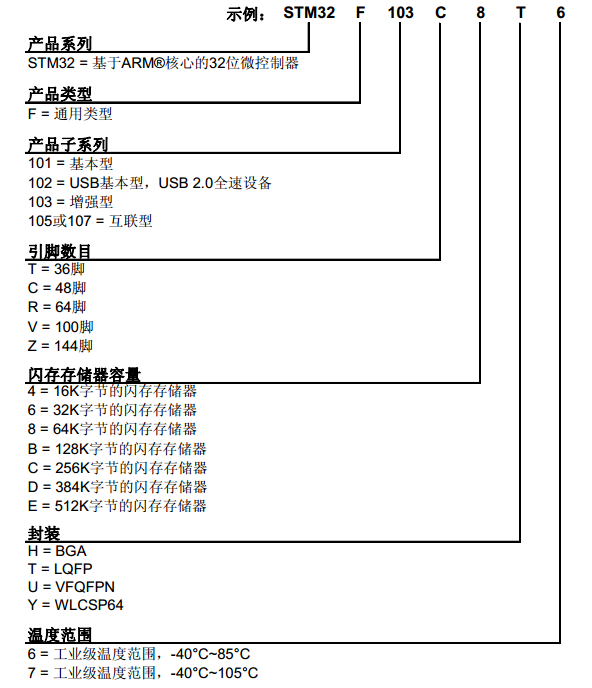
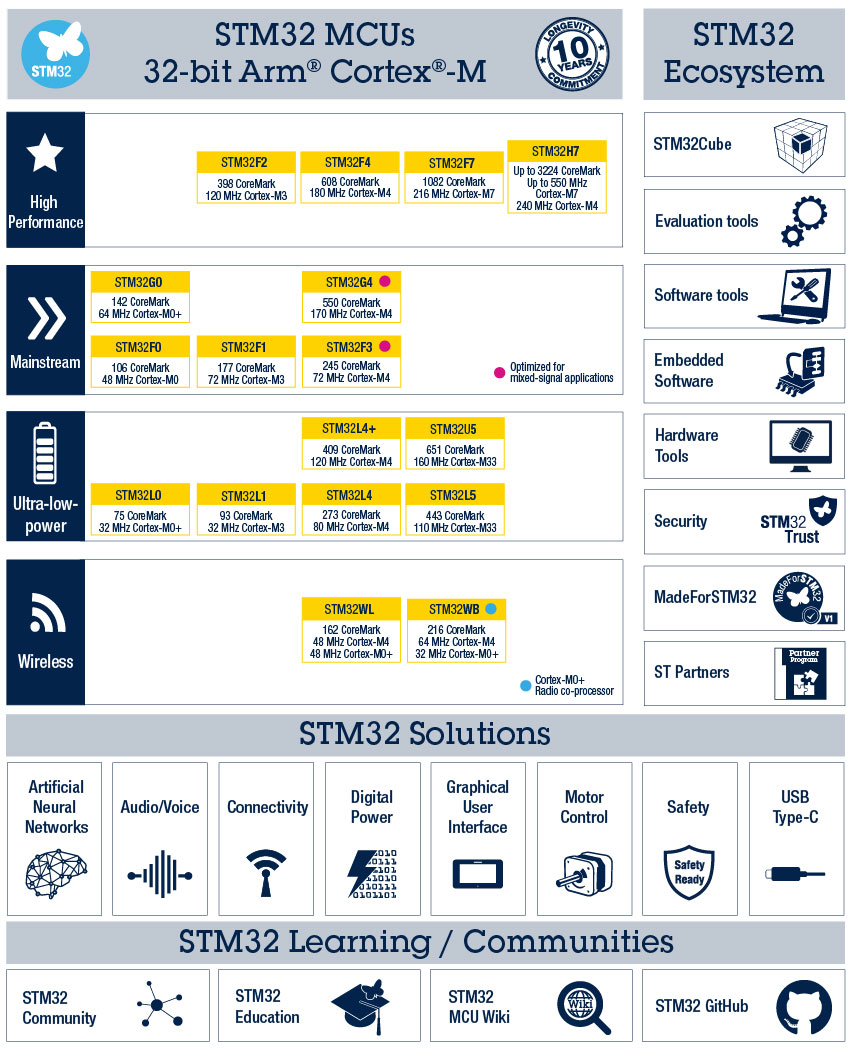
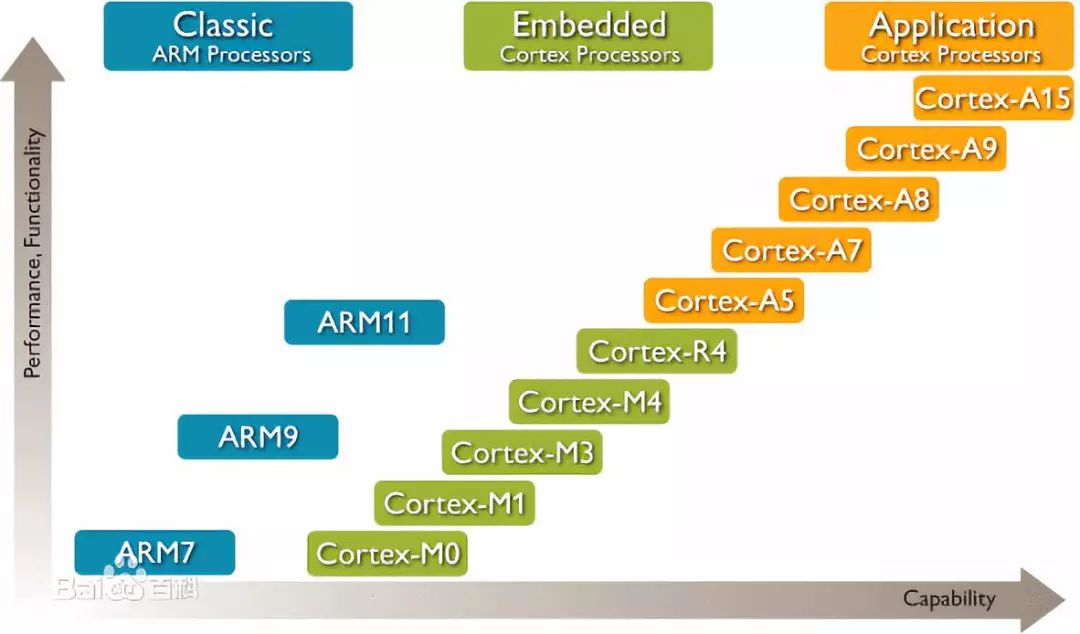
# 1. STM32简介

