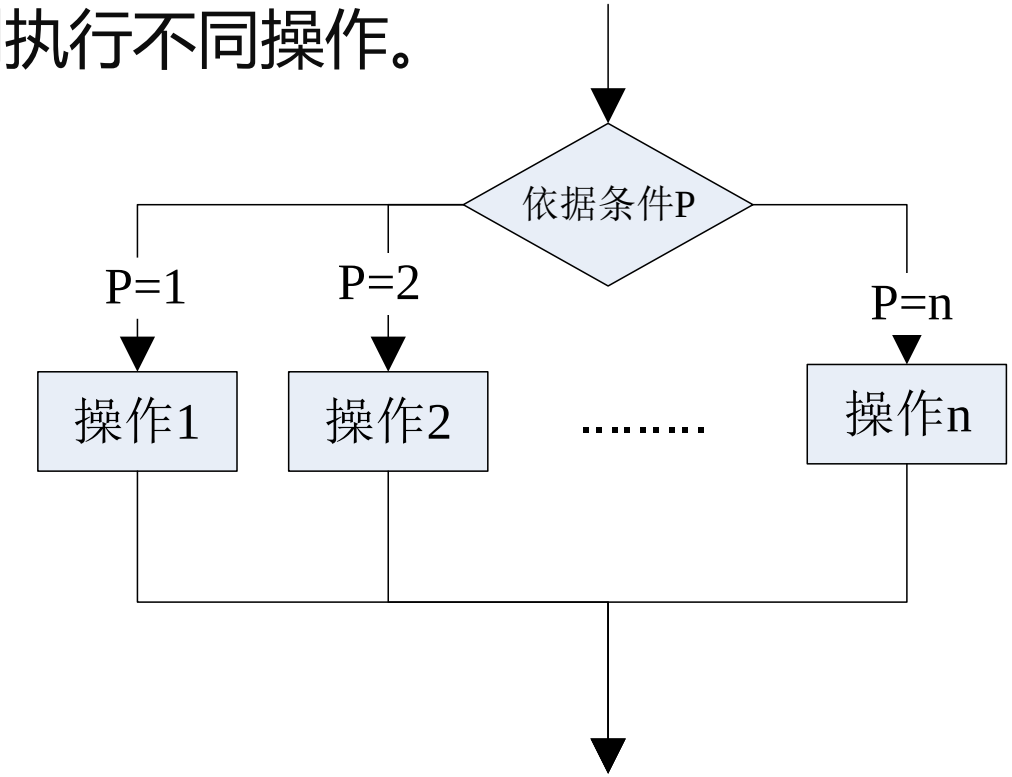
② 语法：switch(条件P)

