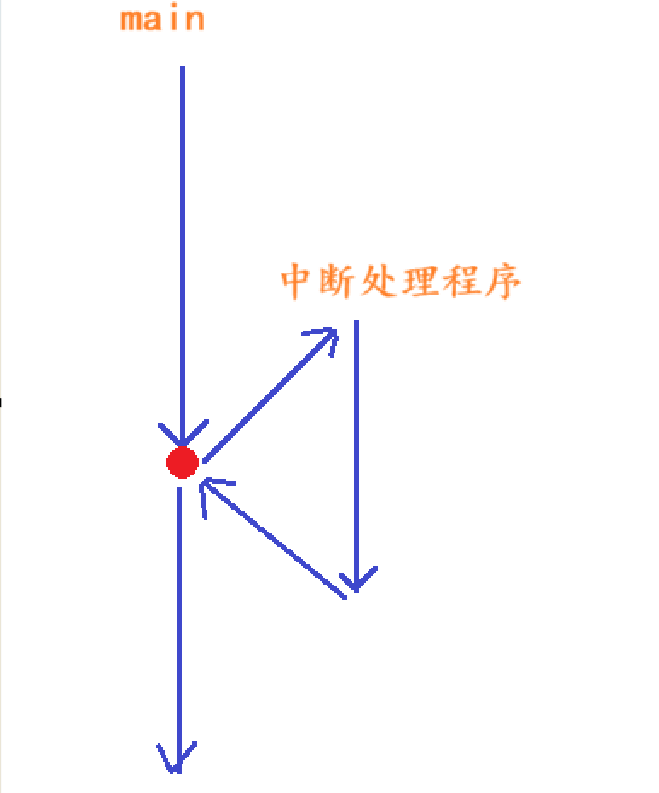
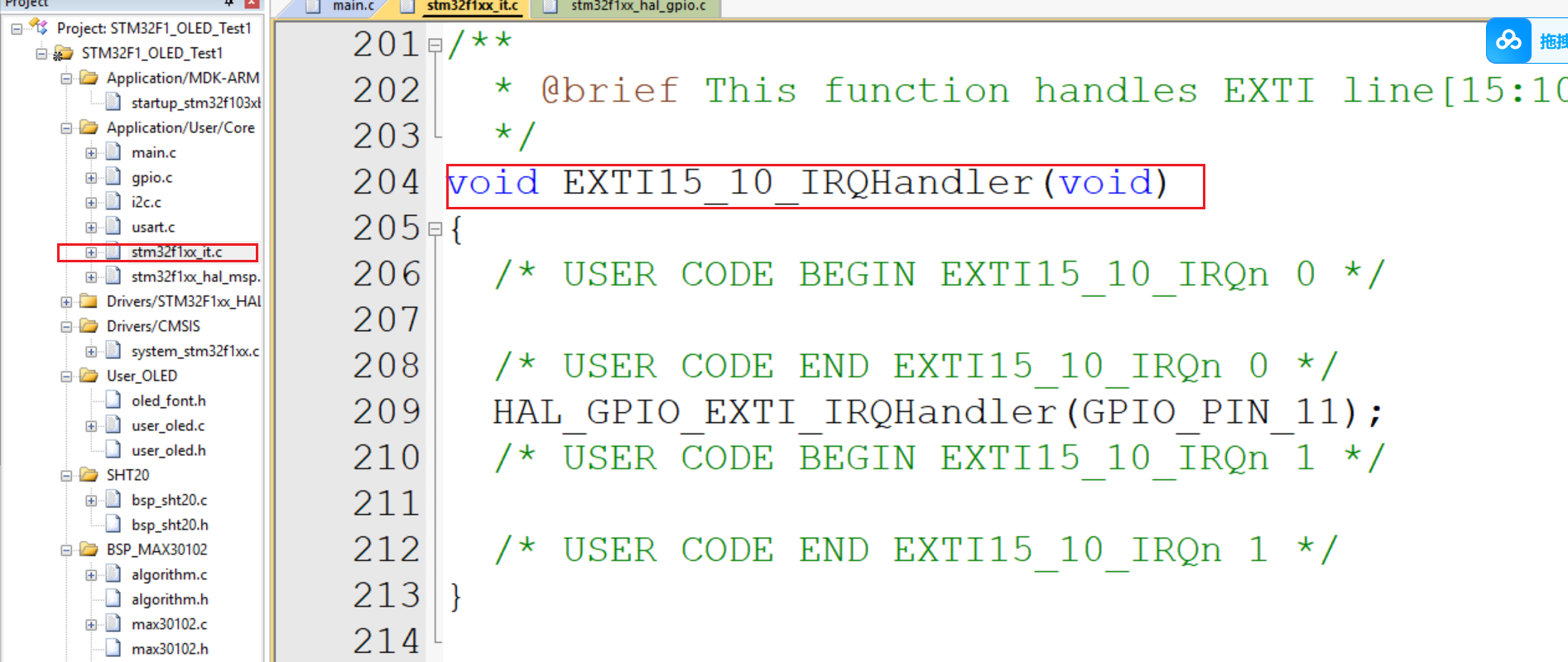
我先要说的是非常简单，简单到不行：