## 1.1 STM32命名规范







ARM既指ARM公司，也指ARM处理器内核。

A：Application，主要偏向于应用

R：Realtime，偏向于实时性

M：Microcontroller，微控制器

## 1.2 STM32F103C8T6特性

系列：主流系列STM32F1

内核：ARM Cortex-M3

主频：72MHz

RAM：20K（SRAM）

ROM：64K（Flash）

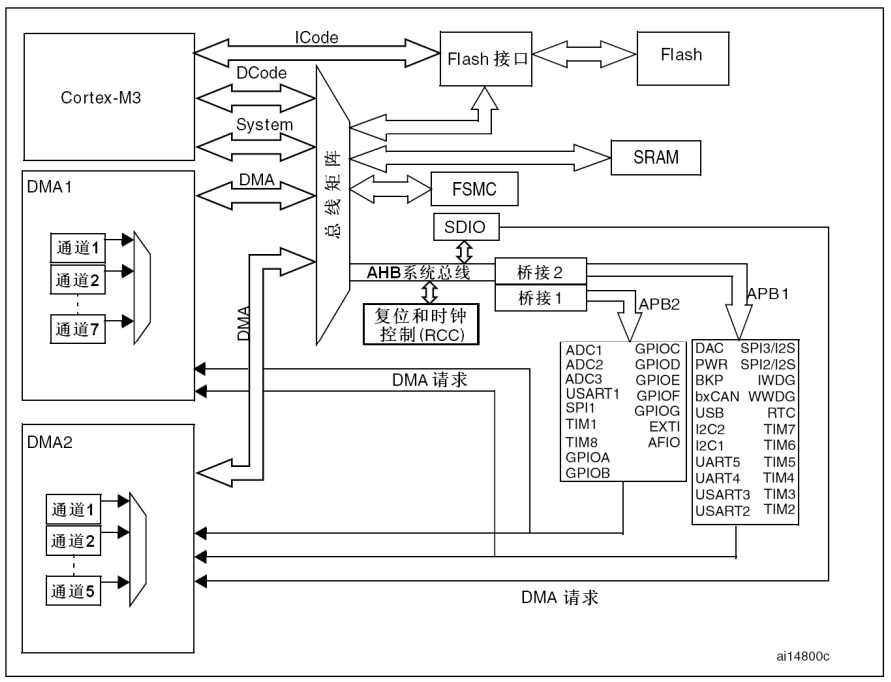
供电：2.0~3.6V（标准3.3V）

封装：LQFP48

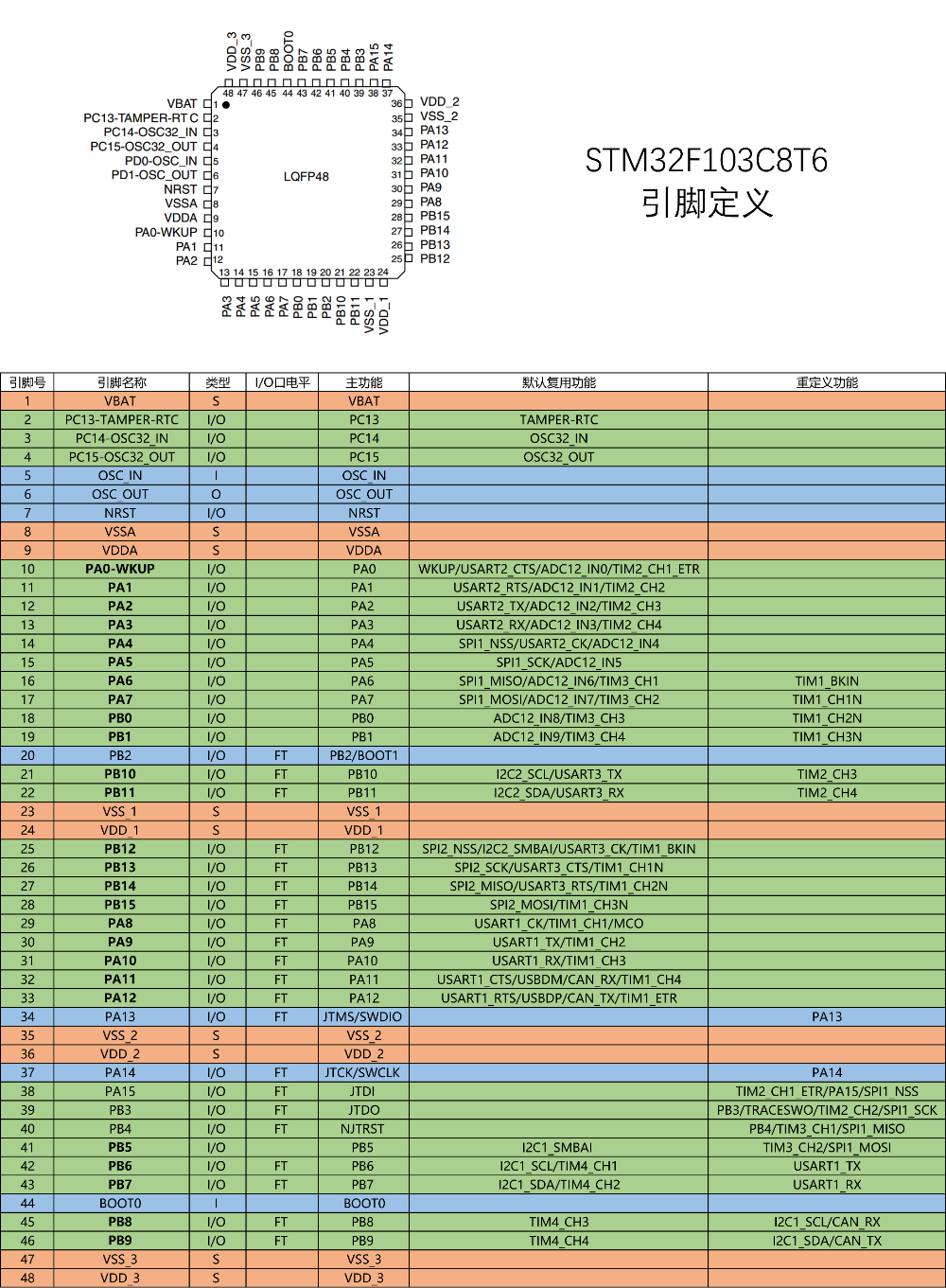
## 1.3 片上资源/外设



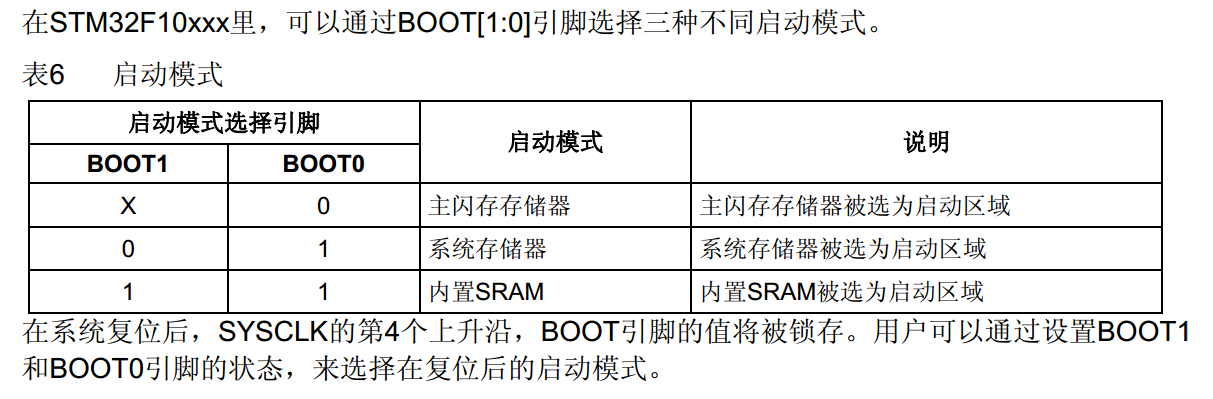
## 1.4 系统架构



## 1.5 引脚定义



## 1.6 启动配置

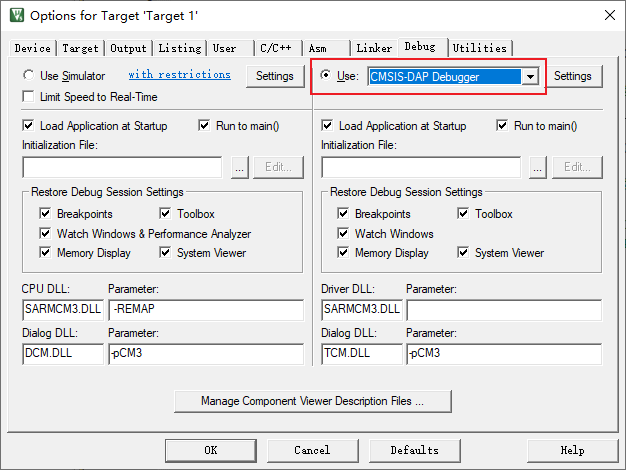


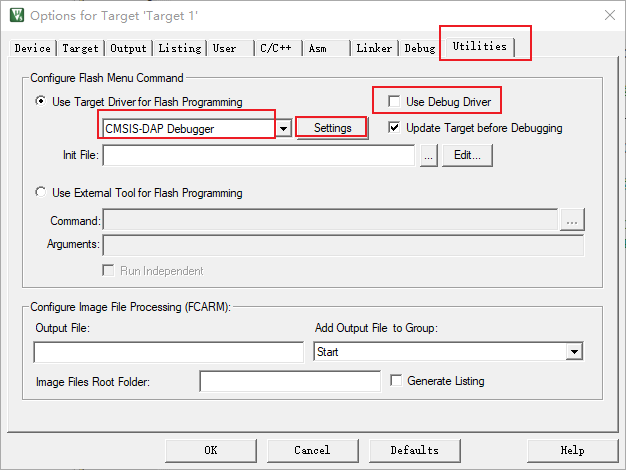
(1)最常用配置：BOOT0设为0，正常的执行Flash闪存里面的程序，使用STLINK、DAPlink进行下载调试

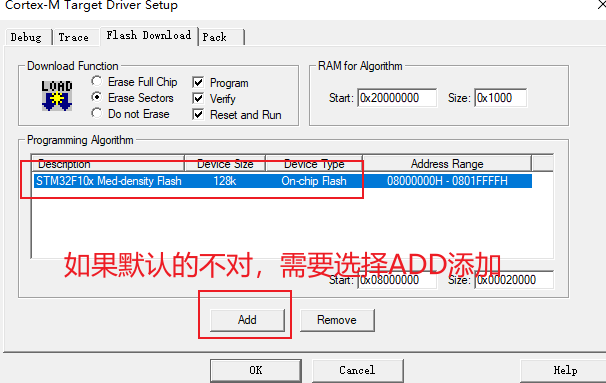
(2)串口下载模式：BOOT0设为1，BOOT1设为0，系统存储器(存放Bootloader程序，接收串口的数据，然后刷新到主闪存中)