CASE 1　DO 1

CASE 2　DO 2

…　…

CASE n　DO n



流程图结构说明

重复结构

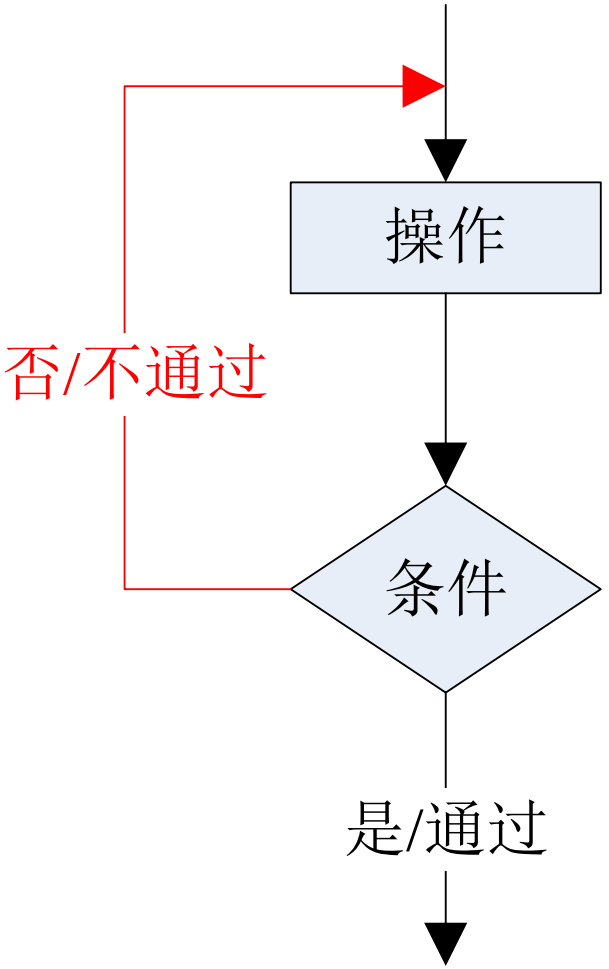
- do - while结构

① 含义：重复执行某操作直到满足某一条件为止即直到条件变成真（True）为止。

② 先执行操作，再判断条件是否继续执行

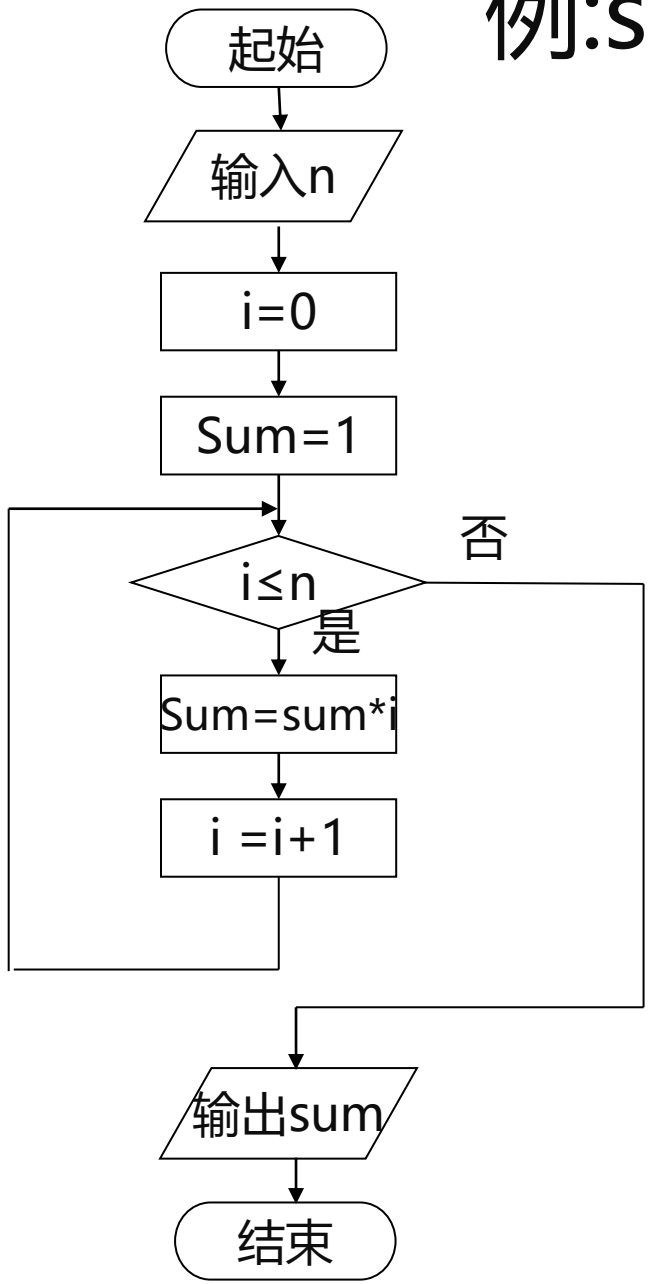
③ 语法：do 进程

while 条件



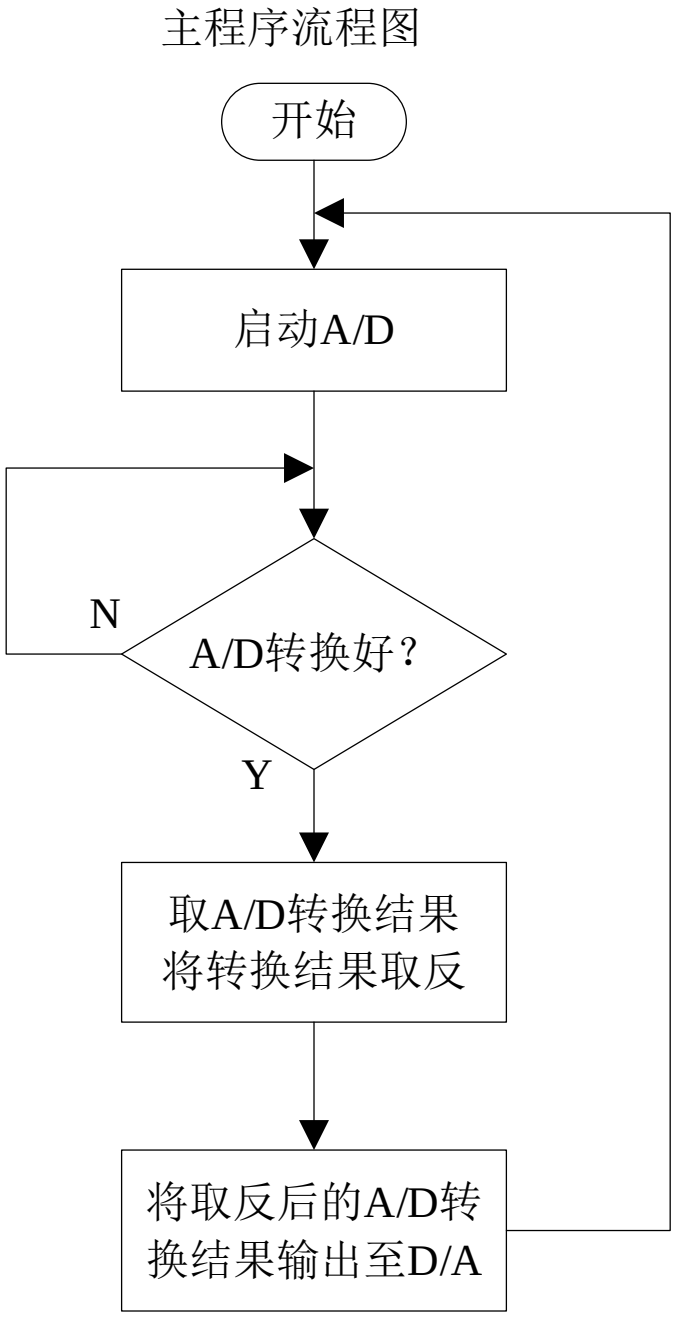
流程图实例

例:sum=1*2*3*...*(n-1)*n



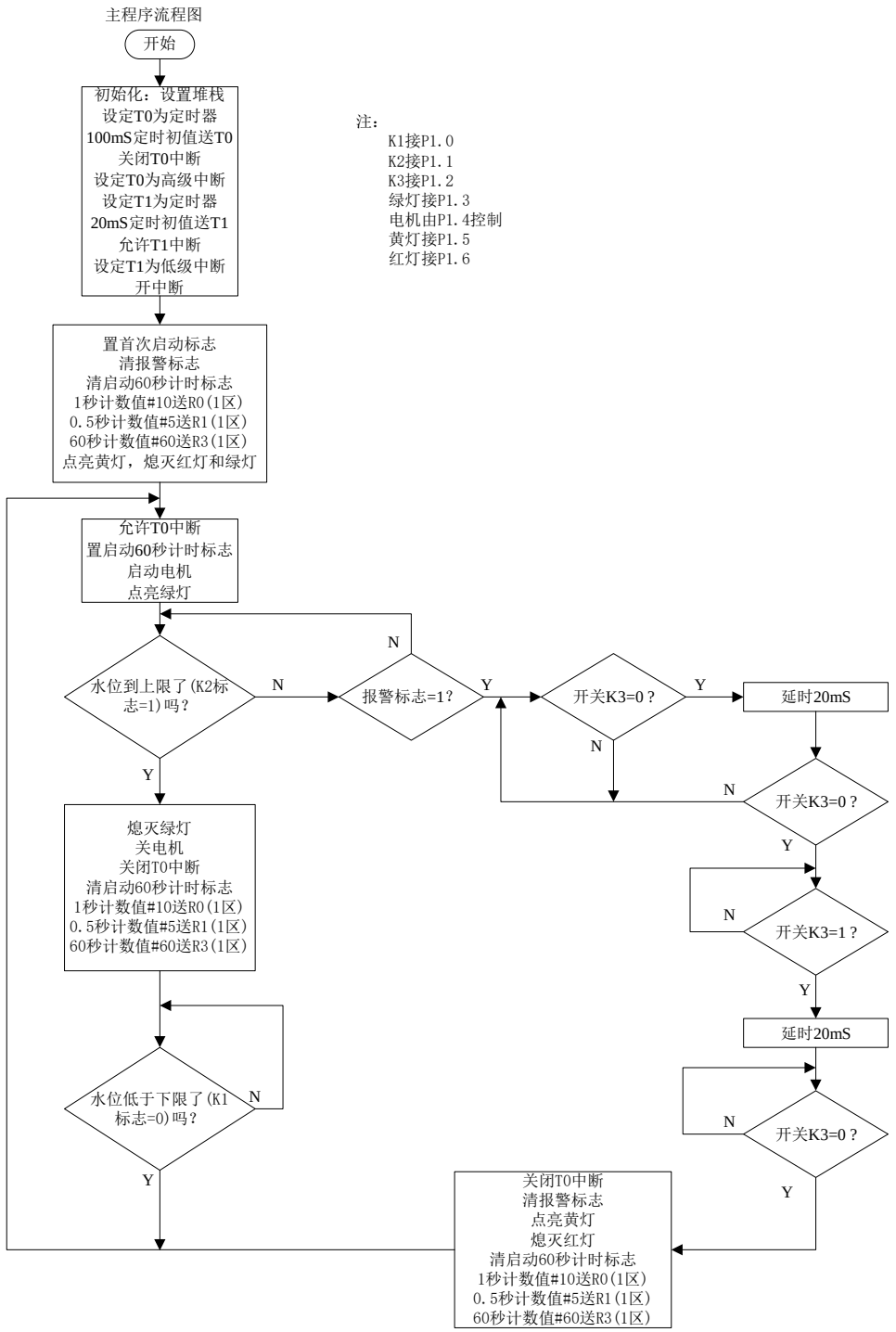
流程图实例

比例电压变换器流程图



流程图实例

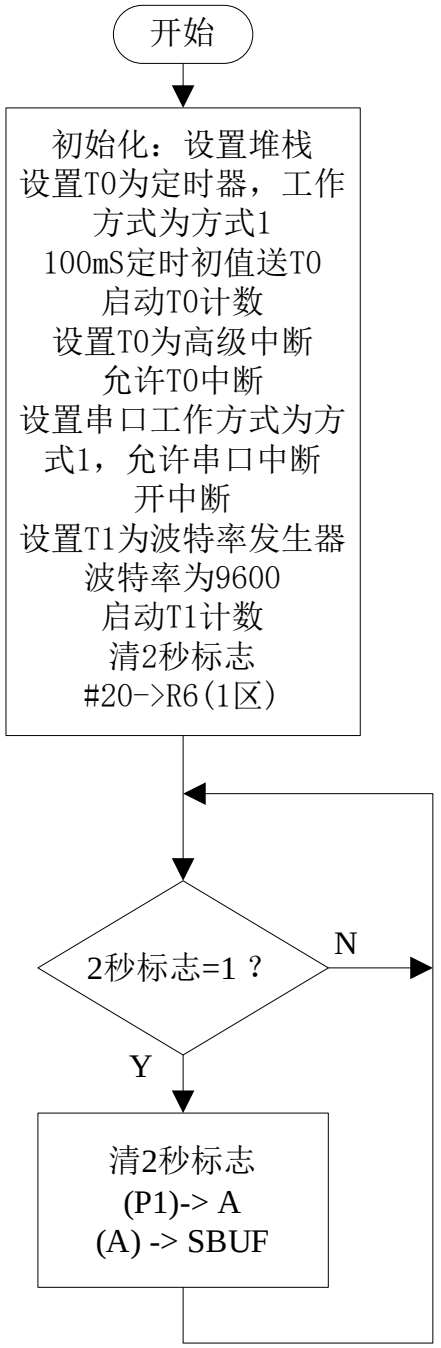
水塔水位控制器流程图



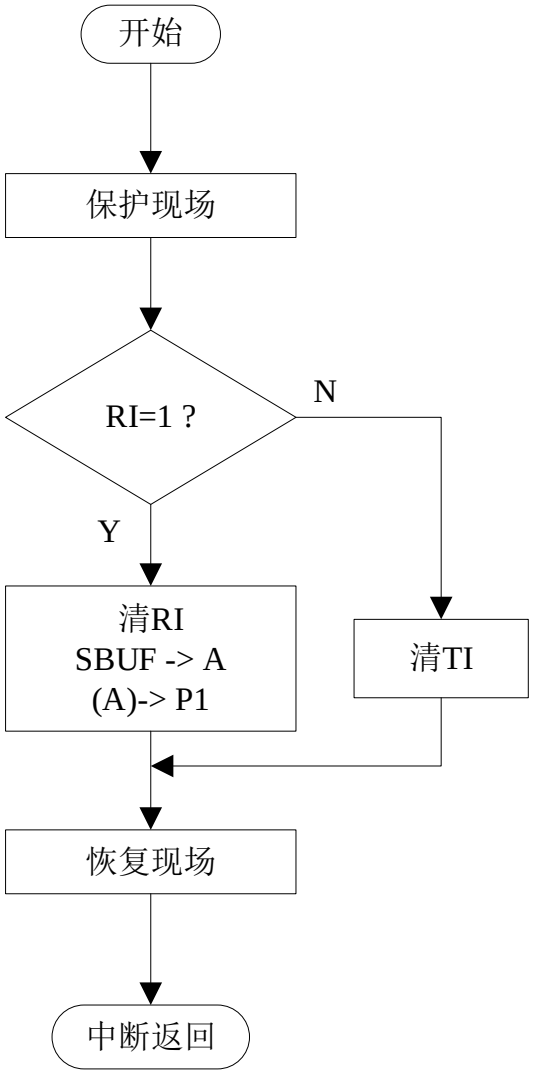
流程图实例

双机串口互通流程图

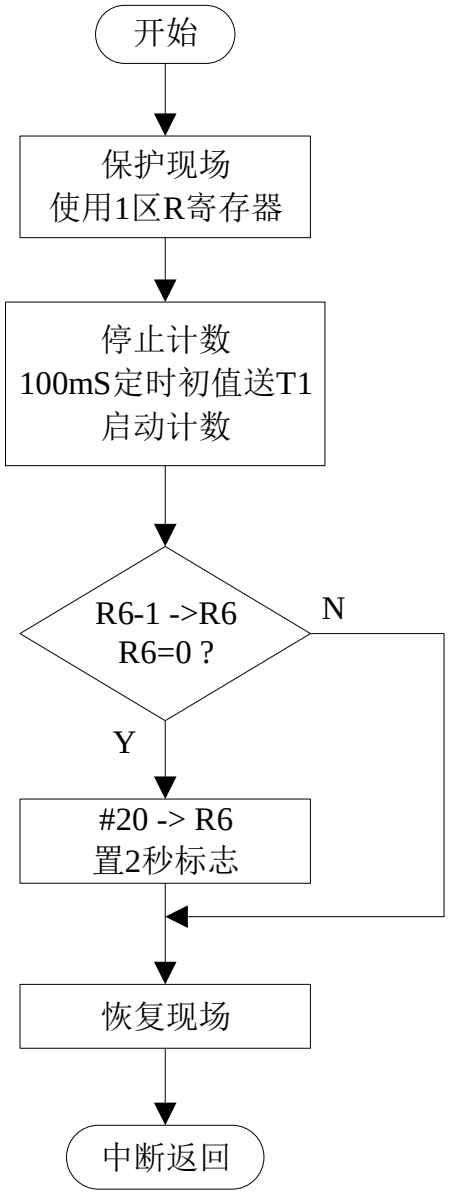
主程序流程图



串口中断服务程序流程图

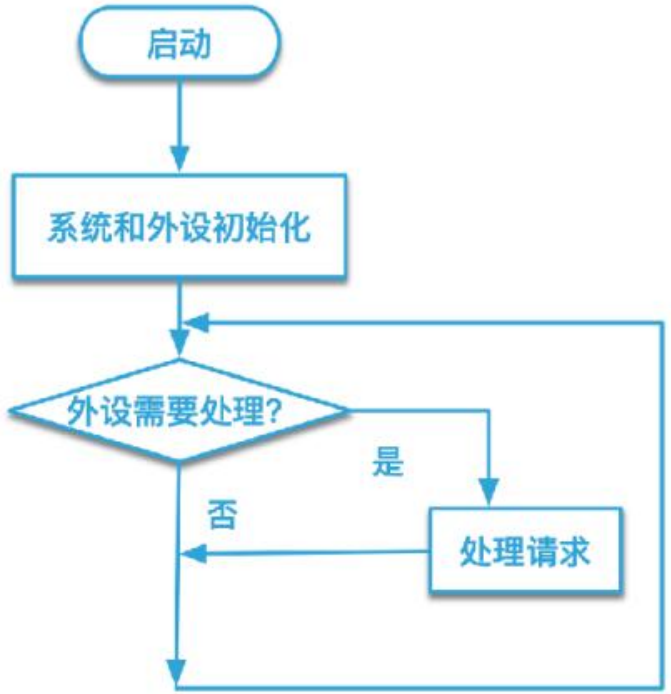


T0中断服务程序流程图

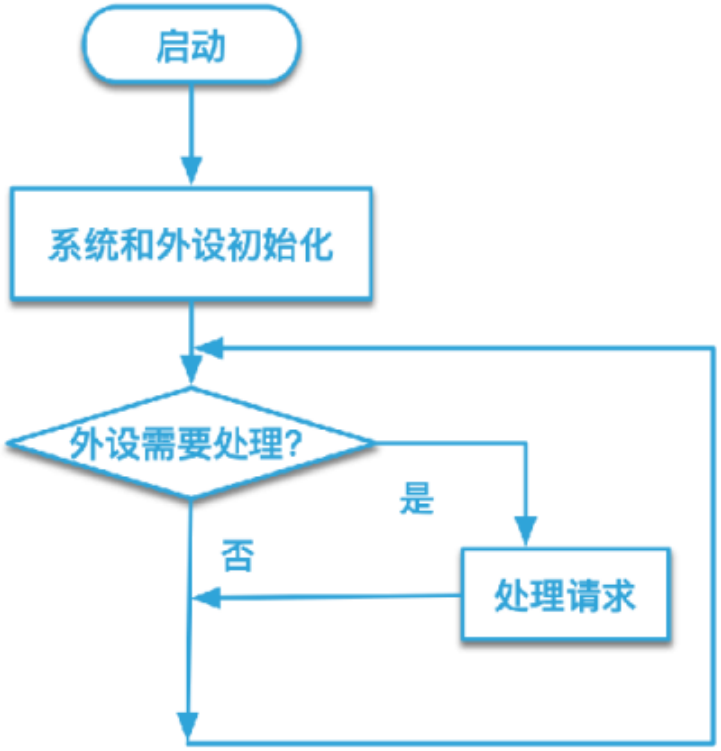


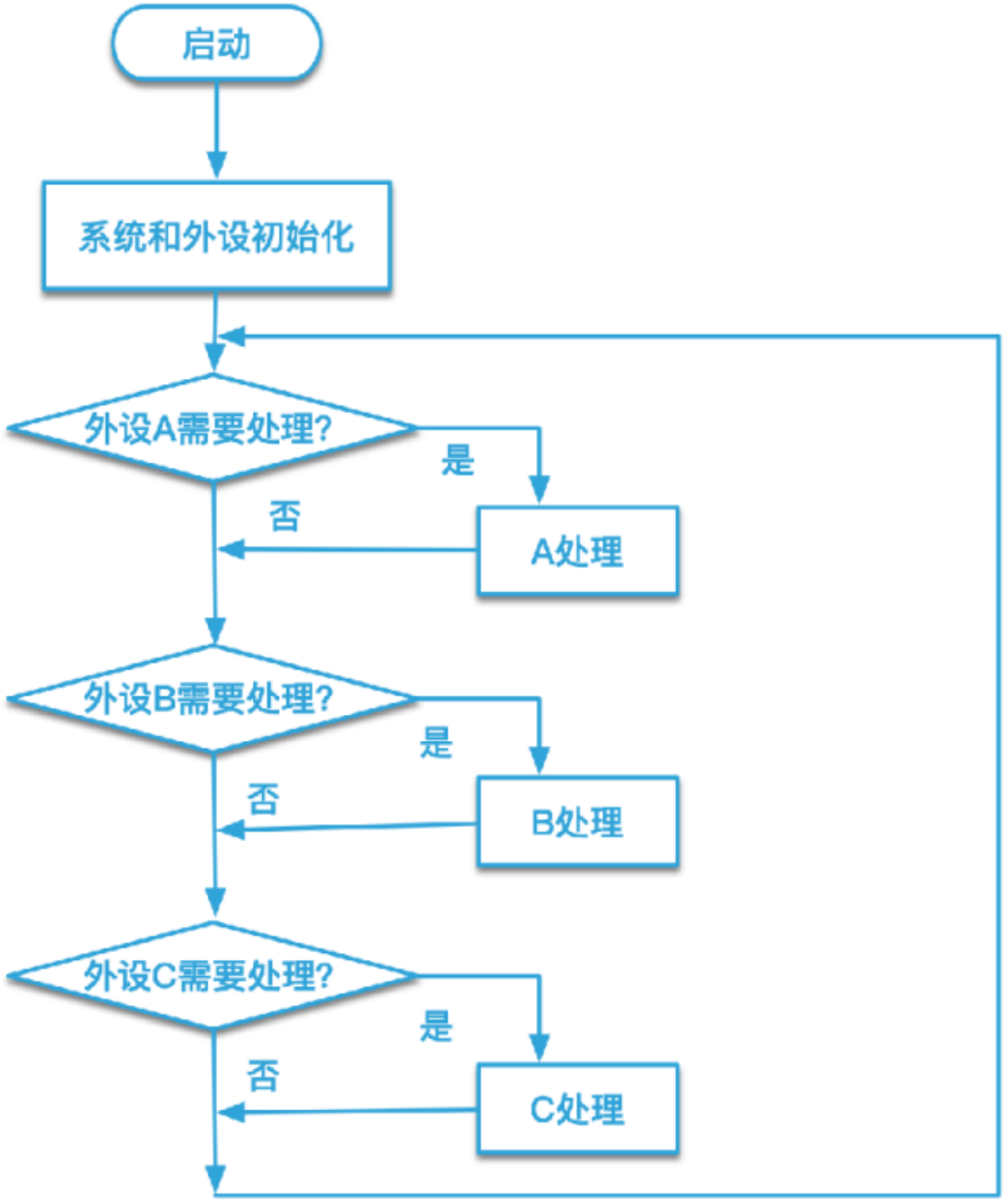
*非stm32

软件运行流程-轮询

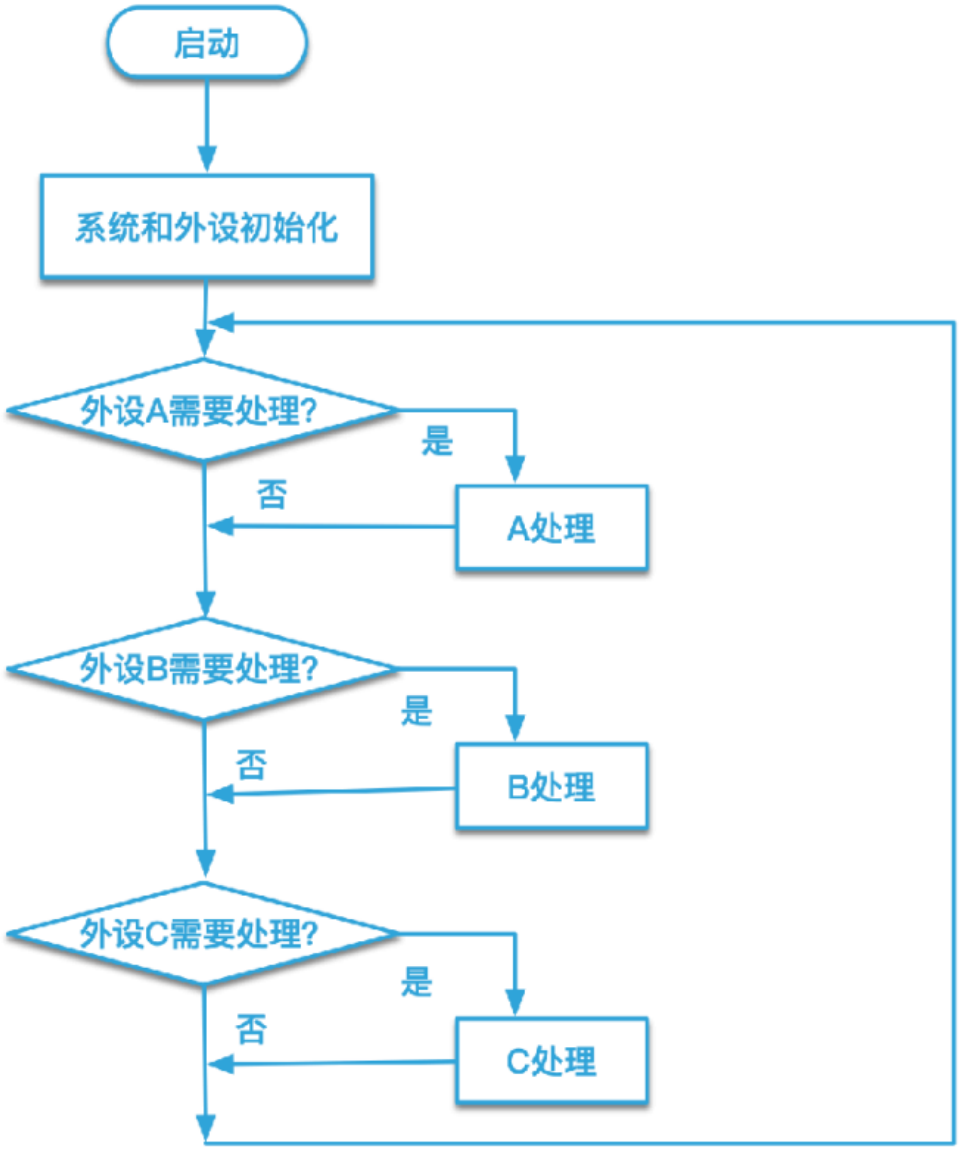


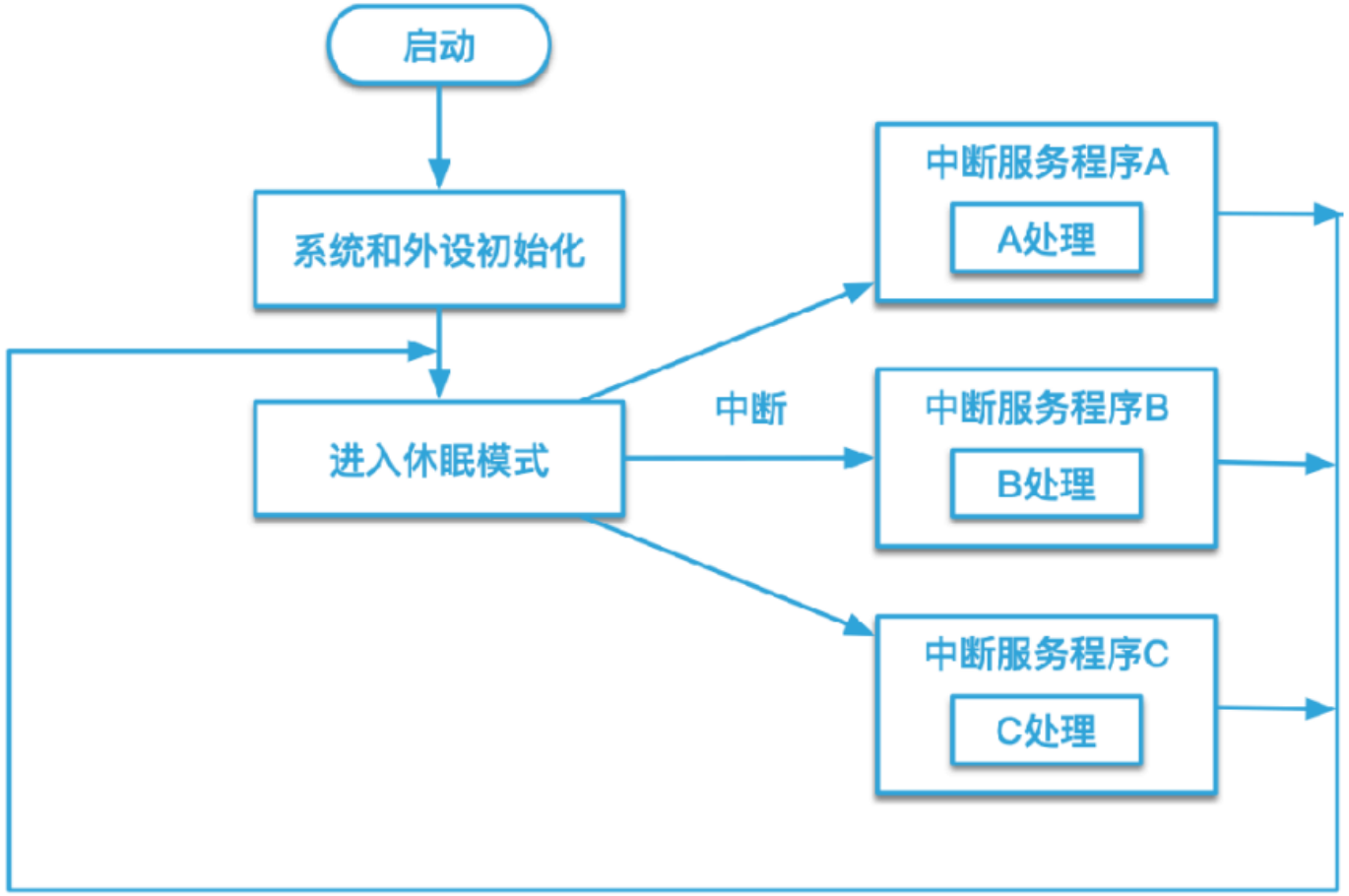
软件运行流程-轮询


启动
系统和外设初始化
外设需要处理?
是
否
处理请求

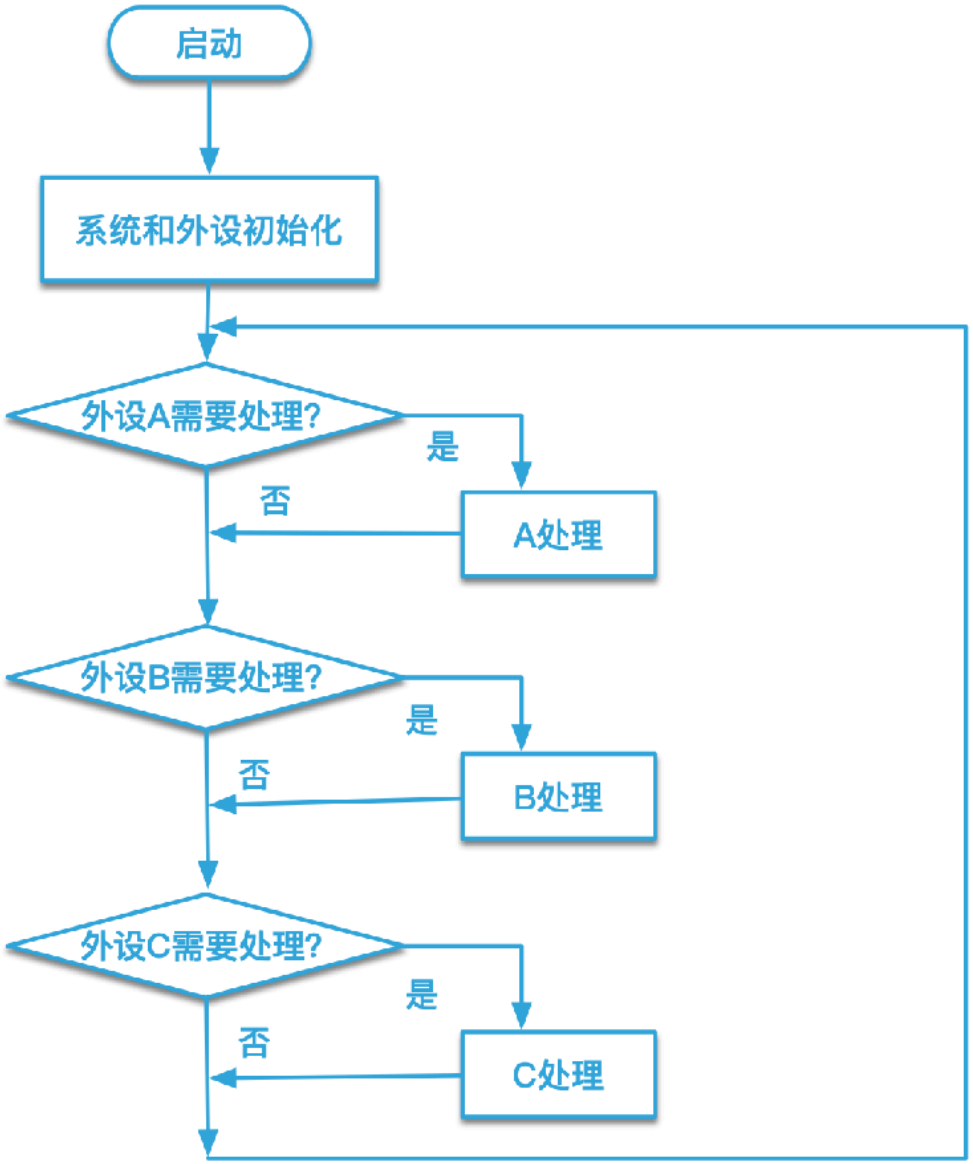

启动
系统和外设初始化
外设A需要处理?
是
否
A处理
外设B需要处理?
是
否
B处理
外设C需要处理?
是
否
C处理

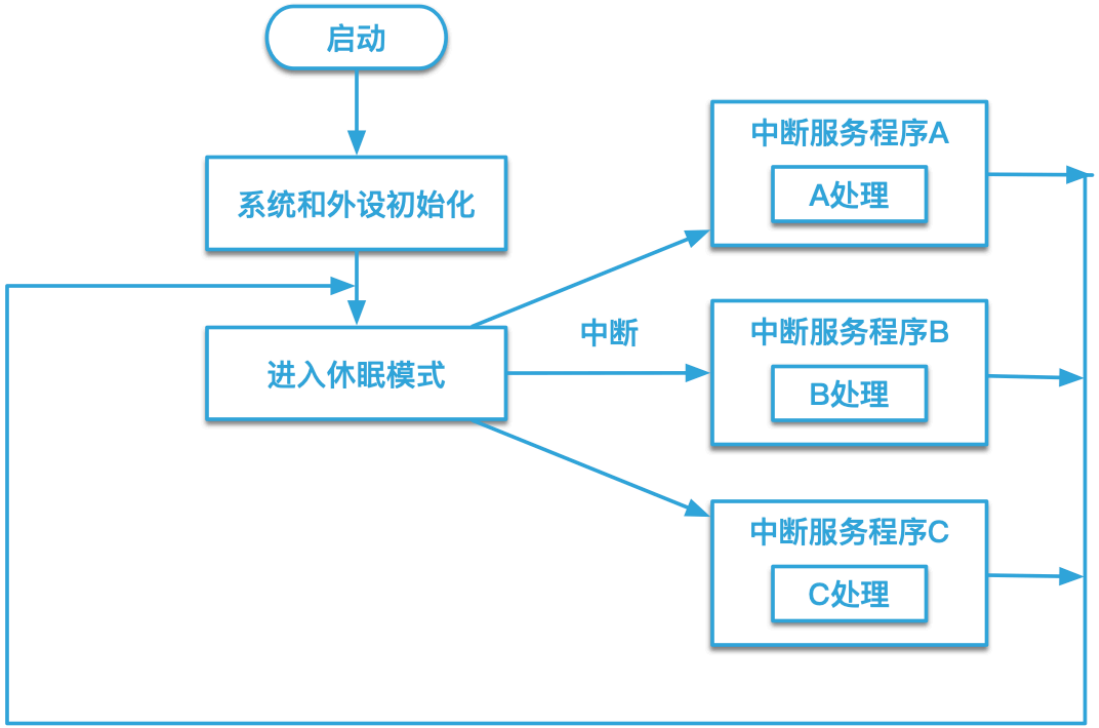
软件运行流程-中断驱动


启动
系统和外设初始化
外设A需要处理?
是
否
A处理
外设B需要处理?
是
否
B处理
外设C需要处理?
是
否
C处理

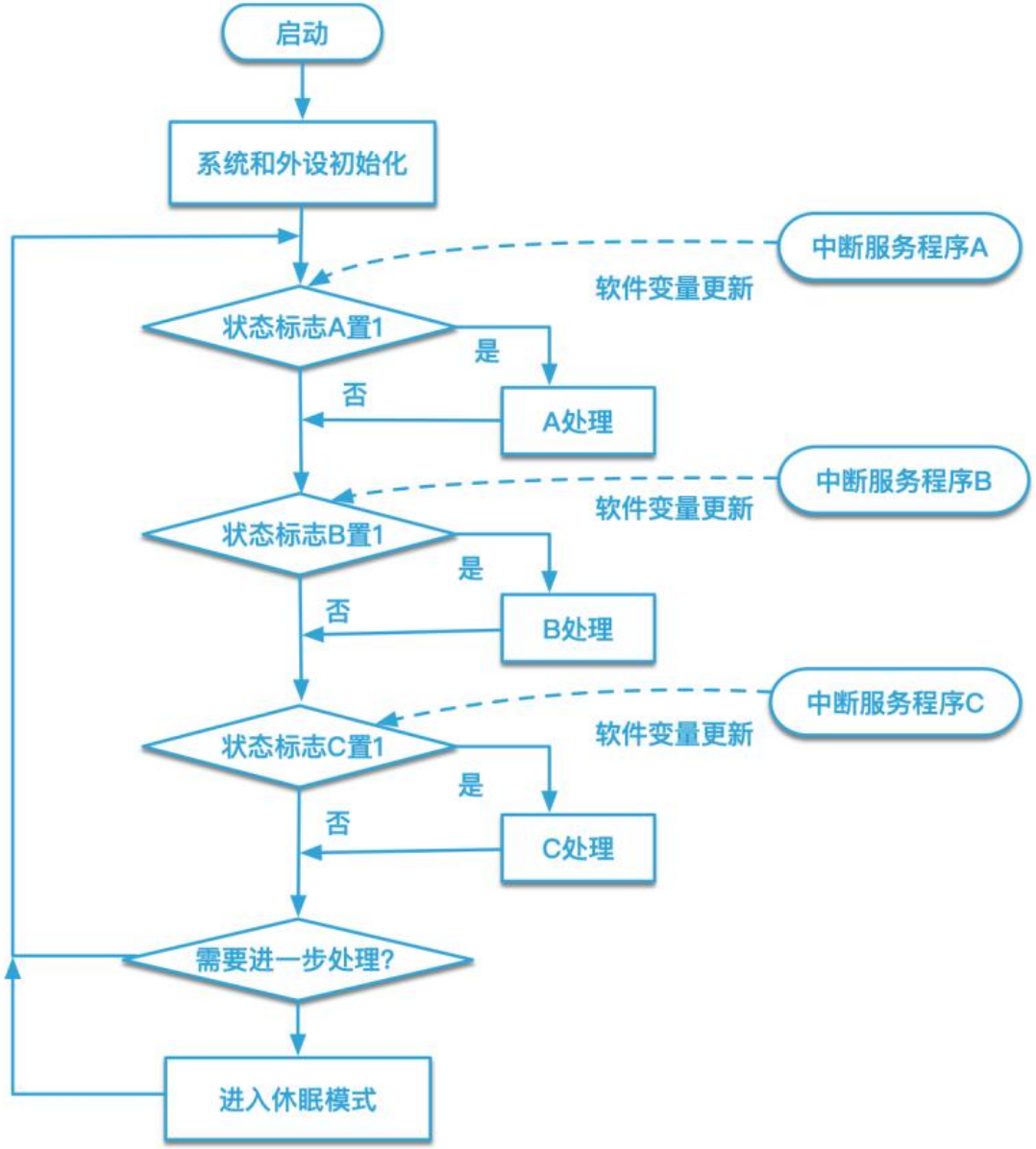

启动
系统和外设初始化
进入休眠模式
中断
中断服务程序A
A处理
中断服务程序B
B处理
中断服务程序C
C处理

软件运行流程-轮询及中断驱动

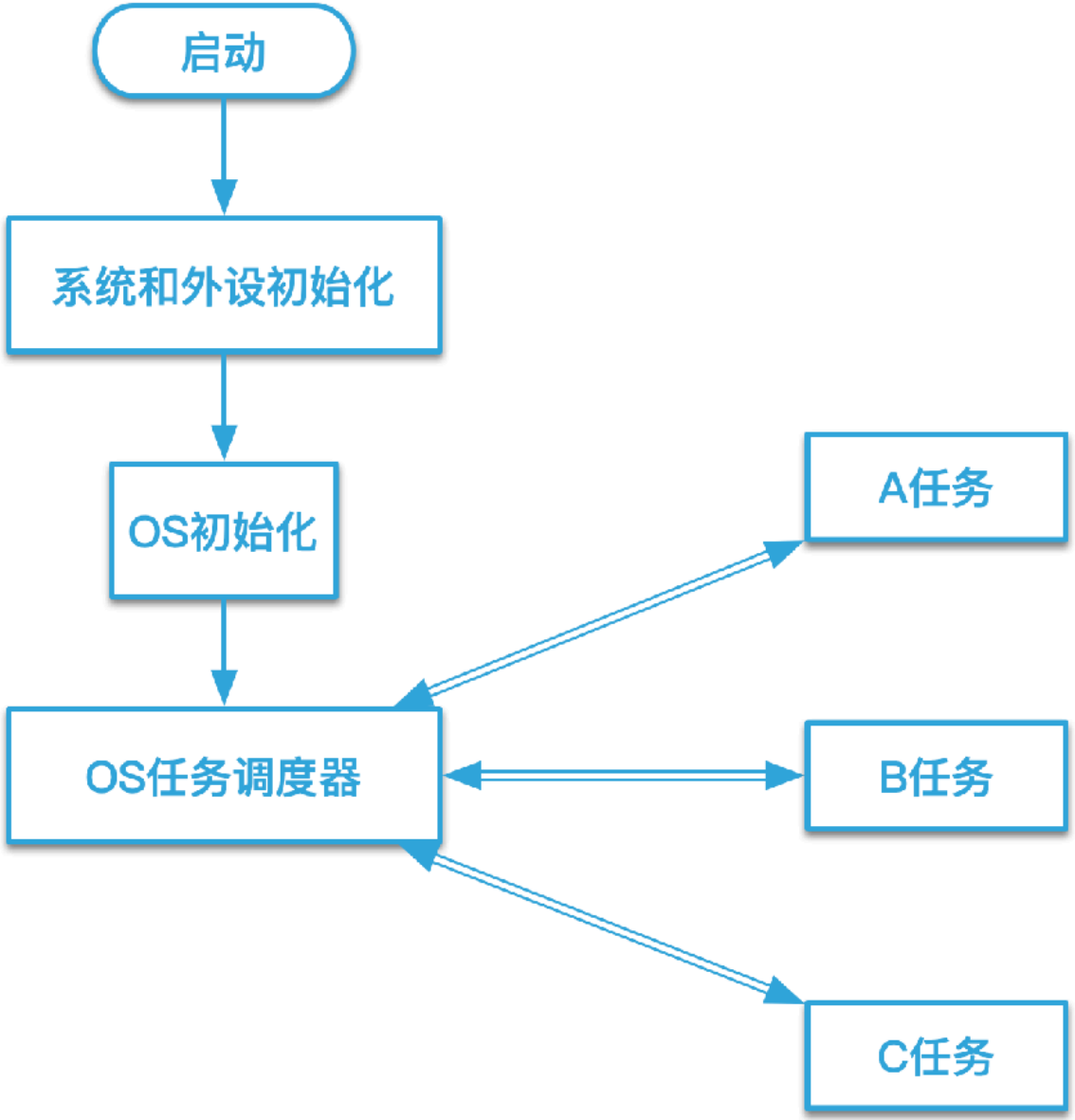




软件运行流程-轮询及中断驱动



软件运行流程-多任务系统



SysTick-系统定时器

SysTick（系统定时器）简介

- SysTick（系统定时器）是CM3 内核的一个外设，内嵌在NVIC 中
- 系统定时器是一个24bit 的向下递减的计数器，计数器每计数一次的时间为1/SYSCLK(一般设置系统时钟SYSCLK 为72M)
- 当重装载数值寄存器的值递减到0 的时候，系统定时器产生一次中断，以此循环往复
- 有关寄存器的定义和部分库函数在core_CM3.h 中

SysTick（系统定时器）简介

- 系统定时器有4 个寄存器

寄存器名称	寄存器描述
CTRL	SysTick 控制及状态寄存器
LOAD	SysTick 重装载数值寄存器
VAL	SysTick 当前数值寄存器
CALIB	SysTick 校准数值寄存器

SysTick 控制及状态寄存器

位段	名称	类型	复位值	描述
16	COUNTFLAG	R/W	0	如果在上次读取本寄存器后， SysTick 已经计到了 0，则该位为 1。
2	CLKSOURCE	R/W	0	时钟源选择位，0=AHB/8，1=处理器时钟 AHB
1	TICKINT	R/W	0	1=SysTick 倒数计数到 0 时产生 SysTick 异常请求， 0=数到 0 时无动作。也可以通过读取 COUNTFLAG 标志位来确定计数器是否递减到 0
0	ENABLE	R/W	0	SysTick 定时器的使能位

SysTick 重装载数值寄存器

位段	名称	类型	复位值	描述
23:0	RELOAD	R/W	0	当倒数计数至零时，将被重装载的值

SysTick 当前数值寄存器

位段	名称	类型	复位值	描述
23:0	CURRENT	R/W	0	读取时返回当前倒计数的值，写它则使之清零，同时还会清除在 SysTick 控制及状态寄存器中的 COUNTFLAG 标志

SysTick（系统定时器）简介

寄存器名称	寄存器描述
CTRL	SysTick 控制及状态寄存器
LOAD	SysTick 重装载数值寄存器
VAL	SysTick 当前数值寄存器
CALIB	SysTick 校准数值寄存器

- 系统定时器有4 个寄存器

表 19-5 SysTick 校准数值寄存器

位段	名称	类型	复位值	描述
31	NOREF	R	0	NOREF flag. Reads as zero. Indicates that a separate reference clock is provided. The frequency of this clock is HCLK/8
30	SKEW	R	1	Reads as one. Calibration value for the 1 ms inexact timing is not known because TENMS is not known. This can affect the suitability of SysTick as a software real time clock
23:0	TENMS	R	0	Indicates the calibration value when the SysTick counter runs on HCLK max/8 as external clock. The value is product dependent, please refer to the Product Reference Manual, SysTick Calibration Value section. When HCLK is programmed at the maximum frequency, the SysTick period is 1ms. If calibration information is not known, calculate the calibration value required from the frequency of the processor clock or external clock.