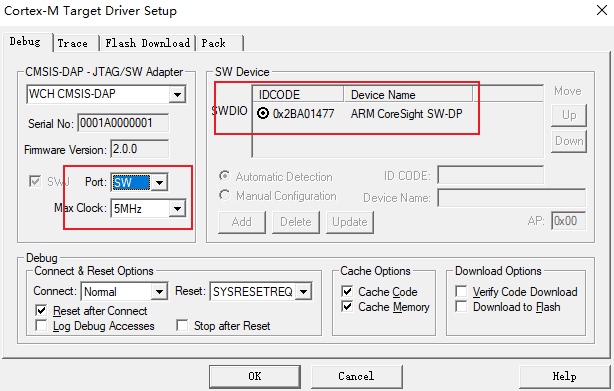
(3)最后一种模式，主要用于程序调试，用的比较少

## 1.7 DAPLink烧录

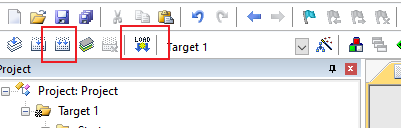








每次下载程序，需要先rebuild编译，在下载！



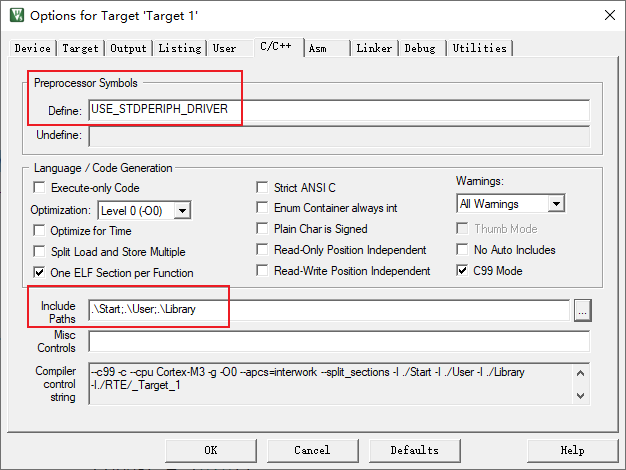
# 2. STM32工程模板

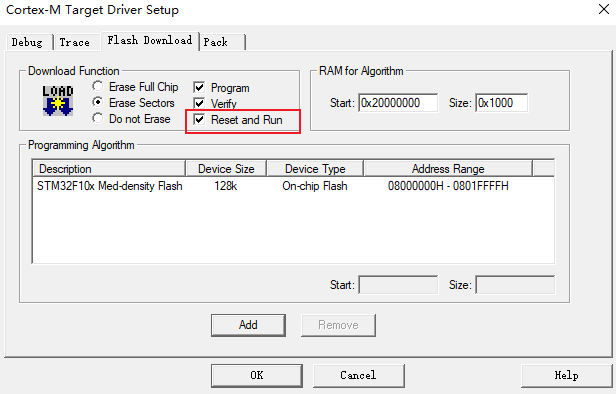
## 2.1 Keil配置工作模板

使用ST库函数的开发方式进行寄存器配置。

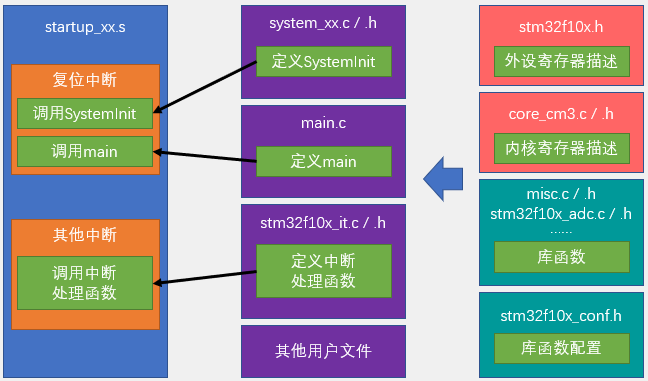
新建工程步骤：

* 建立工程文件夹，Keil中新建工程，选择型号
* 工程文件夹里建立Start、Library、User等文件夹，复制固件库里面的文件到工程文件夹
* 工程里对应建立Start、Library、User等同名称的分组，然后将文件夹内的文件添加到工程分组里
* 工程选项，C/C++，Include Paths内声明所有包含头文件的文件夹
* 工程选项，C/C++，Define内定义USE\_STDPERIPH\_DRIVER (标准外设驱动字符串，使用库函数开发时，这个是必须添加的)
* 工程选项，Debug，下拉列表选择对应调试器，Settings，Flash Download里勾选Reset and Run