SysTick（系统定时器）使用流程

使用 SysTick 定时器的基本流程：

1、配置系统时钟源

时钟源可以是 处理器时钟（HCLK） 或 HCLK/8。默认情况下使用 HCLK/8。如需修改时钟源，通过修改 SysTick_CTRL 寄存器的 CLKSOURCE 位进行设置。

2. 设置重载值

SysTick 是一个 24 位递减计数器，可通过重载值设置定时周期。重载值存储在 SysTick_LOAD 寄存器中。

3. 清除当前计数值

为了确保计数从头开始，需要将 SysTick_VAL 寄存器清零。写入任何值到 SysTick_VAL 寄存器都会自动将其清零。

4. 启用 SysTick 和中断

通过设置 SysTick_CTRL 寄存器来启用 SysTick 定时器和中断。ENABLE 位：启用 SysTick 定时器。TICKINT 位：启用 SysTick 定时器的中断。

5. 编写 SysTick 中断服务函数

当计数器递减到 0 时，SysTick 定时器会触发中断。需要在中断向量表中定义 SysTick_Handler() 函数来处理中断。

SysTick（系统定时器）使用流程

使用固件库编程的时，只需调用库函数SysTick_Config()即可，形参ticks 用来设置重装载寄存器的值

SysTick 配置库函数

```
1  __STATIC_INLINE uint32_t SysTick_Config(uint32_t ticks)
2  {
3      // 不可能的重装载值，超出范围
4      if ((ticks - 1UL) > SysTick_LOAD_RELOAD_Msk) {
5          return (1UL);
6      }
7
8      // 设置重装载寄存器
9      SysTick->LOAD  = (uint32_t)(ticks - 1UL);
10
11     // 设置中断优先级
12     NVIC_SetPriority (SysTick_IRQn, (1UL << __NVIC_PRIO_BITS) - 1UL);
13
14     // 设置当前数值寄存器
15     SysTick->VAL   = 0UL;
16
17     // 设置系统定时器的时钟源为 AHBCLK=72M
18     // 使能系统定时器中断
19     // 使能定时器
20     SysTick->CTRL  = SysTick_CTRL_CLKSOURCE_Msk |
21                      SysTick_CTRL_TICKINT_Msk   |
22                      SysTick_CTRL_ENABLE_Msk;
23     return (0UL);
24 }
```

SysTick（系统定时器）使用流程

使用固件库编程的时，只需调用库函数SysTick_Config()即可，形参ticks 用来设置重装载寄存器的值

SysTick 配置库函数

```
1  __STATIC_INLINE uint32_t SysTick_Config(uint32_t ticks)
2  {
3      // 不可能的重装载值，超出范围
4      if ((ticks - 1UL) > SysTick_LOAD_RELOAD_Msk) {
5          return (1UL);
6      }
7
8      // 设置重装载寄存器
9      SysTick->LOAD  = (uint32_t)(ticks - 1UL);
10
11     // 设置中断优先级
12     NVIC_SetPriority (SysTick_IRQn, (1UL << __NVIC_PRIO_BITS) - 1UL);
13
14     // 设置当前数值寄存器
15     SysTick->VAL   = 0UL;
16
17     // 设置系统定时器的时钟源为 AHBCLK=72M
18     // 使能系统定时器中断
19     // 使能定时器
20     SysTick->CTRL  = SysTick_CTRL_CLKSOURCE_Msk |
21                      SysTick_CTRL_TICKINT_Msk   |
22                      SysTick_CTRL_ENABLE_Msk;
23     return (0UL);
24 }
```

SysTick（系统定时器）使用流程

- SysTick 初始化函数由用户编写，里面调用SysTick_Config()固件库函数
- 通过设置该固件库函数的形参，可决定系统定时器经过多少时间产生一次中断

SysTick 初始化函数

```
1  /**
2    * @brief  启动系统滴答定时器 SysTick
3    * @param  无
4    * @retval 无
5    */
6  void SysTick_Init(void)
7  {
8      /* SystemFrequency / 1000    1ms 中断一次
9       * SystemFrequency / 100000  10us 中断一次
10      * SystemFrequency / 1000000 1us 中断一次
11      */
12     if (SysTick_Config(SystemCoreClock / 100000)) {
13         /* Capture error */
14         while (1);
15     }
16 }
```

SysTick（系统定时器）使用流程

SysTick 中断时间的计算

- SysTick 定时器的计数器是向下递减计数的，计数一次的时间$T=1/CLK_{AHB}$
- 当重装载寄存器中的值$VALUE_{LOAD}$ 减到0 的时候， 产生中断
- 中断一次的时间$T=VALUE_{LOAD} * T= VALUE_{LOAD}/CLK_{AHB}$ ， 其中CLK_{AHB} =72MHz
- 当配置为SystemCoreClock / 100000时，有$VALUE_{LOAD}=72M/100000=720$，代入上面公式得到定时器中断一次的时间：T=720*（1/72M）=10us
- 需要定更长的时间时，可在SysTick的中断服务函数中设置变量t，用来记录进入中断的次数；变量t 乘以单次中断的时间则可计算出需要定时的时间

SysTick（系统定时器）使用流程

SysTick 做延迟定时的方法2

SysTick 控制及状态寄存器

位段	名称	类型	复位值	描述
16	COUNTFLAG	R/W	0	如果在上次读取本寄存器后， SysTick 已经计到了 0，则该位为 1。
2	CLKSOURCE	R/W	0	时钟源选择位，0=AHB/8，1=处理器时钟 AHB
1	TICKINT	R/W	0	1=SysTick 倒数计数到 0 时产生 SysTick 异常请求， 0=数到 0 时无动作。也可以通过读取 COUNTFLAG 标志位来确定计数器是否递减到 0
0	ENABLE	R/W	0	SysTick 定时器的使能位

- SysTick 的counter 从递减到0 时，CTRL 寄存器的位16:countflag 会置1，且读取该位的值可清0
- 可利用软件查询countflag的方法来实现延时

systick 微秒级延时

```
1  void SysTick_Delay_Us( __IO uint32_t us)
2  {
3      uint32_t i;
4      SysTick_Config(SystemCoreClock/1000000);
5
6      for (i=0; i<us; i++) {
7          // 当计数器的值减小到 0 的时候，CRTL 寄存器的位 16 会置 1
8          while ( !((SysTick->CTRL)&(1<<16)) );
9      }
10     // 关闭 SysTick 定时器
11     SysTick->CTRL &=~SysTick_CTRL_ENABLE_Msk;
12 }
```

练习1：使用SysTick实现LED闪烁的精确定时

- 结合“实验参考20241024.pdf”内容进行实验，实现：

1、参照实验参考20241024.pdf P160实验和代码“19-SysTick—系统定时器”完成使用SysTick 定时器对LED的控制实验；

2、将之前的红绿灯代码修改为使用SysTick进行精准定时。

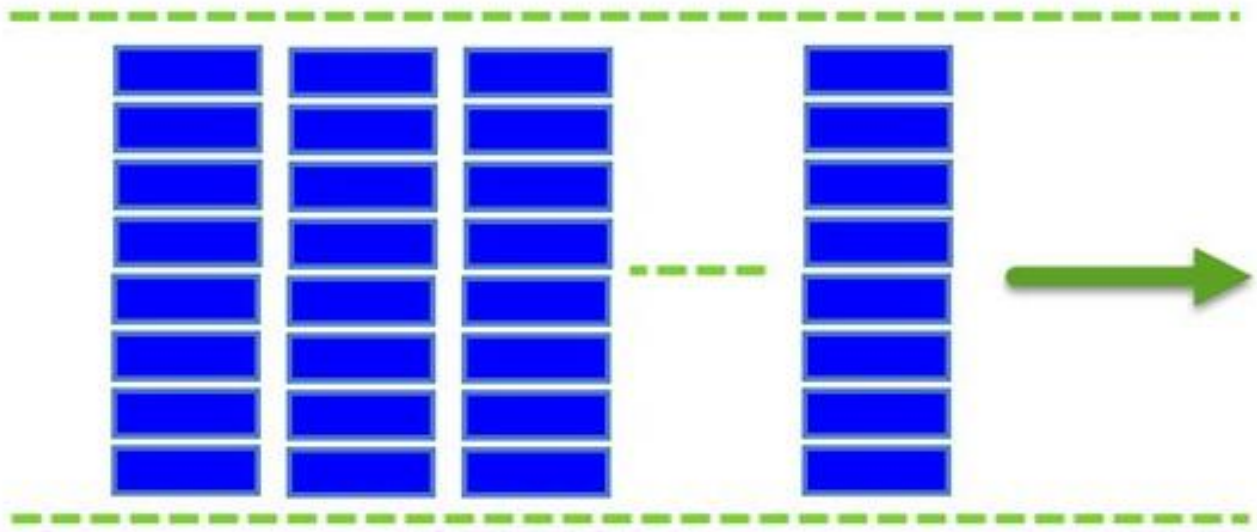
本练习无需提交实验报告

通信基本概念

串行通信与并行通信

并行通信

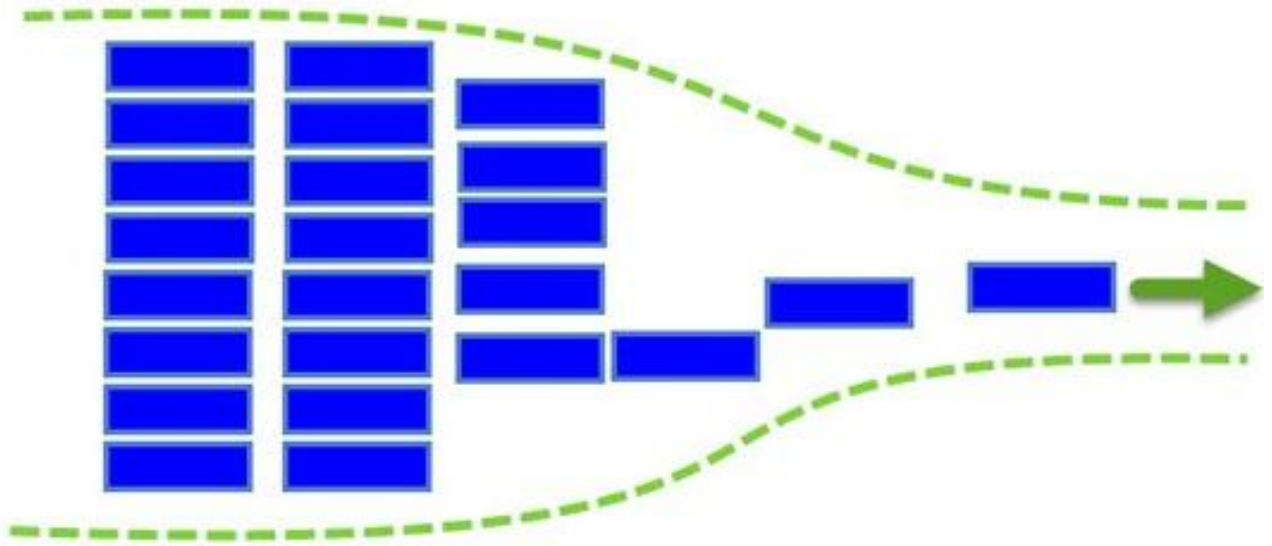
-传输原理：数据各个位同时传输。
-优点：速度快
-缺点：占用引脚资源多



并行通讯

串行通信

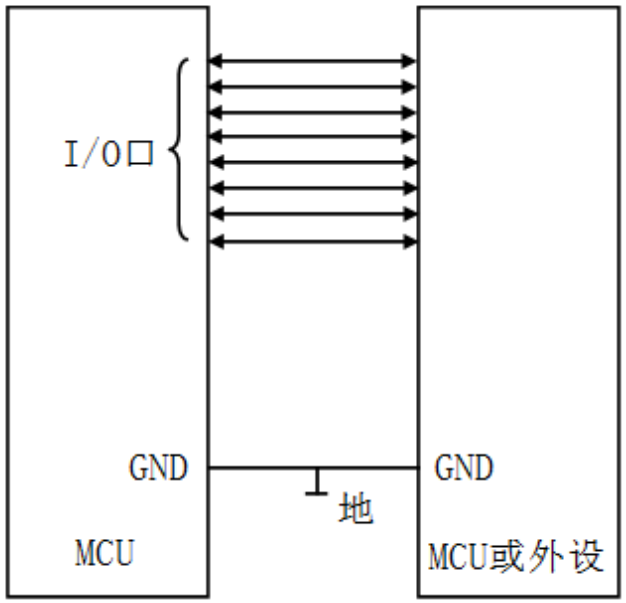
-传输原理：数据按位顺序传输。
-优点：占用引脚资源少
-缺点：速度相对较慢



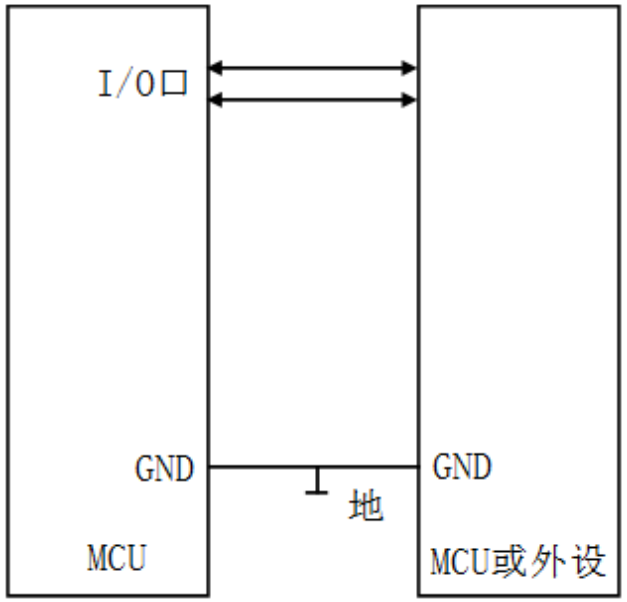
串行通讯

串行通信与并行通信

- 并行通信:是指使用多条数据线传输数据。并行通信时，各个位同时在不同的数据线上传送，数据可以字或字节为单位并行进行传输
- 串行通信:是指使用一条数据线，将数据一位一位地在这条数据线上依次传输



(a) 并行通信



(b) 串行通讯

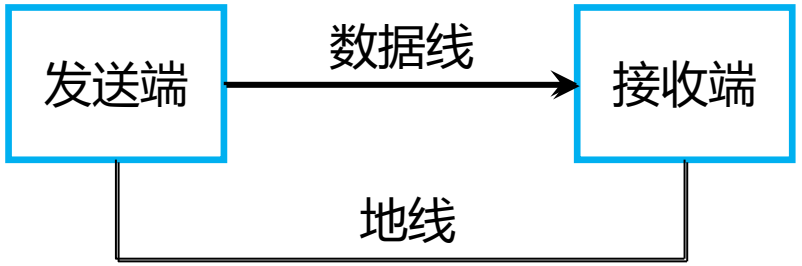
串行通信的制式

全双工
可同时收发数据

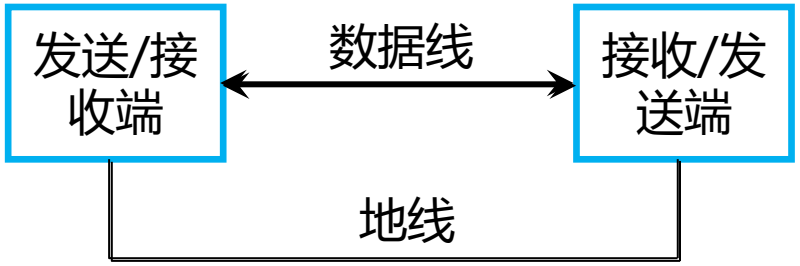
半双工
不可同时收发数据，可分时收发数据

单工
任何时刻都只能往某一个固定方向传输数据

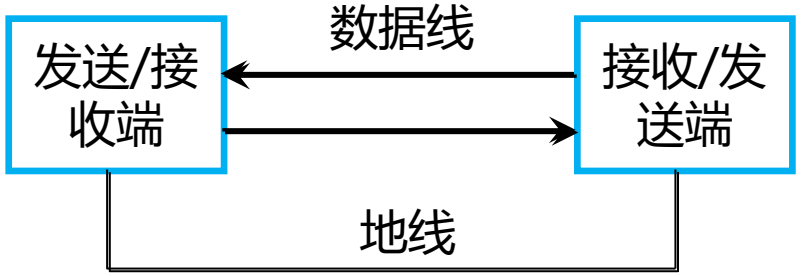
串行通信的制式



单工制式（Simplex）是指甲乙双方通信时只能单向传送数据。系统组成以后，发送方和接收方固定。

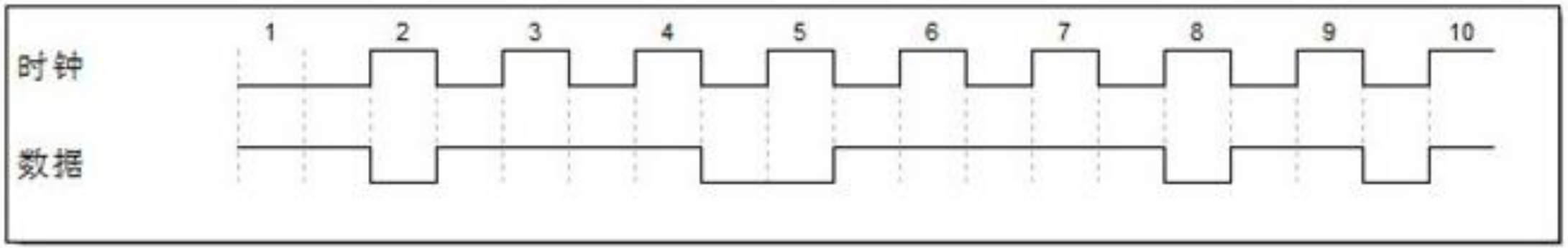


半双工制式（Half Duplex）是指通信双方都具有发送器和接收器，既可发送也可接收，但不能同时接收和发送，发送时不能接收，接收时不能发送。



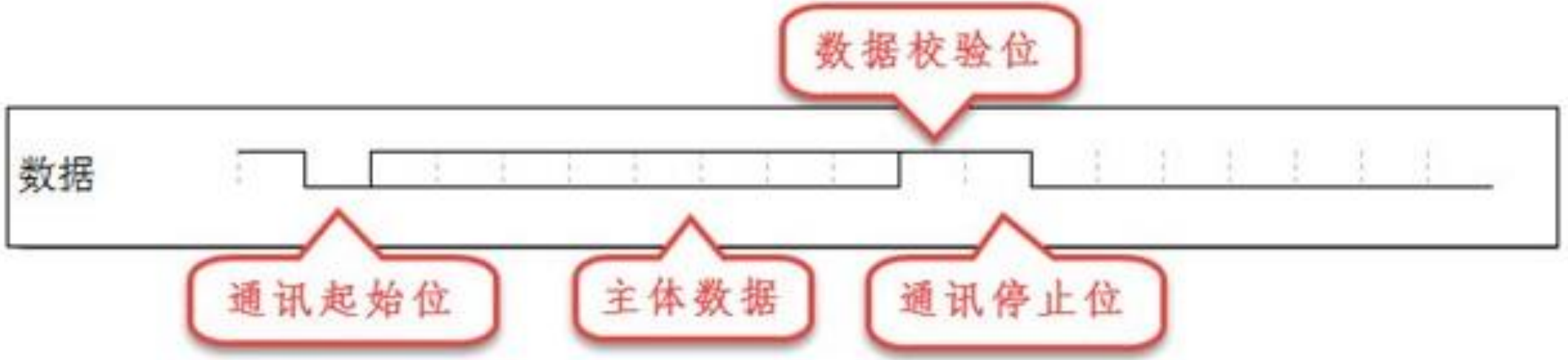
全双工制式（Full Duplex）是指通信双方均设有发送器和接收器，并且信道划分为发送信道和接收信道，因此全双工制式可实现甲方（乙方）同时发送和接收数据，发送时能接收，接收时也能发送。

同步通信



同步通信：带时钟同步信号传输 如：SPI，IIC通信接口

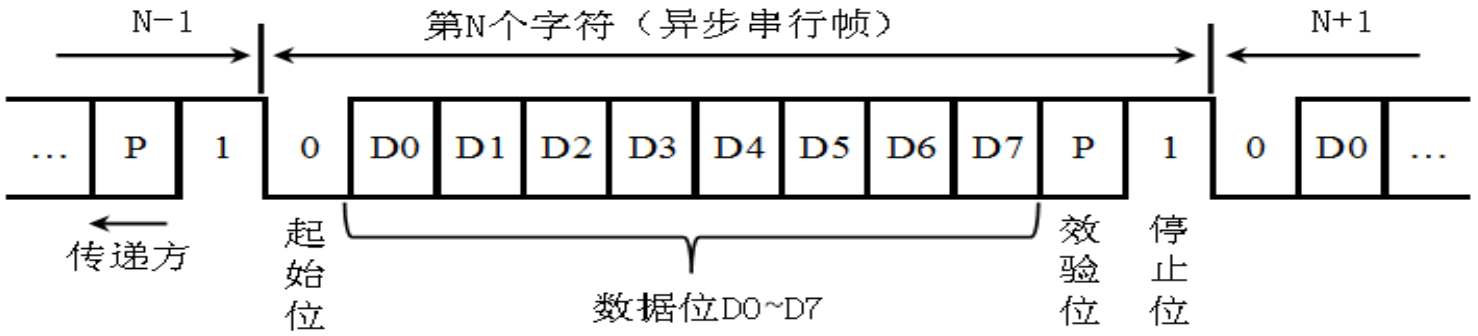
异步通信



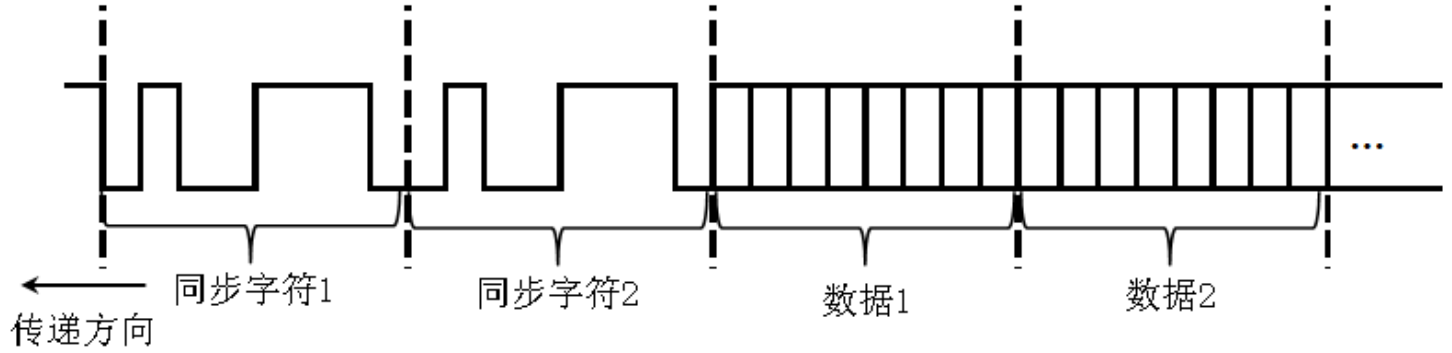
异步通信：不带时钟同步信号 如UART(通用异步收发器)，单总线

异步通信与同步通信

- 异步通信:异步通信数据传送按帧传输，一帧数据包含起始位、数据位、校验位和停止位。



- 同步通信:同步通信是由1~2个同步字符和多字节数据位组成，同步字符作为起始位以触发同步时钟开始发送或接受数据;多字节数据之间不允许有空隙，每位占用的时间相等；空闲位需发送同步字符.



通信速率

衡量通讯性能的一个非常重要的参数就是通讯速率，通常以比特率（ Bitrate）来表示，即每秒钟传输的二进制位数，单位为比特每秒bit/s

容易与比特率混淆的概念是“波特率” （ Baudrate），它表示每秒钟传输了多少个码元

通信速率

波特率（baud rate）是串行通信中一个重要概念，是指传输数据的速率。波特率（bit per second）的定义是每秒传输数据的位数，即：

1波特=1位/秒（1bit/s）

波特率的倒数即为每位传输所需的时间

由以上串行通信原理可知，互相通信的甲乙双方必须具有相同的波特率，否则无法成功地完成串行数据通信。

串行通信的校验

奇偶校验:奇偶校验在发送数据时，数据位尾随的1的位数为奇偶校验位（1或0），当设置为奇校验时，数据中1的个数与校验位1的个数之和应为奇数；当设置为偶校验时，数据中1的个数与校验位中的1的个数之和应为偶数。

累加和校验:累加和校验是指发送方将所发送的数据块求和，并将“校验和”附加到数据块末尾。接收方接收数据时也是对数据块求和，将所得结果与发送方的“校验和”进行比较，相符则无差错，否则即出现了差错。

循环冗余码校验:循环冗余码校验（Cyclic Redundancy Check，简称CRC）的基本原理是将一个数据块看成一个位数很长的二进制数，然后用一个特定的数去除它，将余数作校验码附在数据块后一起发送。

串行通信的校验

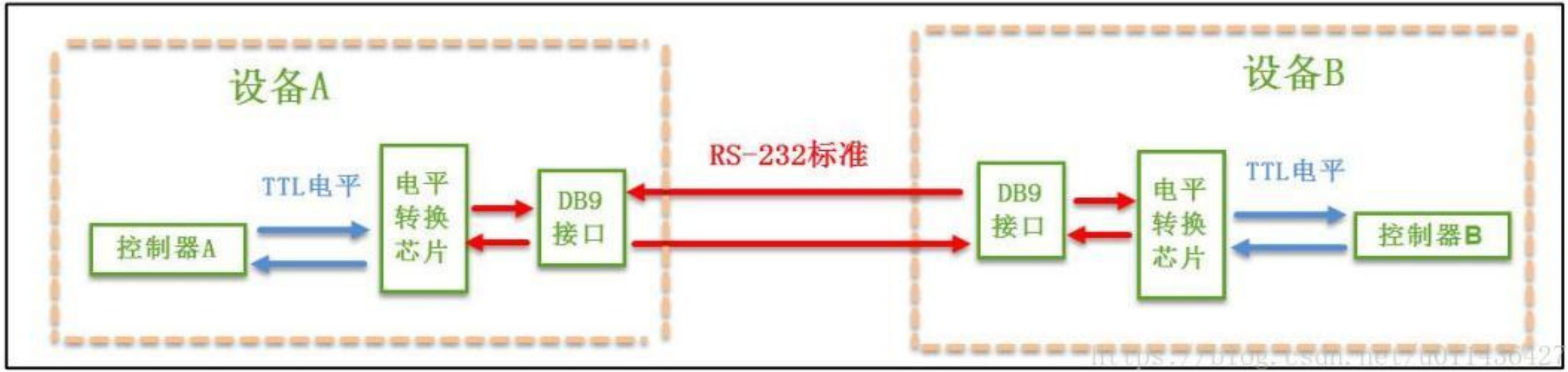
常见的串行通信接口：

通信标准	引脚说明	通信方式	通信方向
UART (通用异步收发器)	TXD:发送端 RXD:接受端 GND:公共地	异步通信	全双工
单总线 (1-wire)	DQ:发送/接受端	异步通信	半双工
SPI	SCK:同步时钟 MISO:主机输入，从机输出 MOSI:主机输出，从机输入	同步通信	全双工
I2C	SCL:同步时钟 SDA:数据输入/输出端	同步通信	半双工

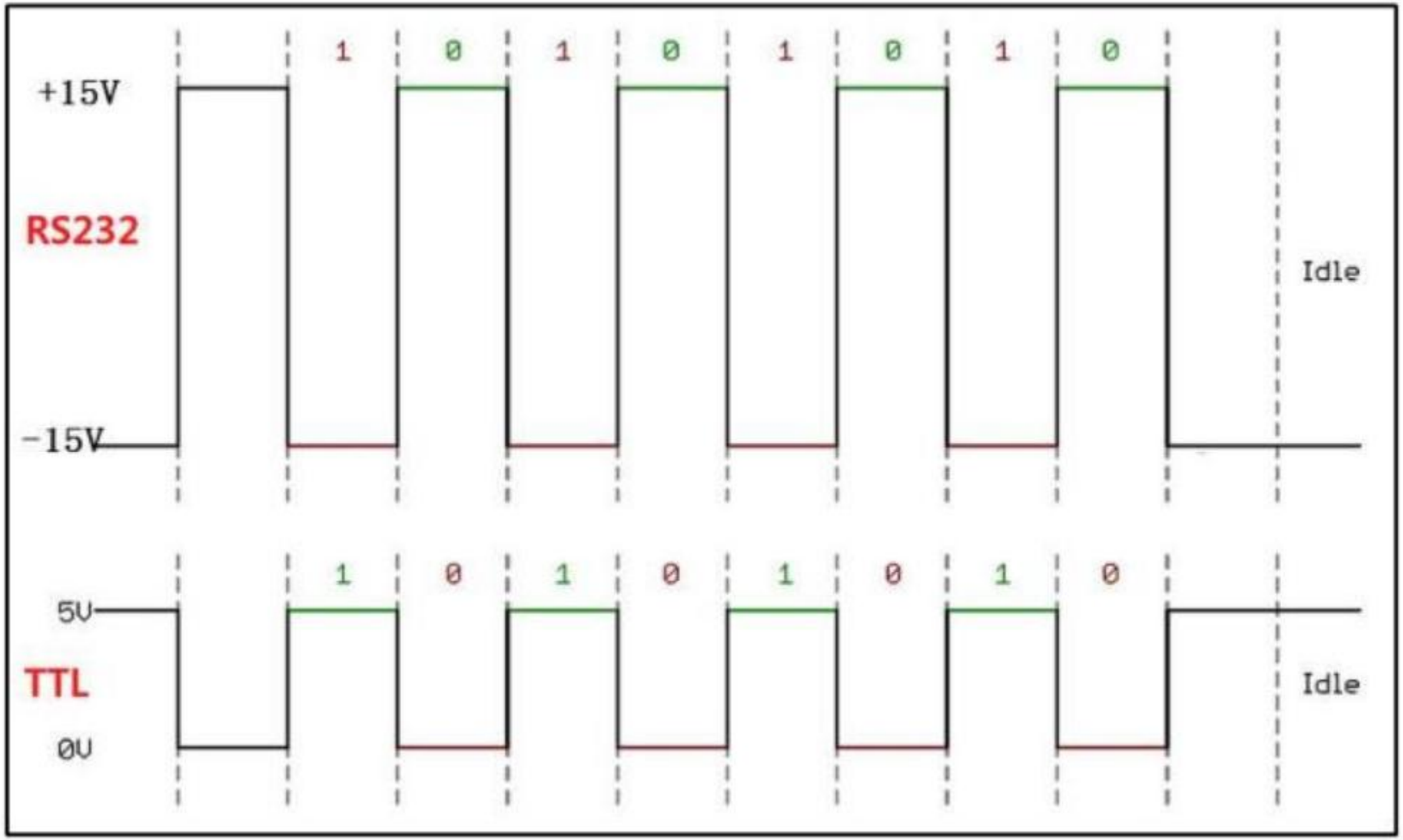
USART

物理层

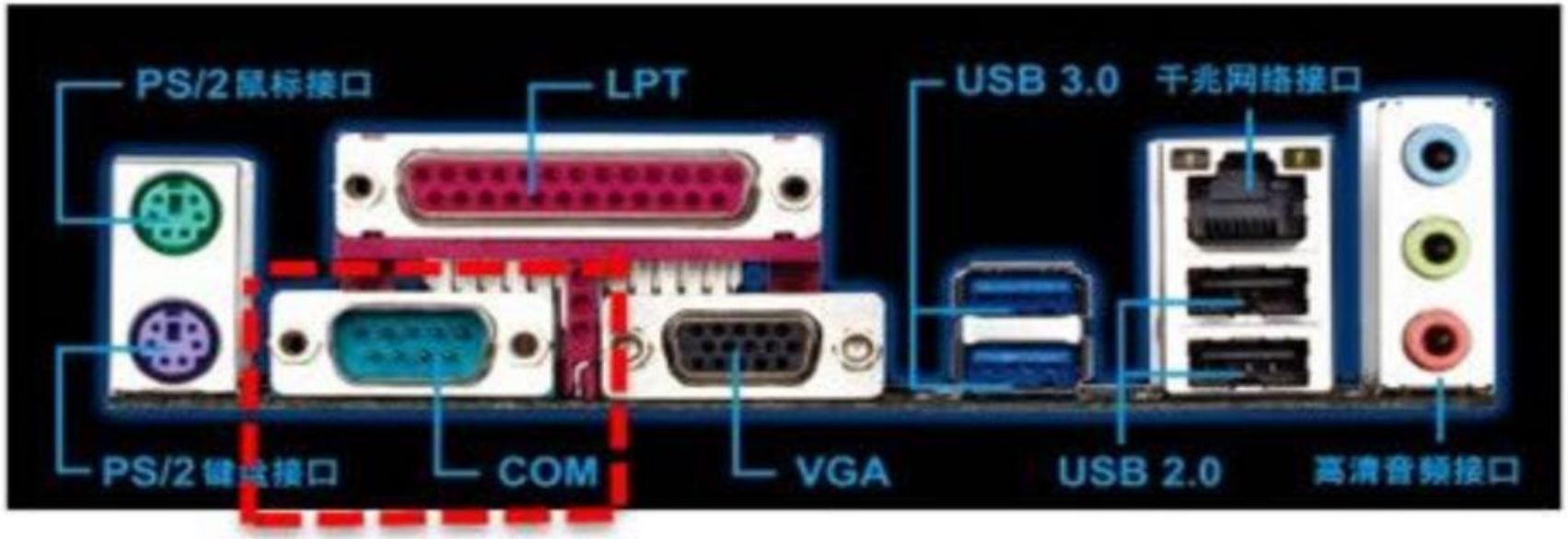
- 串口通讯的物理层有很多标准及变种，例如RS-232标准
- RS232标准主要规定了信号的用途、通讯接口以及信号的电平标准
- 使用RS-232标准的串口设备间常见的通讯结构