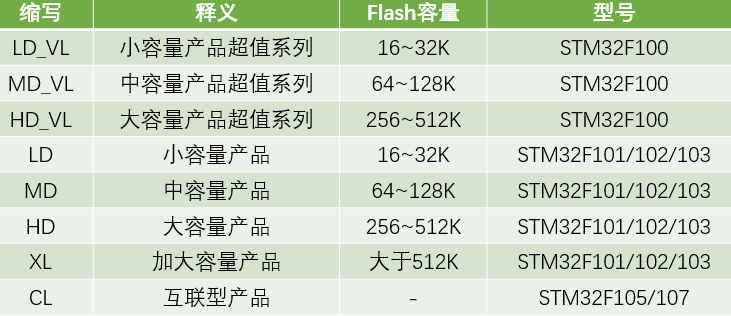


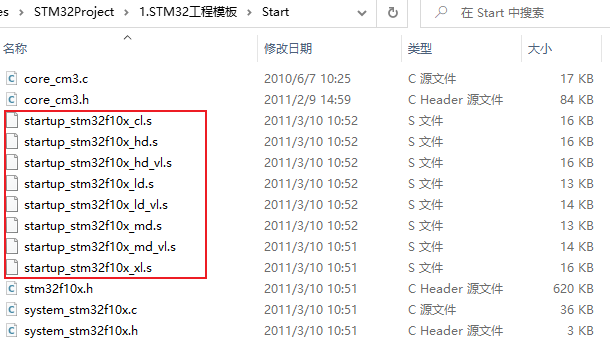


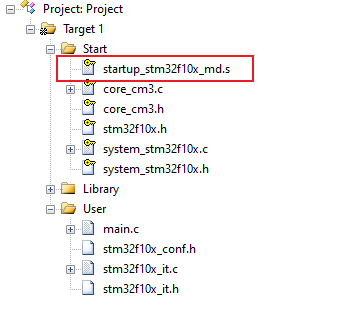
工程架构：

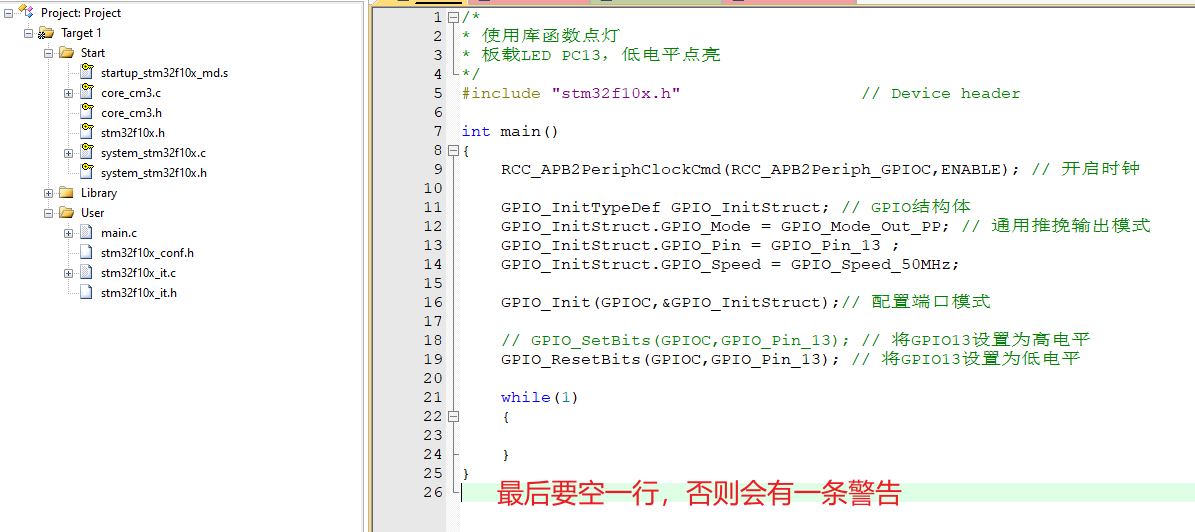


根据不同的产品型号，选择不同的文件



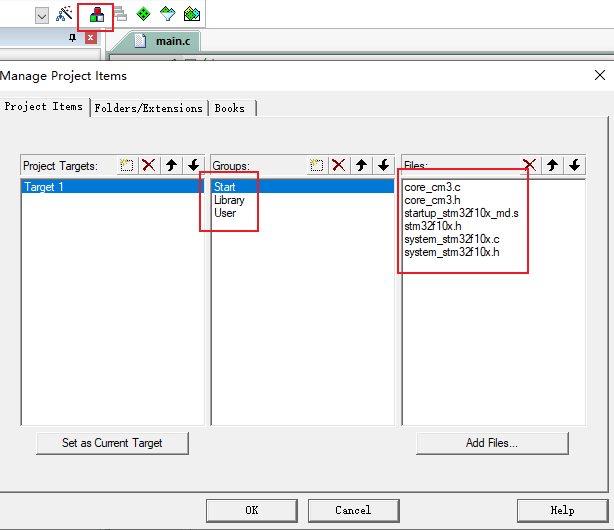




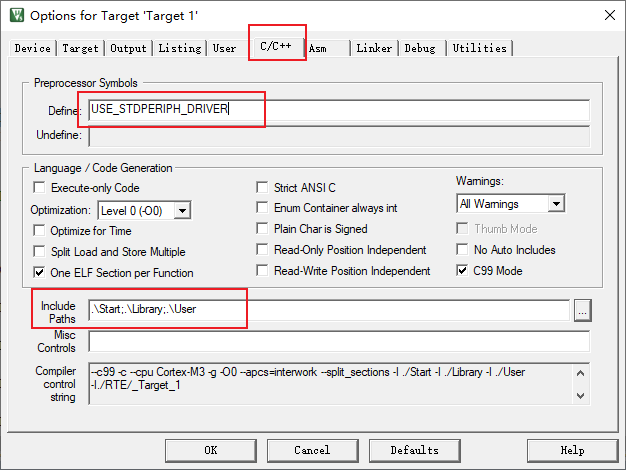


## 2.2 创建工程步骤总结

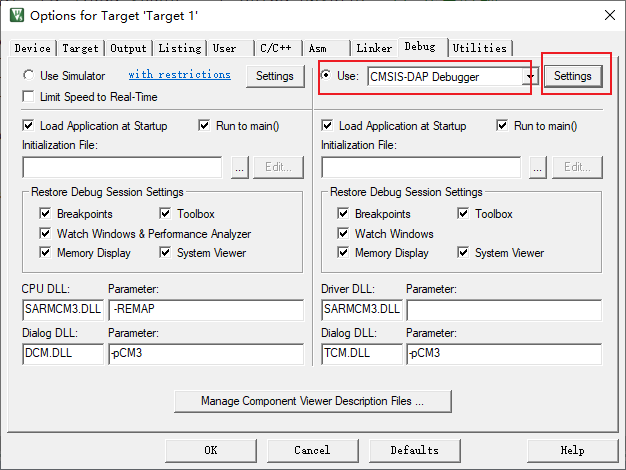
1. 快速创建Start、Library、User组，添加相关的文件

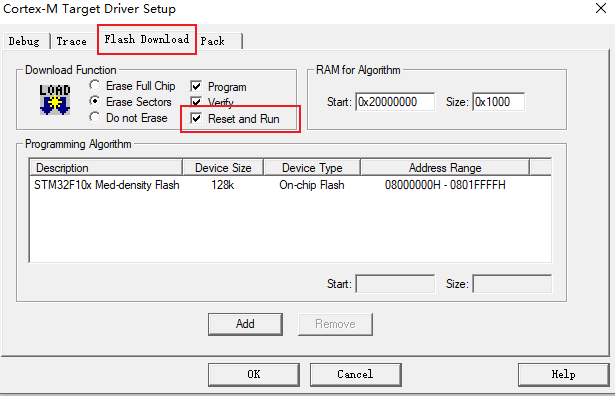


2.点击魔术棒按钮，添加第一步文件的路径，并添加这句话USE\_STDPERIPH\_DRIVER



3. 选择对应的调试器(DAPLink)

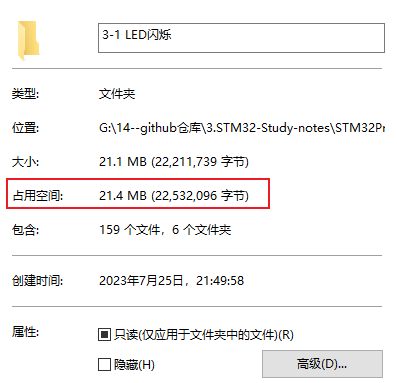
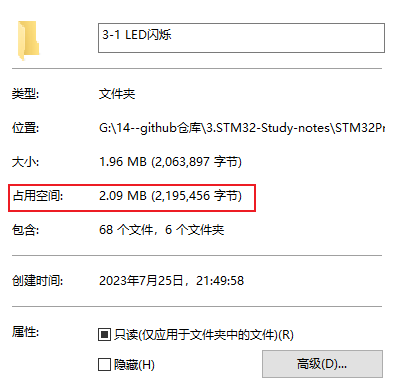