物理层

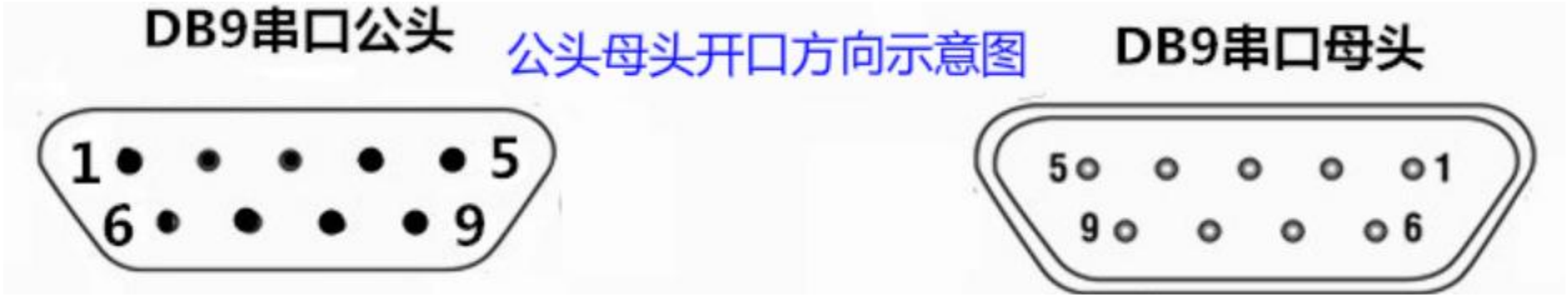


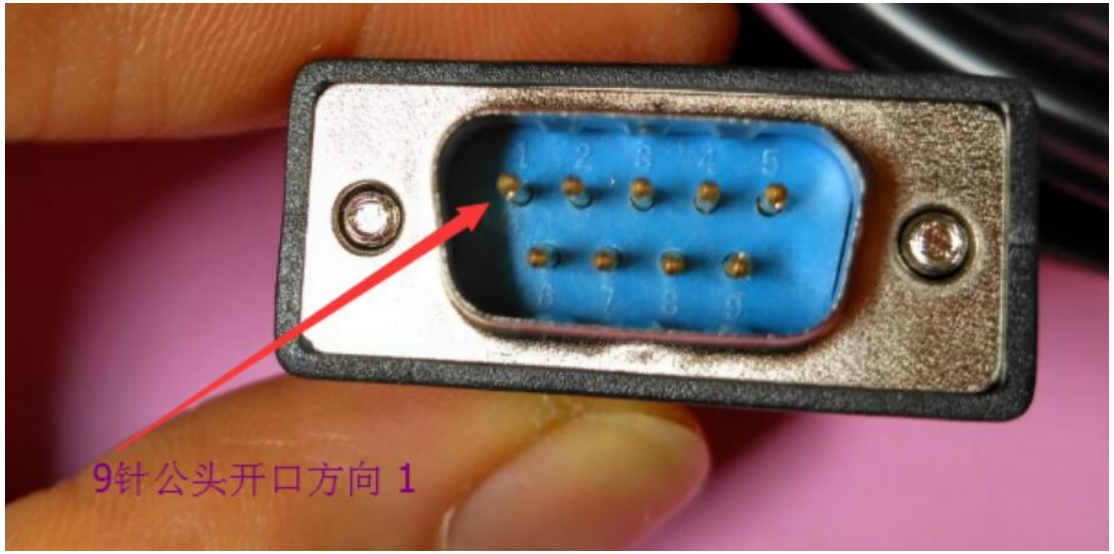
RS232信号线

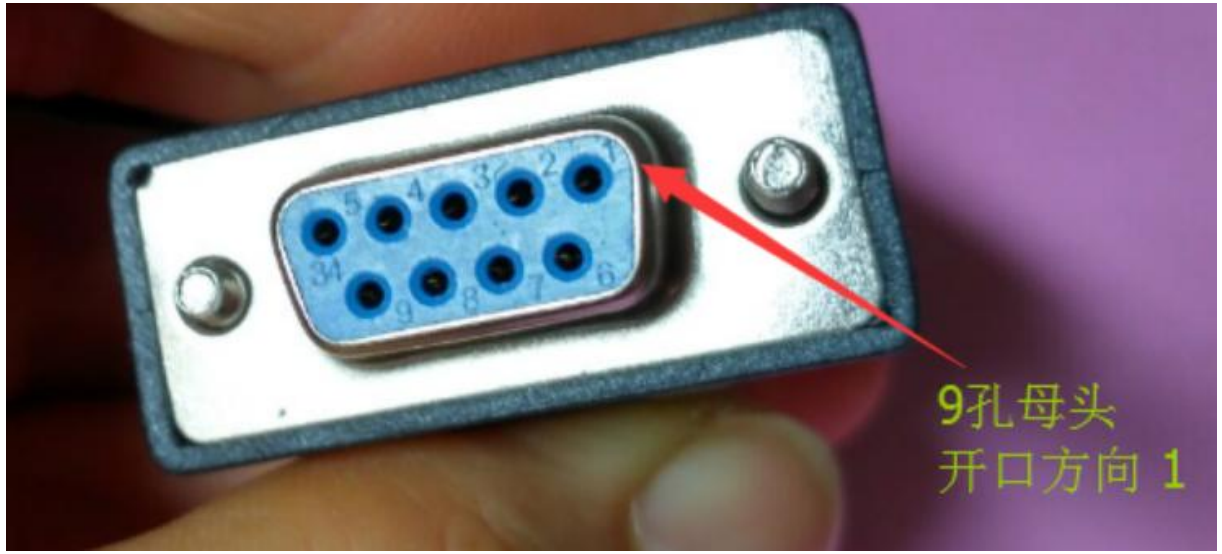


COM口即DB9接口

RS232信号线


DB9串口公头
公头母头开口方向示意图
DB9串口母头
1 5
6 9
5 1
9 6

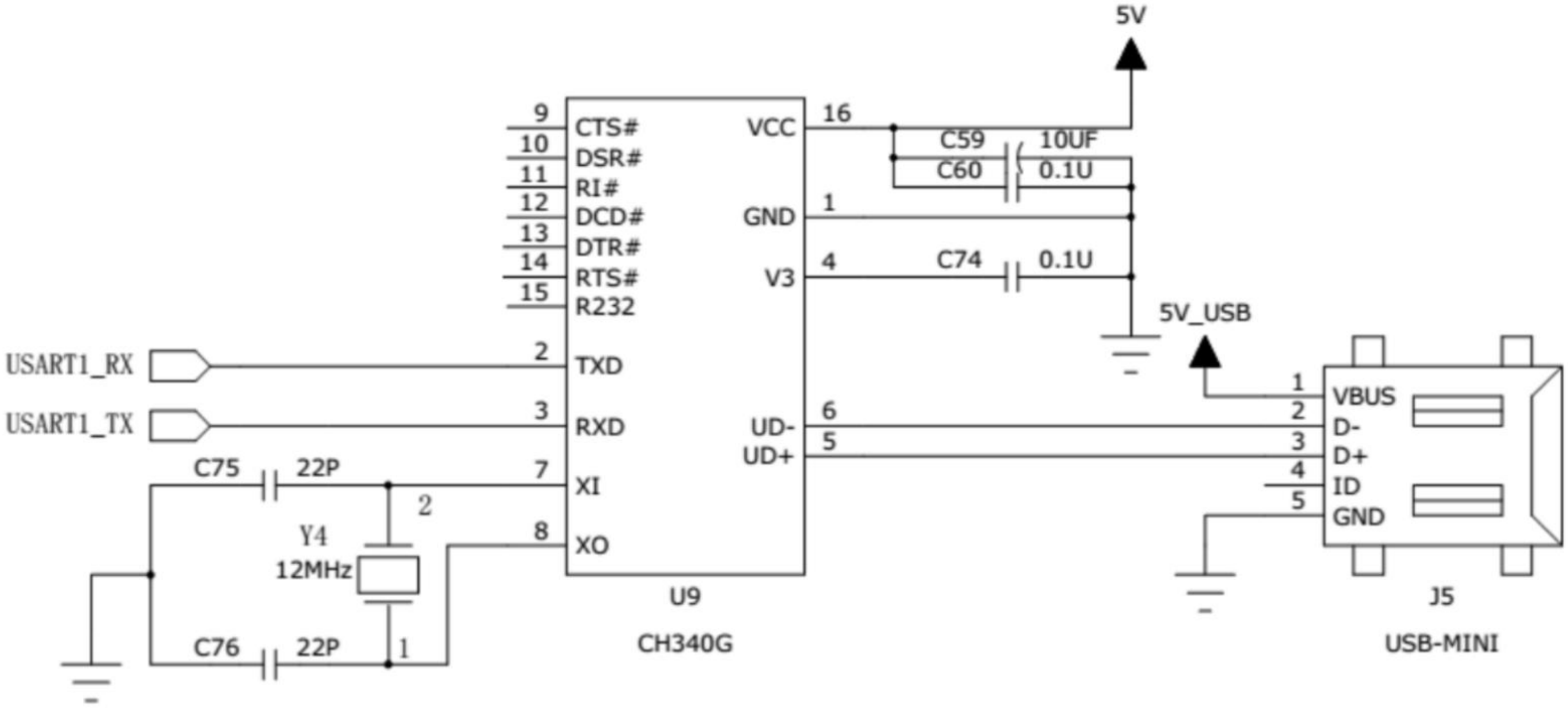

9针公头开口方向 1


9孔母头
开口方向 1

DB9信号定义

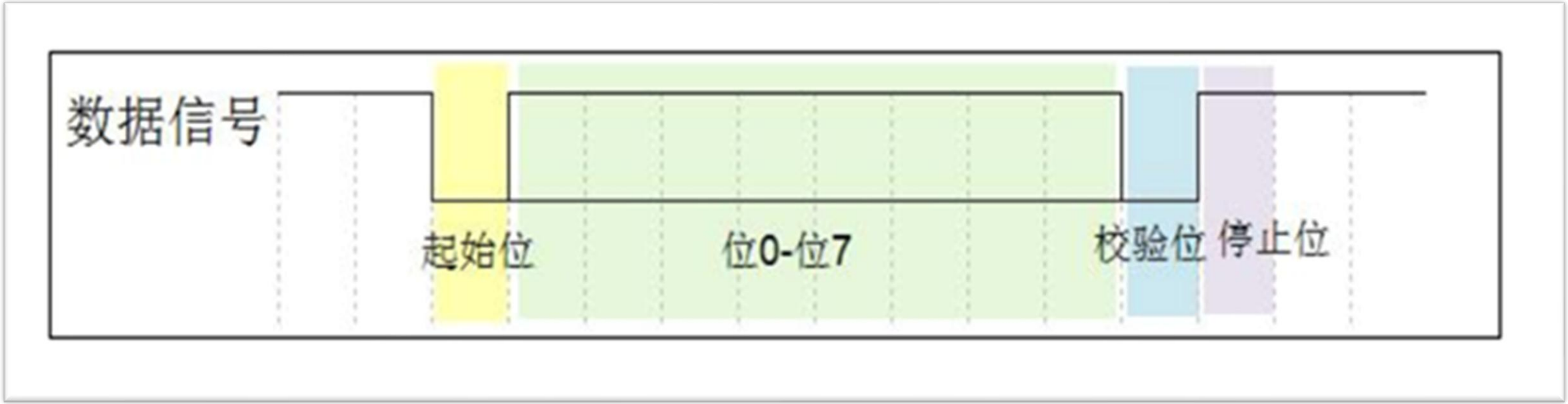
序号	名称	符号	数据方向	说明
1	载波检测	DCD	DTE->DCE	Data Carrier Detect，数据载波检测，用于DTE告知对方，本机是否收到对方的载波信号
2	接收数据	RXD	DTE<-DCE	Receive Data，数据接收信号，即输入。
3	发送数据	TXD	DTE->DCE	Transmit Data，数据发送信号，即输出。两个设备之间的TXD与RXD应交叉相连
4	数据终端(DTE) 就绪	DTR	DTE->DCE	Data Terminal Ready，数据终端就绪，用于DTE向对方告知本机是否已准备好
5	信号地	GND	-	地线，两个通讯设备之间的地电位可能不一样，这会影响收发双方的电平信号，所以两个串口设备之间必须要使用地线连接，即共地。
6	数据设备(DCE)就绪	DSR	DTE<-DCE	Data Set Ready，数据发送就绪，用于DCE告知对方本机是否处于待命状态
7	请求发送	RTS	DTE->DCE	Request To Send，请求发送， DTE 请求 DCE 本设备向DCE端发送数据
8	允许发送	CTS	DTE<-DCE	Clear To Send，允许发送，DCE回应对方的RTS发送请求，告知对方是否可以发送数据
9	响铃指示	RI	DTE<-DCE	Ring Indicator，响铃指示，表示DCE端与线路已接通

USB转RS232


5V
9
CTS#
10
DSR#
11
RI#
12
DCD#
13
DTR#
14
RTS#
15
R232
VCC
16
GND
1
V3
4
C59
10UF
C60
0.1U
C74
0.1U
5V_USB
USART1_RX
2
TXD
USART1_TX
3
RXD
UD-
UD+
6
5
C75
22P
7
XI
2
Y4
12MHz
8
XO
C76
22P
1
U9
CH340G
1
VBUS
2
D-
3
D+
4
ID
5
GND
J5
USB-MINI

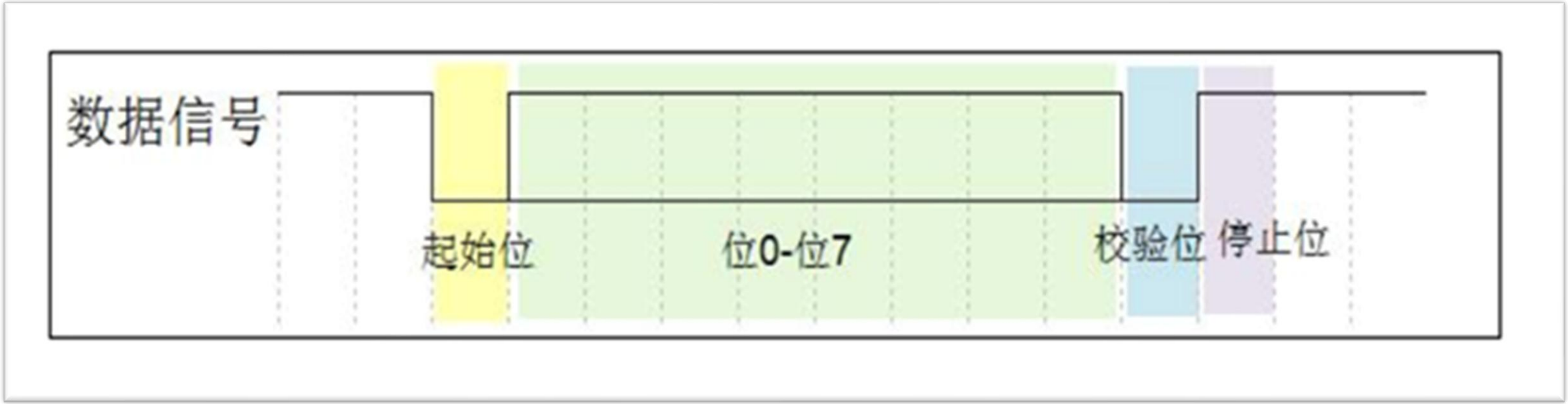
协议层

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



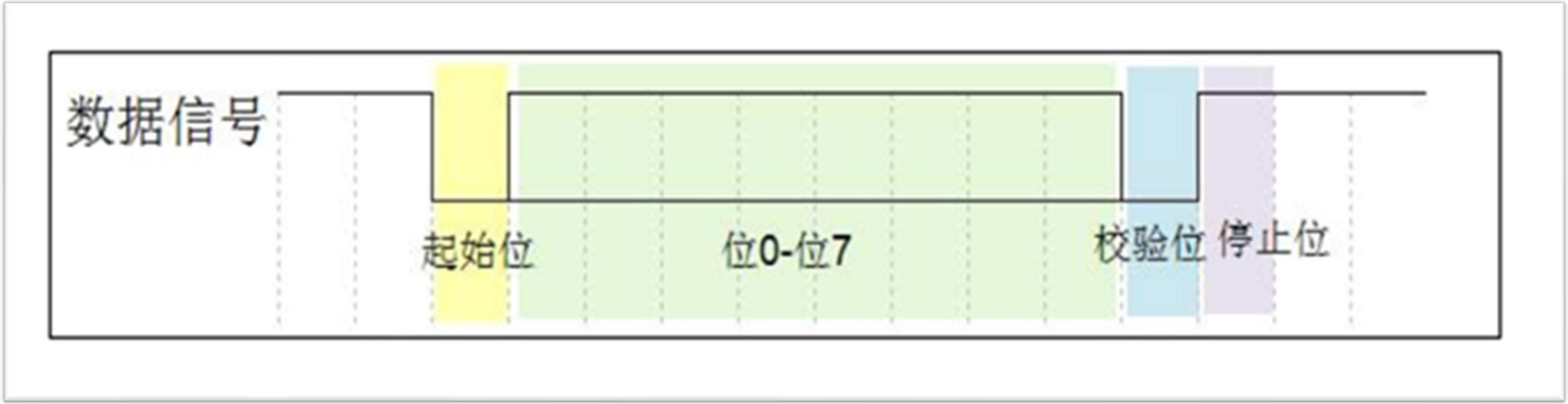
波特率

串口异步通讯，异步通讯中由于没有时钟信号，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。常见的波特率为4800、9600、115200等。