4.小工具：keilkill，是一个批处理文件，可以把工程编译产生的中间文件都删掉



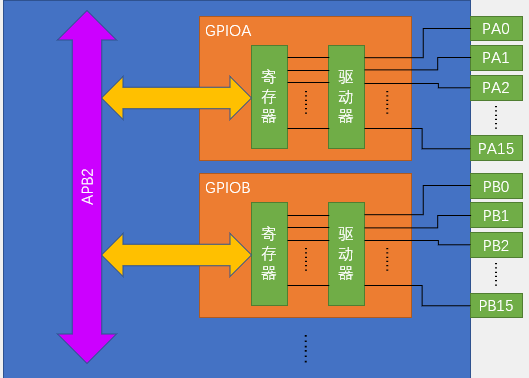
使用前后的对比：减小工程体积，之后可以添加到压缩包分享发送给别人

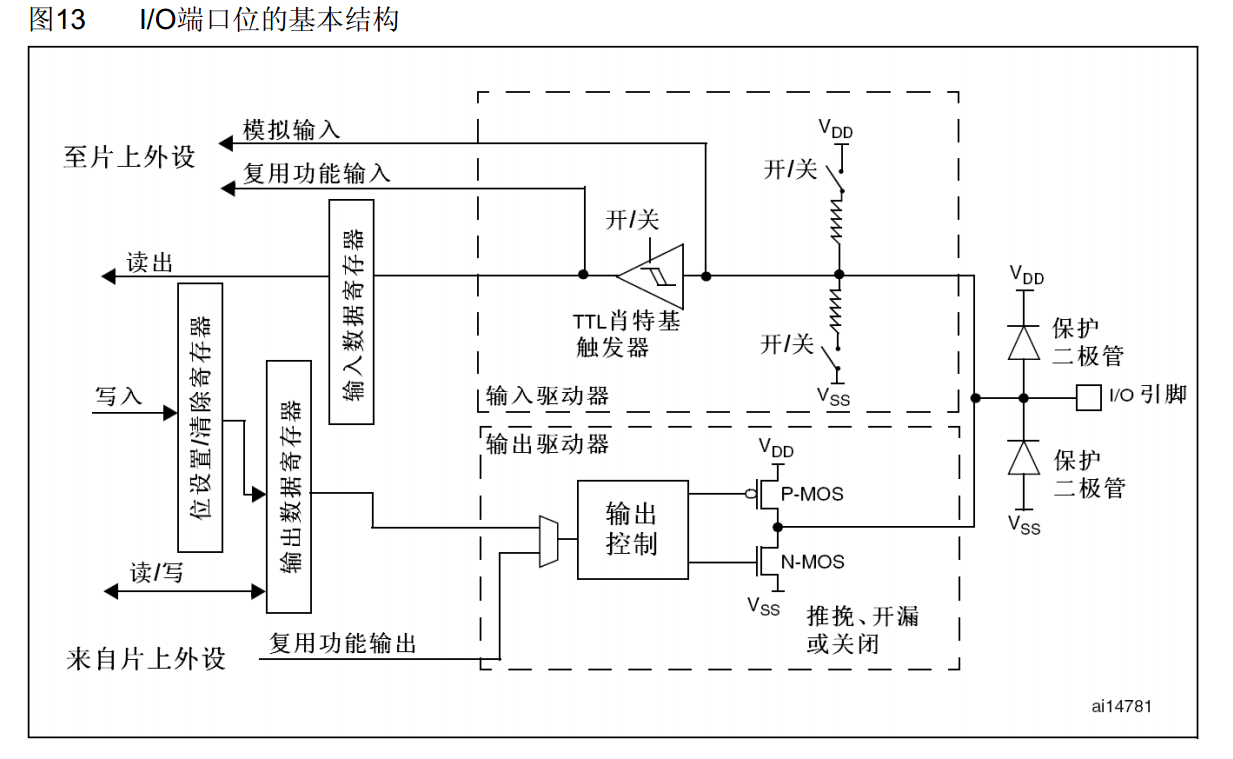
# 3. GPIO配置及应用

## 3.1 GPIO工作模式介绍

GPIO基本结构：

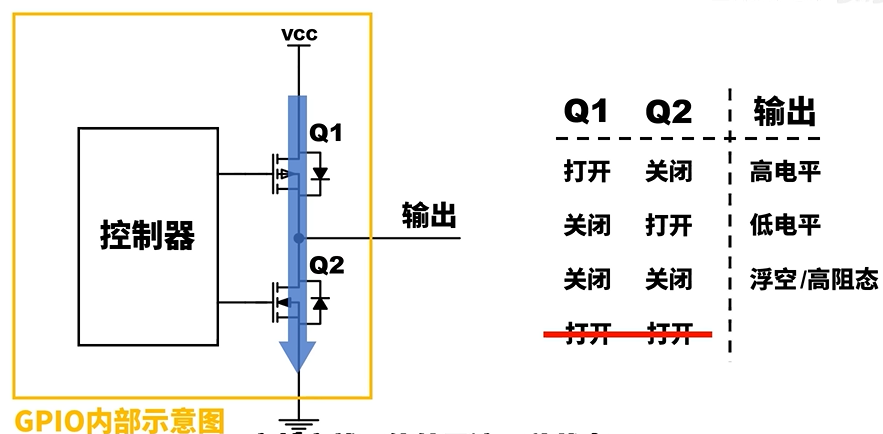


GPIO位结构：





单片机的GPIO配置为输出模式时，会有两种模式：推挽模式、开漏模式。



上下MOS同时导通时，发生短路，这种状态需要避免。

推完输出模式理解：GPIO输出高低电平时，电流的一个动作而已

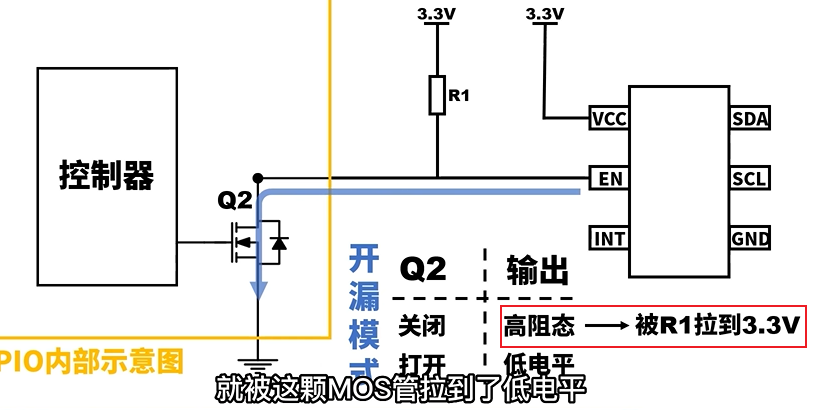
推：IO输出高电平时，把电流推出去

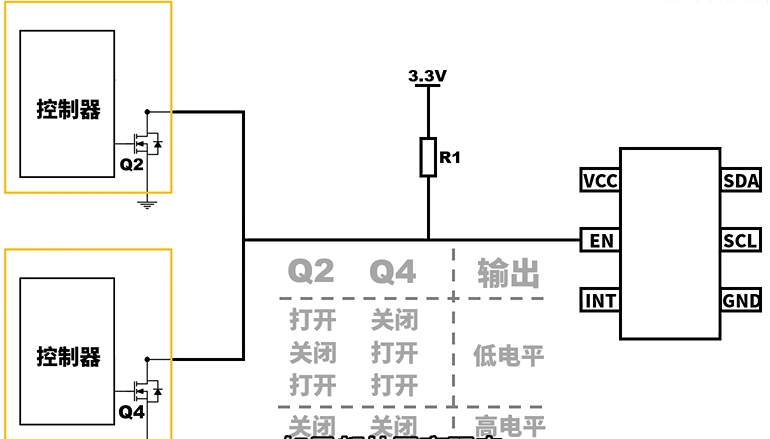
挽：IO输出低电平时，外界电流流进来，把电流挽回来。

开漏模式：低电平和高阻态(被外接上拉电阻拉到高电平)的组合，称为开漏输出。

需要在外面接一个上拉电阻，常用于几个GPIO控制一个输入。

开漏模式输出高电平相当于高阻态，没有驱动能力；低电平有驱动能力。





只要有任意一个GPIO输出低电平，enable就是低电平

如果都处于高阻态，enable就是高电平