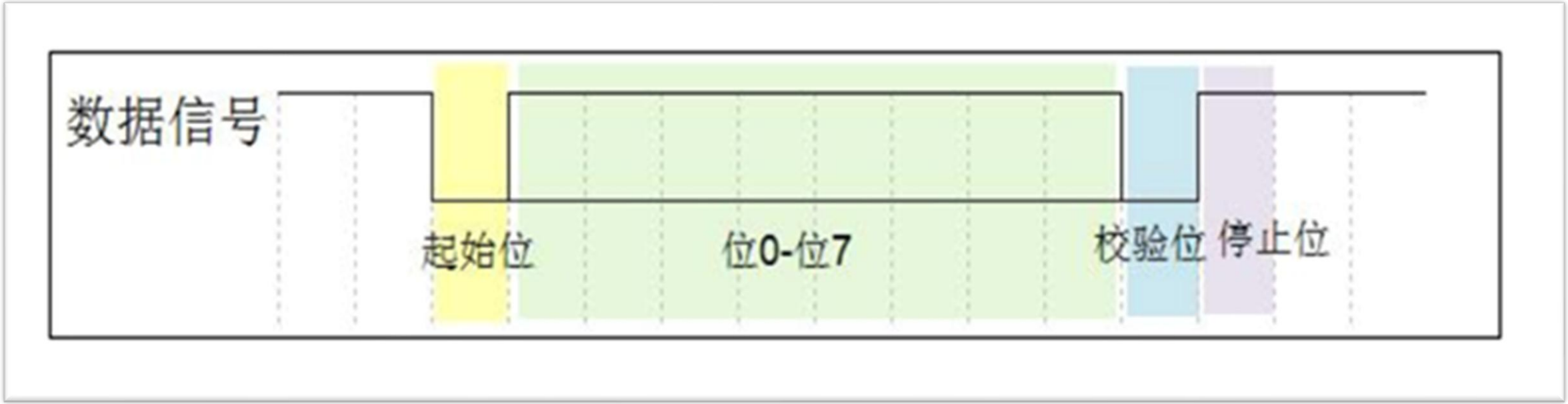
通信的起始和停止信号

串口通信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



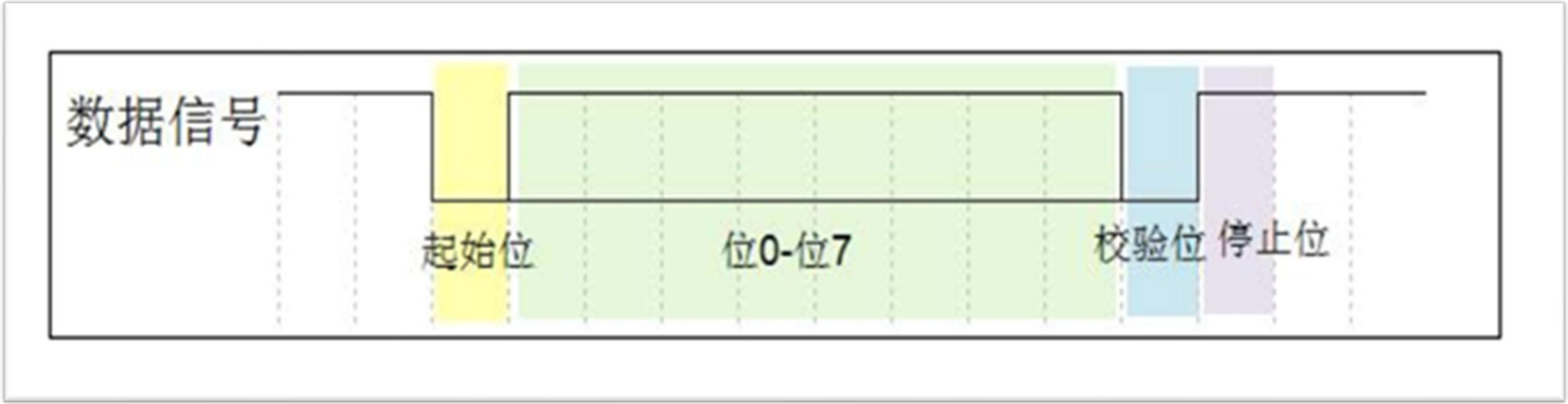
有效数据

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。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)



数据校验

- 奇(偶)校验要求有效数据和校验位中”１“的个数为奇(偶)数，最后传输的数据将是8位的有效数据加上1位的校验位总共9位
- 0校验是不管有效数据中的内容是什么，校验位总为“0"，1校验是校验位总为“1 “。
- 在无校验的情况下，数据包中不包含校验位

STM32的串口通信接口

UART:通用异步收发器

USART:通用同步异步收发器

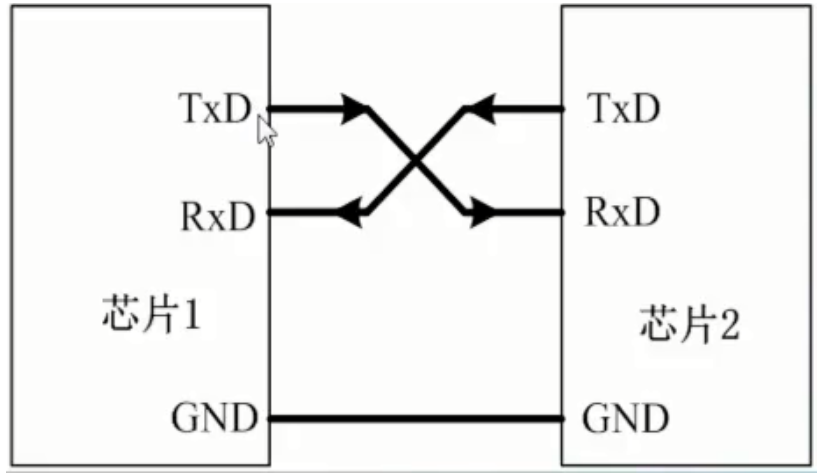
- 同步异步收发器可以灵活地与外部设备进行全双工数据交换
- USART支持使用DMA，可实现高速数据通信
- STM32的 USART输出的是TTL电平信号，若需要RS-232标准的信号可使用MAX3232芯片进行转换

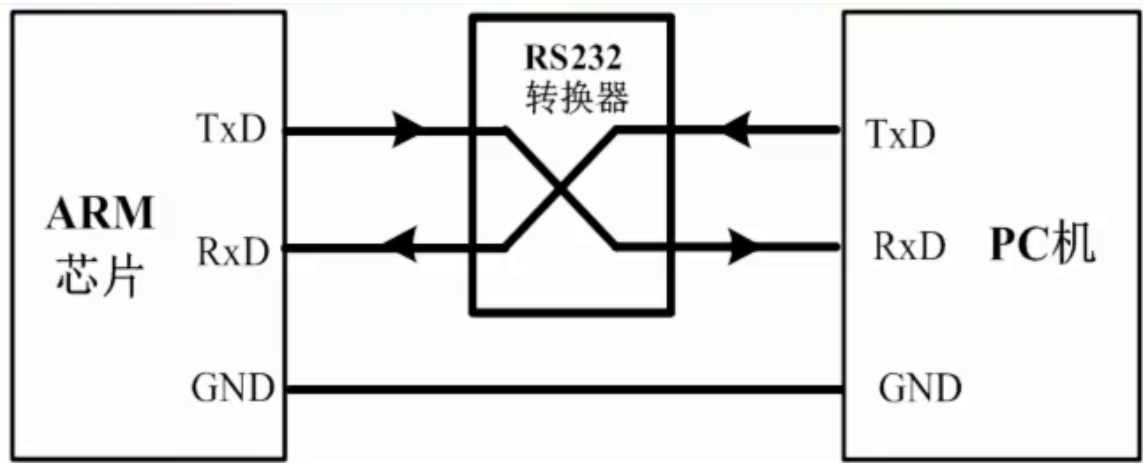
STM32的串口通信接口

UART异步通信方式引脚连接方法：

-RXD:数据输入引脚，数据接收

-TXD:数据发送引脚，数据发送





STM32的串口通信接口

UART异步通信方式引脚连接方法：

-RXD:数据输入引脚，数据接收

-TXD:数据发送引脚，数据发送

串口号	RXD	TXD
1	PA10	PA9
2	PA3	PA2
3	PB11	PB10
4	PC11	PC10
5	PD2	PC12

STM32的串口通信接口

UART异步通信方式特点：

- 全双工异步通信。
- 分数波特率发生器系统，提供精确的波特率。
- -发送和接受共用的可编程波特率，最高可达4.5Mbits/s
- 可编程的数据字长度（8位或者9位）；
- 可配置的停止位（支持1或者2位停止位）；
- 可配置的使用DMA多缓冲器通信。
- 单独的发送器和接收器使能位。
- 检测标志：① 接受缓冲器 ②发送缓冲器空 ③传输结束标志
- 多个带标志的中断源。触发中断。
- 其他：校验控制，四个错误检测标志。

STM32的串口通信接口

串口通信过程

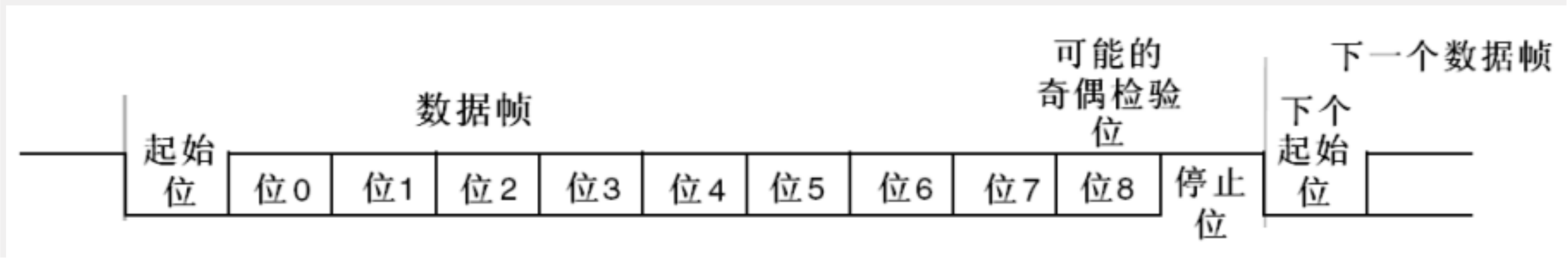
数据接收过程：

数据发送过程：

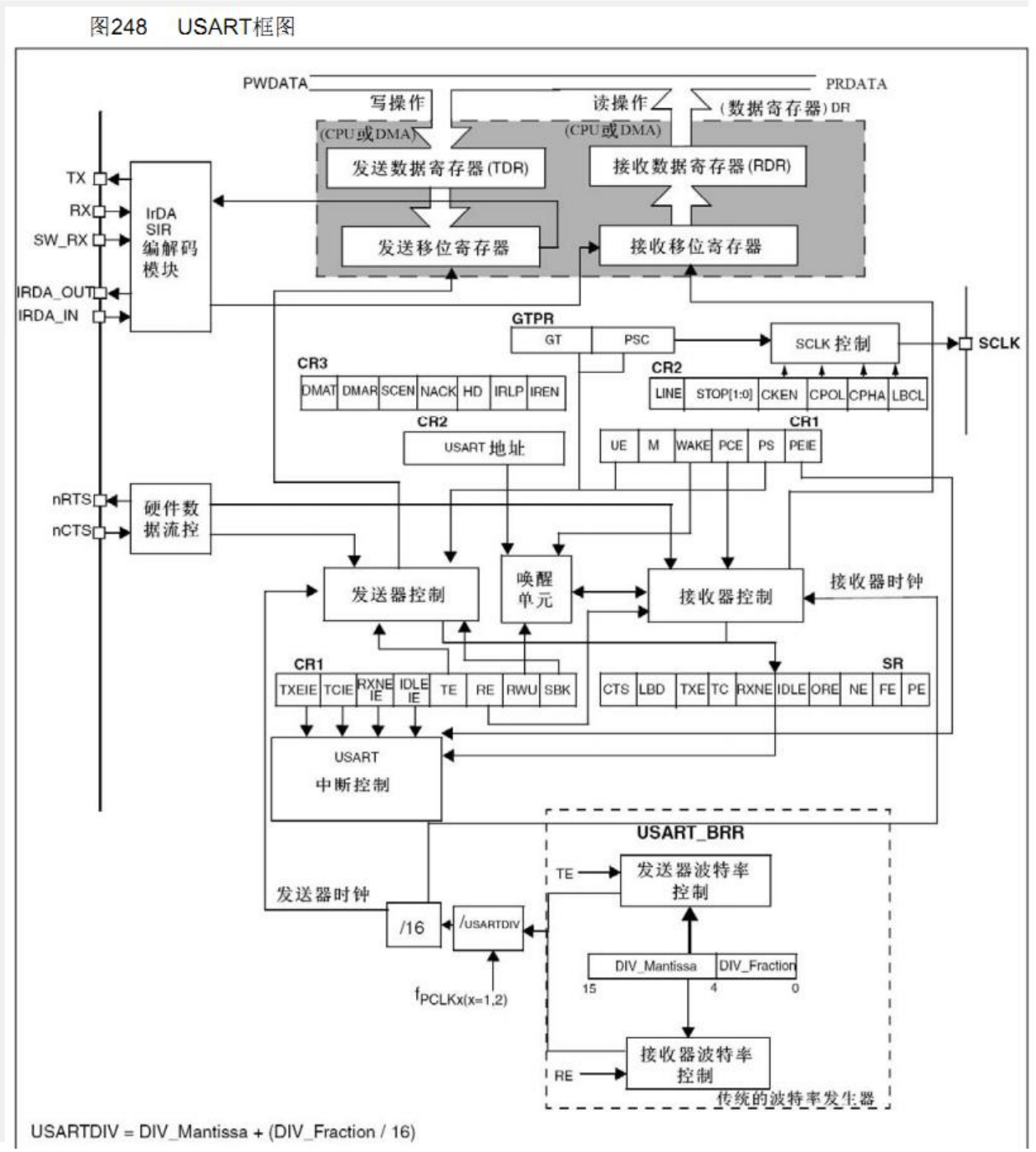
STM32的串口通信接口

STM32串口异步通信需要定义的参数

① 起始位
② 数据位（8位或者9位）
③ 奇偶校验位（第9位）
④ 停止位（1,1.5,2位）
⑤ 波特率设置



STM32的串口通信接口



STM32的串口通信接口

常用的串口相关寄存器

- 状态寄存器(USART_SR)
- 数据寄存器(USART_DR)
- 波特比率寄存器(USART_BRR)
- 控制寄存器 1(USART_CR1)
- 控制寄存器 2(USART_CR2)
- 控制寄存器 3(USART_CR3)
- 保护时间和预分频寄存器(USART_GTPR)

STM32的串口通信接口

串口操作相关函数

void USART_Init(); //串口初始化：波特率，数据字长，奇偶校验，硬件流控以及收发使能
void USART_Cmd();//使能串口
void USART_ITConfig();//使能相关中断

void USART_SendData();//发送数据到串口，DR
uint16_t USART_ReceiveData();//接受数据，从DR读取接受到的数据

FlagStatus USART_GetFlagStatus();//获取状态标志位
void USART_ClearFlag();//清除状态标志位
ITStatus USART_GetITStatus();//获取中断状态标志位
void USART_ClearITPendingBit();//清除中断状态标志位

STM32的串口通信接口

状态寄存器(USART_SR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

位31:10	保留位，硬件强制为0
位9	**CTS**: CTS 标志 (CTS flag) 如果设置了CTSE位，当nCTS输入变化状态时，该位被硬件置高。由软件将其清零。如果USART_CR3中的CTSIE为'1'，则产生中断。 0：nCTS状态线上没有变化； 1：nCTS状态线上发生变化。 注：UART4和UART5上不存在这一位。
位8	**LBD**: LIN断开检测标志 (LIN break detection flag) 当探测到LIN断开时，该位由硬件置'1'，由软件清'0'(向该位写0)。如果USART_CR3中的LBDIE = 1，则产生中断。 0：没有检测到LIN断开；

FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG);

STM32的串口通信接口

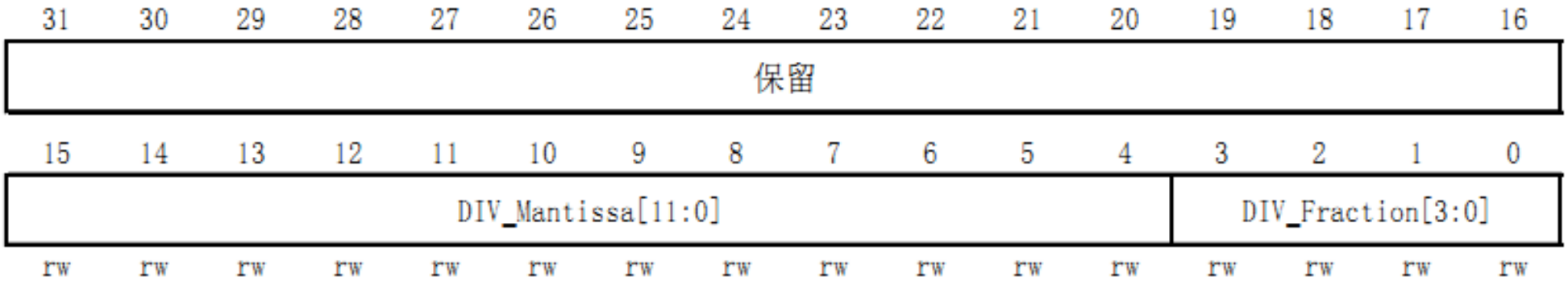
数据寄存器(USART_DR)

位31:9	保留位，硬件强制为0
位8:0	**DR[8:0]**：数据值 (Data value) 包含了发送或接收的数据。由于它是由两个寄存器组成的，一个给发送用(TDR)，一个给接收用(RDR)，该寄存器兼具读和写的功能。TDR寄存器提供了内部总线和输出移位寄存器之间的并行接口(参见图248)。RDR寄存器提供了输入移位寄存器和内部总线之间的并行接口。 当使能校验位(USART_CR1中PCE位被置位)进行发送时，写到MSB的值(根据数据的长度不同，MSB是第7位或者第8位)会被后来的校验位该取代。 当使能校验位进行接收时，读到的MSB位是接收到的校验位。

```
void USART_SendData(USART_TypeDef* USARTx, uint16_t Data);
uint16_t USART_ReceiveData(USART_TypeDef* USARTx);
```

STM32的串口通信接口

波特比率寄存器(USART_BRR)



位31:16	保留位，硬件强制为0
位15:4	**DIV_Mantissa[11:0]**：USARTDIV的整数部分 这12位定义了USART分频器除法因子(USARTDIV)的整数部分。
位3:0	**DIV_Fraction[3:0]**：USARTDIV的小数部分 这4位定义了USART分频器除法因子(USARTDIV)的小数部分。

void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct);

STM32的串口通信接口

波特率计算方法

$$\text{Tx / Rx 波特率} = \frac{f_{PCLKx}}{(16 * USARTDIV)}$$

上式中，f_{PCLKx} 是给串口的时钟（PCLK1 用于 USART2、3、4、5，PCLK2 用于 USART1）；

USARTDIV 是一个无符号定点数。我们只要得到 USARTDIV 的值，就可以得到串口波特率寄存器 USART1->BRR 的值，反过来，我们得到 USART1->BRR 的值，也可以推导出 USARTDIV 的值。但我们更关心的是如何从 USARTDIV 的值得到 USART_BRR 的值，因为一般我们知道的是波特率，和 PCLKx 的时钟，要求的就是 USART_BRR 的值。

下面我们来介绍如何通过 USARTDIV 得到串口 USART_BRR 寄存器的值。假设我们的串口 1 要设置为 115200 的波特率，而 PCLK2 的时钟为 72M。这样，我们根据上面的公式有：

USARTDIV=72000000/(115200*16)= 39.0625

那么得到：

DIV_Fraction=16*0. 0625=1=0X01;

DIV_Mantissa=39=0X27;

这样，我们就得到了 USART1->BRR 的值为 0X0271。只要设置串口 1 的 BRR 寄存器值为 0X0271 就可以得到 115200 的波特率。

参考STM32F10X-中文参考手册.pdf P524

STM32的串口通信接口　串口配置的一般步骤

①串口时钟使能，GPIO时钟使能:RCC_APB2PeriphClockCmd();
②串口复位:USART_DeInit(); 这一步不是必须的
③GPIO端口模式设置:GPIO_Init();
④串口参数初始化：USART_Init();
⑤开启中断并且初始化NVIC（如果需要开启中断才需要这个步骤）
　　NVIC_Init();
　　USART_ITConfig();
⑥使能串口:USART_Cmd();
⑦编写中断处理函数：USARTx_IRQHandler();
⑧串口数据收发：
void USART_SendData();//发送数据到串口，DR
uint16_t USART_ReceiveData();//接受数据，从DR读取接受到的数据
⑨串口传输状态获取：
FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG);
void USART_ClearITPendingBit(USART_TypeDef* USARTx, uint16_t USART_IT);

STM32的串口通信接口

串口操作相关函数

void USART_Init(); //串口初始化：波特率，数据字长，奇偶校验，硬件流控以及收发使能
void USART_Cmd();//使能串口
void USART_ITConfig();//使能相关中断

void USART_SendData();//发送数据到串口，DR
uint16_t USART_ReceiveData();//接受数据，从DR读取接受到的数据

FlagStatus USART_GetFlagStatus();//获取状态标志位
void USART_ClearFlag();//清除状态标志位
ITStatus USART_GetITStatus();//获取中断状态标志位
void USART_ClearITPendingBit();//清除中断状态标志位

STM32的串口通信接口

USART相关库函数-USART_DeInit

函数名	USART_DeInit
函数原形	void USART_DeInit(USART_TypeDef* USARTx)
功能描述	将外设 USARTx 寄存器重设为缺省值
输入参数	USARTx： x 可以是 1,2…，来选择 USART 外设
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_APB2PeriphResetCmd() RCC_APB1PeriphResetCmd()

STM32的串口通信接口

USART相关库函数-USART_Init

函数名	USART_Init
函数原形	void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct)
功能描述	根据 USART_InitStruct 中指定的参数初始化外设 USARTx 寄存器
输入参数 1	USARTx： x 可以是 1， 2 …，来选择 USART 外设
输入参数 2	USART_InitStruct：指向结构 USART_InitTypeDef 的指针，包含了外设 USART 的配置信息。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

USART_WordLength：

USART_WordLength	描述
USART_WordLength_8b	8 位数据
USART_WordLength_9b	9 位数据

USART_StopBits：

USART_StopBits	描述
USART_StopBits_1	在帧结尾传输 1 个停止位
USART_StopBits_0_5	在帧结尾传输 0.5 个停止位
USART_StopBits_2	在帧结尾传输 2 个停止位
USART_StopBits_1_5	在帧结尾传输 1.5 个停止位

USART_Parity：

USART_Parity	描述
USART_Parity_No	无校验
USART_Parity_Even	偶模式
USART_Parity_Odd	奇模式

USART_Mode：

USART_Mode	描述
USART_Mode_Tx	发送使能
USART_Mode_Rx	接收使能

USART_HardwareFlowControl：

USART_HardwareFlowControl	描述
USART_HardwareFlowControl_None	硬件流控制失能
USART_HardwareFlowControl_RTS	发送请求 RTS 使能
USART_HardwareFlowControl_CTS	清除发送 CTS 使能
USART_HardwareFlowControl_RTS_CTS	RTS 和 CTS 使能

STM32的串口通信接口

USART相关库函数-USART_Cmd

函数名	USART_ Cmd
函数原形	void USART_Cmd(USART_TypeDef* USARTx, FunctionalState NewState)
功能描述	使能或者失能 USART 外设
输入参数 1	USARTx： x 可以是 1， 2 ..，来选择 USART 外设
输入参数 2	NewState: 外设 USARTx 的新状态 这个参数可以取： ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

STM32的串口通信接口

USART相关库函数-USART_SendData

函数名	USART_ SendData
函数原形	void USART_SendData(USART_TypeDef* USARTx, uint16_t Data);
功能描述	通过外设 USARTx 发送单个数据
输入参数 1	USARTx： x 可以是 1， 2 …，来选择 USART 外设
输入参数 2	Data: 待发送的数据
输出参数	无
返回值	无
先决条件	无
被调用函数	无

STM32的串口通信接口

USART相关库函数-USART_ReceiveData

函数名	USART_ ReceiveData
函数原形	u8 USART_ReceiveData(USART_TypeDef* USARTx)
功能描述	返回 USARTx 最近接收到的数据
输入参数	USARTx： x 可以是 1， 2 …，来选择 USART 外设
输出参数	无
返回值	接收到的字
先决条件	无
被调用函数	无

STM32的串口通信接口

USART相关库函数-USART_GetFlagStatus

函数名	USART_ GetFlagStatus
函数原形	FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG)
功能描述	检查指定的 USART 标志位设置与否
输入参数 1	USARTx： x 可以是 1， 2 …，来选择 USART 外设
输入参数 2	USART_FLAG：待检查的 USART 标志位 参阅 Section： USART_FLAG 查阅更多该参数允许取值范围
输出参数	无
返回值	USART_FLAG 的新状态（SET 或者 RESET）
先决条件	无
被调用函数	无

USART_FLAG:

USART_FLAG	描述
USART_FLAG_CTS	CTS 标志位
USART_FLAG_LBD	LIN 中断检测标志位
USART_FLAG_TXE	发送数据寄存器空标志位
USART_FLAG_TC	发送完成标志位
USART_FLAG_RXNE	接收数据寄存器非空标志位
USART_FLAG_IDLE	空闲总线标志位
USART_FLAG_ORE	溢出错误标志位
USART_FLAG_NE	噪声错误标志位
USART_FLAG_FE	帧错误标志位
USART_FLAG_PE	奇偶错误标志位

STM32的串口通信接口

USART相关库函数-USART_ClearFlag

函数名	USART_ ClearFlag
函数原形	void USART_ClearFlag(USART_TypeDef* USARTx, uint16_t USART_FLAG)
功能描述	清除 USARTx 的待处理标志位
输入参数 1	USARTx： x 可以是 1， 2 …，来选择 USART 外设
输入参数 2	USART_FLAG：待清除的 USART 标志位
输出参数	无
返回值	无
先决条件	无
被调用函数	无

STM32的串口通信接口

USART相关库函数-USART_ITConfig

函数名	USART_ITConfig
函数原形	void USART_ITConfig(USART_TypeDef* USARTx, uint16_t USART_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 USART 中断
输入参数 1	USARTx： x 可以是 1， 2 ..，来选择 USART 外设
输入参数 2	USART_IT：待使能或者失能的 USART 中断源 参阅 Section： USART_IT 查阅更多该参数允许取值范围
输入参数 3	NewState： USARTx 中断的新状态 这个参数可以取： ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

USART_IT：

USART_IT	描述
USART_IT_PE	奇偶错误中断
USART_IT_TXE	发送中断
USART_IT_TC	传输完成中断
USART_IT_RXNE	接收中断
USART_IT_IDLE	空闲总线中断
USART_IT_LBD	LIN 中断检测中断
USART_IT_CTS	CTS 中断
USART_IT_ERR	错误中断

STM32的串口通信接口

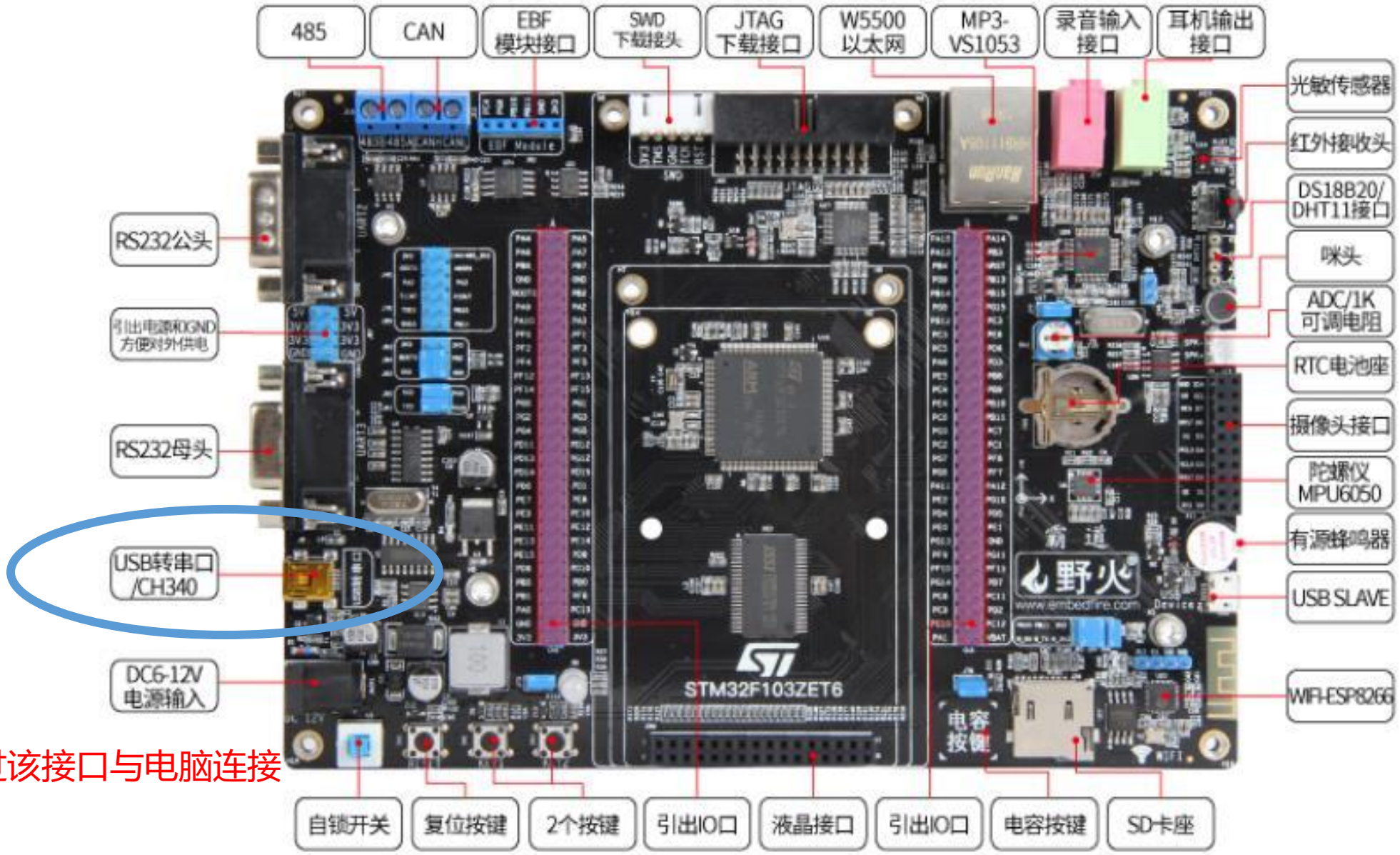
USART相关库函数-USART_GetITStatus

函数名	USART_ GetITStatus
函数原形	ITStatus USART_GetITStatus(USART_TypeDef* USARTx, uint16_t USART_IT)
功能描述	检查指定的 USART 中断发生与否
输入参数 1	USARTx： x 可以是 1， 2 ..，来选择 USART 外设
输入参数 2	USART_IT：待检查的 USART 中断源
输出参数	无
返回值	USART_IT 的新状态
先决条件	无
被调用函数	无

USART_IT：

USART_IT	描述
USART_IT_PE	奇偶错误中断
USART_IT_TXE	发送中断
USART_IT_TC	发送完成中断
USART_IT_RXNE	接收中断
USART_IT_IDLE	空闲总线中断
USART_IT_LBD	LIN 中断探测中断
USART_IT_CTS	CTS 中断
USART_IT_ORE	溢出错误中断
USART_IT_NE	噪音错误中断
USART_IT_FE	帧错误中断

实验内容-串口通信实验



通过该接口与电脑连接

谢谢大家