## 3.2 LED闪烁



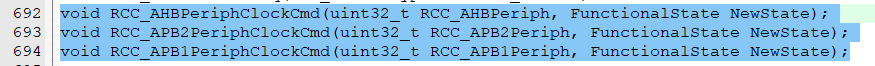
操作STM32的GPIO共需要3个步骤：

1.使用RCC开启GPIO的时钟

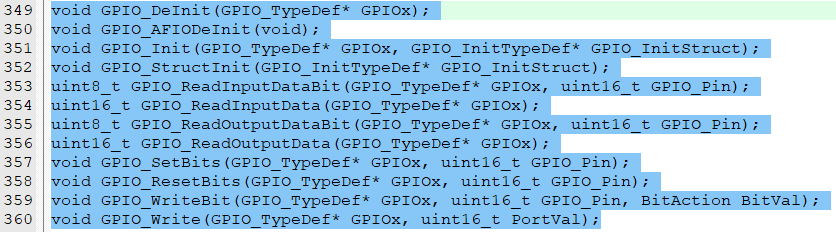
2.使用GPIO\_Init函数初始化GPIO

3.使用输出或者输入的函数控制GPIO口

1.RCC(Reset Clock Controller)，复位与时钟控制器，常用RCC AHB外设时钟控制、RCC APB2外设时钟控制、RCC APB1外设时钟控制

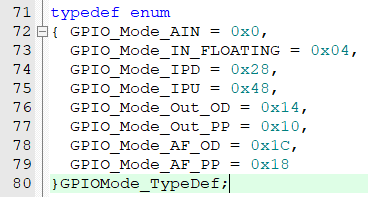


2.GPIO常用库函数



GPIO\_Init()函数：用结构体的参数来初始化GPIO口，需要先定义一个结构体变量，然后给结构体赋值，最后调用这个函数。

GPIO工作模式：



typedef enum

{ GPIO\_Mode\_AIN = 0x0, // Analog IN，模拟输入

GPIO\_Mode\_IN\_FLOATING = 0x04, // 浮空输入

GPIO\_Mode\_IPD = 0x28, // In Pull Down，下拉输入

GPIO\_Mode\_IPU = 0x48, // In Pull Up，上拉输入

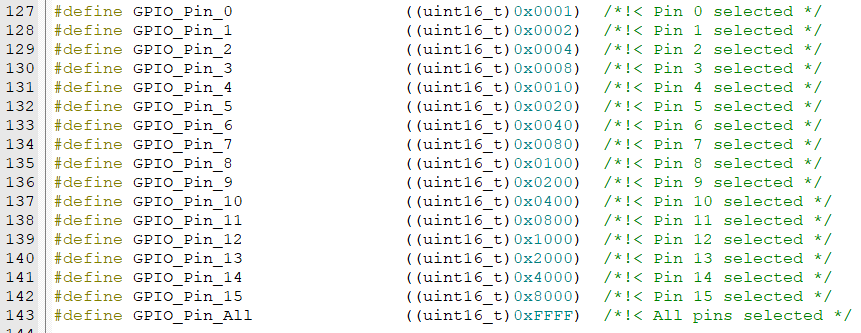
GPIO\_Mode\_Out\_OD = 0x14, // Out Open Drain，开漏输出

GPIO\_Mode\_Out\_PP = 0x10, // Out Push pull，推挽输出

GPIO\_Mode\_AF\_OD = 0x1C, // Atl Open Drain，复用开漏

GPIO\_Mode\_AF\_PP = 0x18 // Atl Push Pull，复用推挽

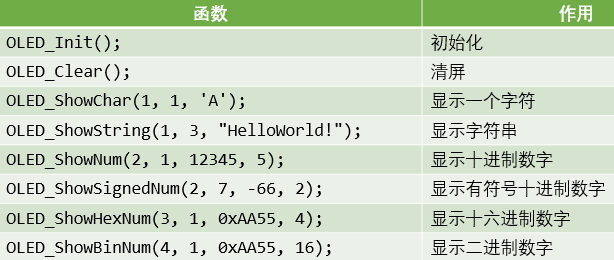
}GPIOMode\_TypeDef;



void GPIO\_SetBits(GPIO\_TypeDef\* GPIOx, uint16\_t GPIO\_Pin); // 把指定的端口设置为高电平

void GPIO\_ResetBits(GPIO\_TypeDef\* GPIOx, uint16\_t GPIO\_Pin); 设置GPIO输出低电平

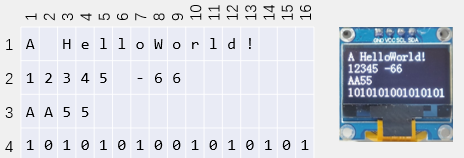
# 4. OLED显示屏使用



参数1：指定起始行；

参数2：指定起始列

0.96寸OLED，分辨率：128\*64，把分辨率设置为4行16列，字符大小8\*8



STM32的引脚上电后，如果不初始化，默认为浮空输入的模式。

SCL --PB8

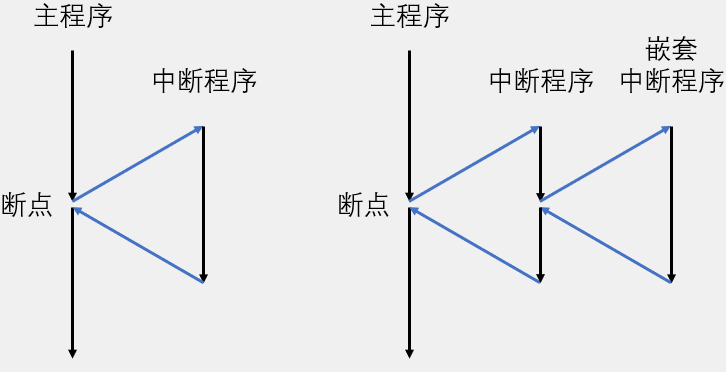
SDA --PB9

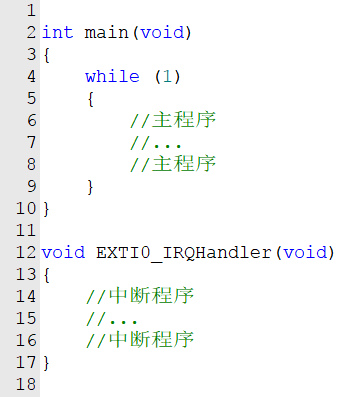
# 5. EXIT外部中断

使用中断系统可以加大的提高程序的运行效率，如果没有中断系统，为了防止外部中断被忽略或者串口数据被覆盖，主程序就只能不断地查询是否有这些事件发生。

比如如果没有定时器中断，主程序只能使用delay函数，才能实现定时的功能。

有了中断系统后，主程序就可以放心执行其他事情，有中断的时候再去处理。





抢占优先级高的可以中断嵌套，响应优先级高的可以优先排队。

抢占优先级和响应优先级均相同的按**中断号**排队。

中断触发方式：上升沿/下降沿/双边沿(上升沿和下降沿都可以触发)/软件触发(如何进行软件触发设计？程序里执行一句代码，就可以触发中断)

触发响应方式：中断响应/事件响应

通道数：16个GPIO\_PIN、外加PVD输出、RTC闹钟、USB唤醒、以太网唤醒

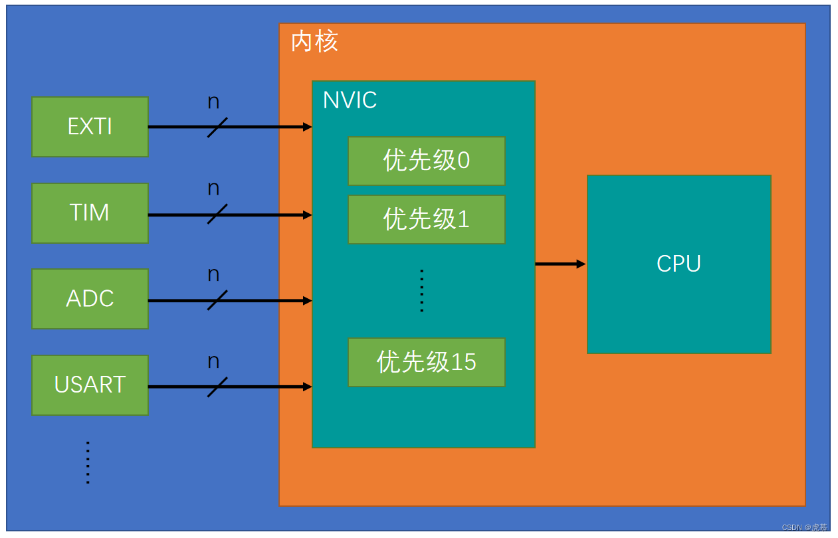
中断响应：引脚电平变化触发中断

事件响应：不会触发中断，而是触发别的外设操作，属于外设之间的联合工作，比如触发ADC转换、DMA

注意：

1. 进入中断函数的时候要检查中断标志位，退出的时候清零中断标志位。

2. 中断函数一般执行简短快速的代码，如操作中断标志位。



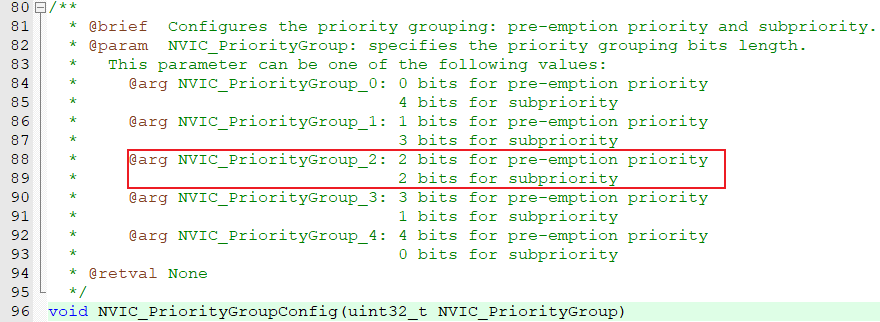
NVIC用于统一管理中断和分配中断优先级，属于内核外设，是CPU的小助手，可以让CPU专注于运算。相当于医院里的叫号系统(给病人将进行排号，CPU相当于医生，只负责看病)。

## 5.1 NVIC优先级分组

优先级：抢占优先级、响应优先级。

* NVIC的中断优先级由优先级寄存器的4位（0~15）决定，这4位可以进行切分，分为高n位的抢占优先级和低4-n位的响应优先级(插队)
* 抢占优先级高的可以中断嵌套，响应优先级高的可以优先排队，抢占优先级和响应优先级均相同的按中断号排队





如果中断不多，很难导致中断冲突时，可以随意选择优先级分组，那个都行。

## 5.2 EXIT(Extern Interrupt)外部中断

EXIT的基本功能：检测引脚的电平变化，申请中断。

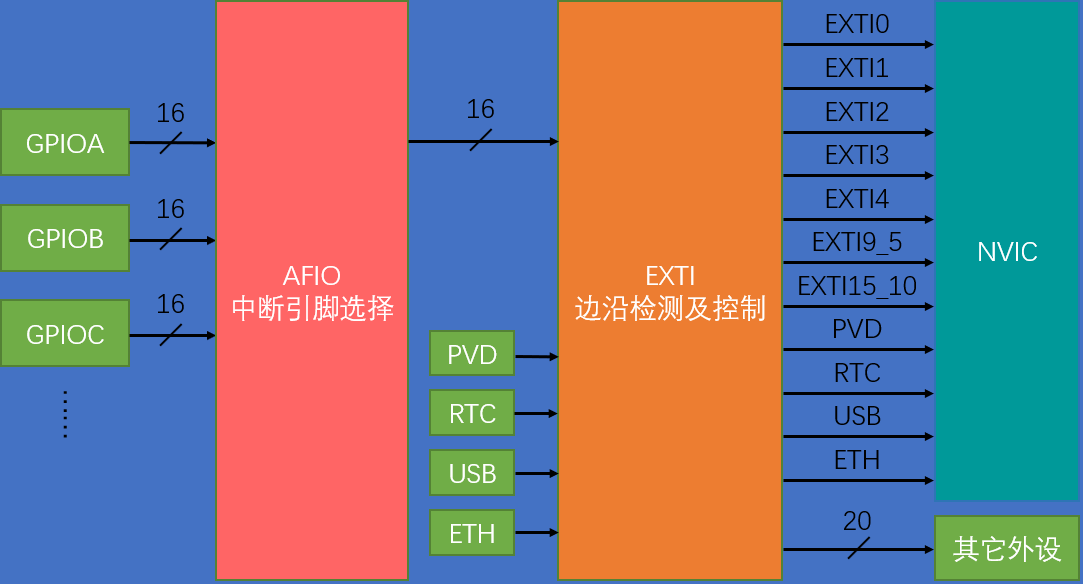
支持的GPIO口：所有的GPIO口，但相同的Pin不能同时触发中断。

比如PA0和PB0不能同时用，端口GPIO\_Pin\*只能选择一个作为中断引脚。

PA0和PA1可以同时触发中断。

通道数：16个GPIO\_pin(也就是只能有16个引脚可以触发中断)，外加PVD输出(电源电压监测)、RTC闹钟、USB唤醒、以太网唤醒。

EXIT基本结构：



AFIO中断引脚选择：在GPIO(A~G)外设的16个GPIO\_Pin中选择一个连接到后面的EXIT的通道里。

## 5.3 常用的触发外部中断的硬件模块

对于STM32来说，想要获取的信号是外部驱动的很快的突发信号，但是又不需要一直进行检测。

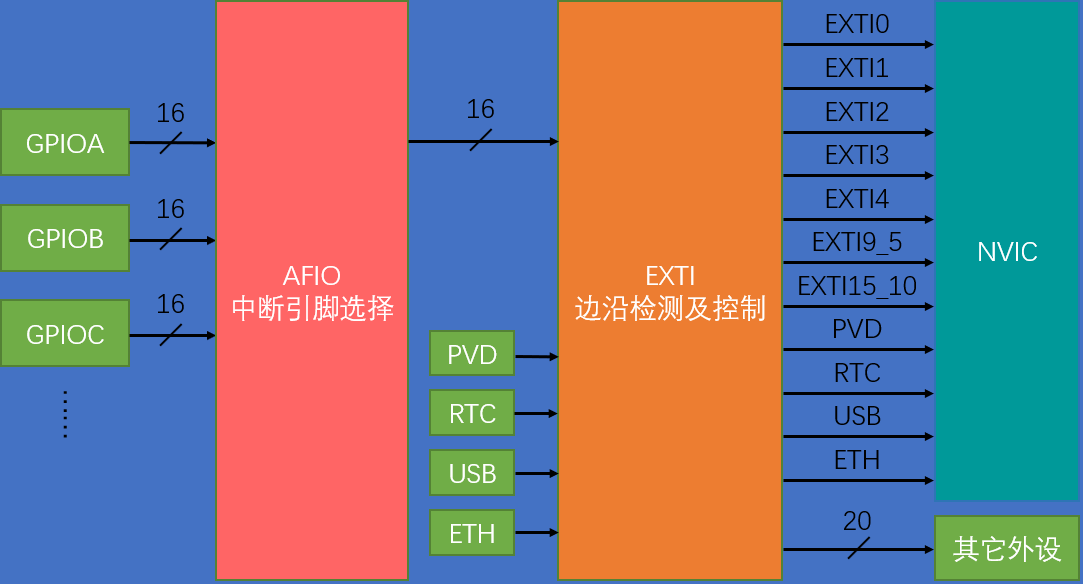
不需要经常使用、不需要一直检测的模块：

1. 按键(注意抖动问题)，不推荐，可以使用定时器中断

2. 旋转编码器

3. 红外遥控接收器

## 5.4 外部中断程序设计步骤



涉及到的外设：RCC、GPIO、AFIO、EXTI、NVIC

1. 打开RCC时钟，不打开时钟，外设是无法进行工作的

2. 配置GPIO，选择需要的端口，设为输入模式(推荐浮空、上拉、下拉)

3. 配置AFIO，选择用的这一路GPIO，连接到后面的EXTI

4. 配置EXTI，选择外部中断触发方式，选择触发响应方式(一般选择中断响应)

5. 配置NVIC，给中断选择一个合适的优先级

配置完之后，写中断执行函数：

在STM32中，中断函数的名字都是固定的，每个中断通道都对应一个中断函数。

中断函数的名字可以参考启动文件中的“stratup\_stm32f10x\_md文件

配置AFIO相关函数



void GPIO\_PinRemapConfig(uint32\_t GPIO\_Remap, FunctionalState NewState);

参数1：选择重映射方式

参数2：新的状态

void GPIO\_EXTILineConfig(uint8\_t GPIO\_PortSource, uint8\_t GPIO\_PinSource);

配置AFIO的数据选择器，来选择想要的中断引脚

注意：主函数和中断函数不要操控同一个硬件(比如OLED显示)，避免不必要的冲突，中断函数只执行突发的事件，需要快速的执行，如操作变量、中断标志位。