先来看一下中断的原理。



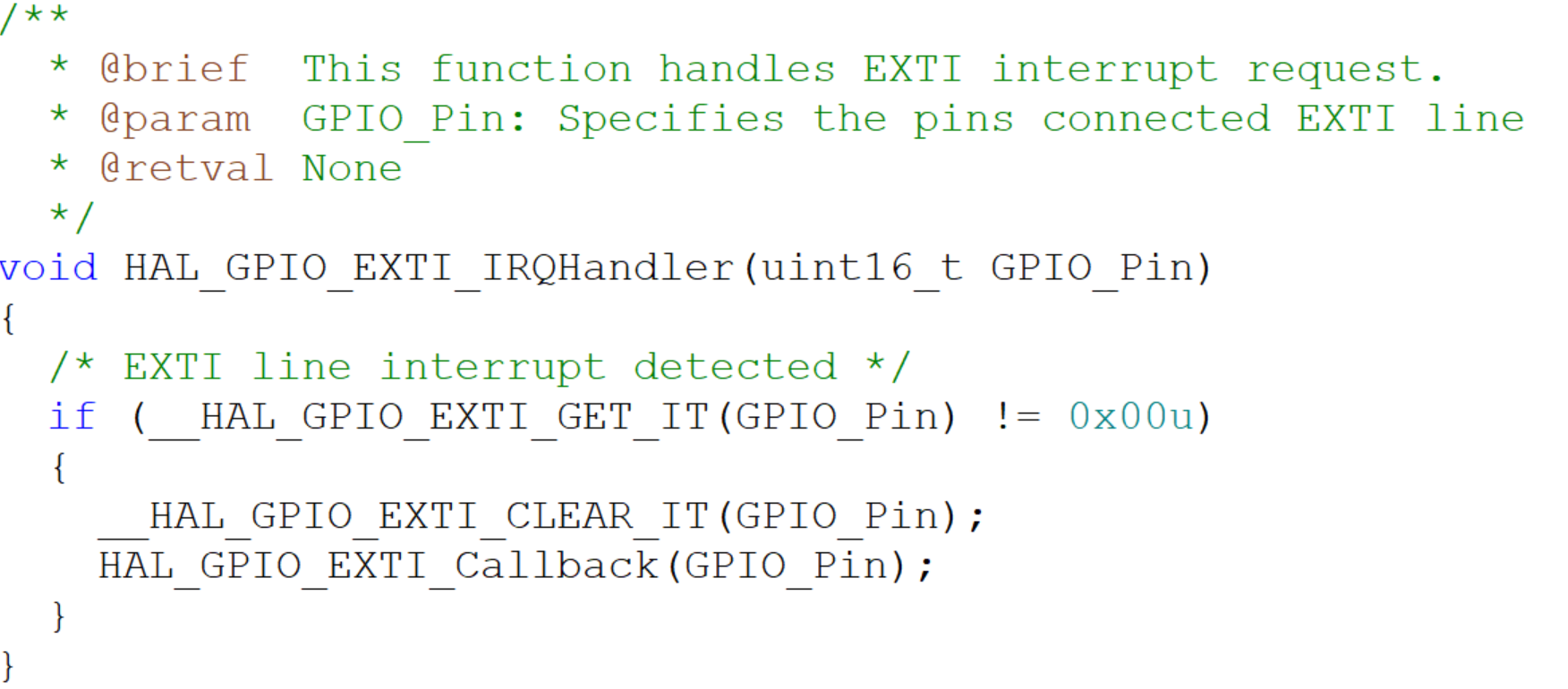
这个框图，感觉不知道看了多少遍了，感觉没有什么好说的了。

下面看一下在HAL库中这个是怎么来写程序吧



在it.c中如果触发了中断会进入到这个中断的句柄函数中，执行这个外部中断句柄的功能。

由于在CubeMx中配置好了所以进入到中断句柄函数中有下面的东西:

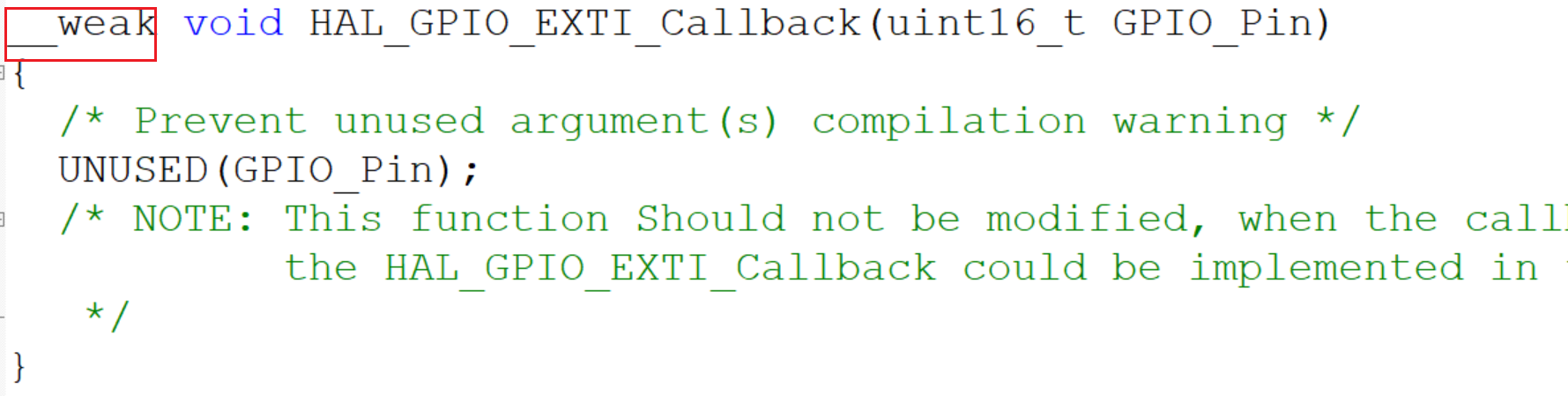


会自动地看一下，中断是不是由相应的GPIO口来触发的。

如果是那么就会清除中断标志位。如果中断标志位在的话，中断可能会一直产生，造成一些不好的影响。

然后会执行中断的回调函数。

这样看HAL库的代码架构是非常清晰的。



\_\_weak修辞符的意思是如果在其它地方定义了这个函数，那么自定义的那个函数会覆盖掉这个函数。

UNUSED（）的功能是如果有没有用到的变量，那么不会报错。强迫症的人很高兴。

在 C 语言里，\_\_weak 并非标准 C 语言的关键字，而是编译器（如 GCC）或者特定开发环境（像 ARM Cortex - M 系列单片机开发中使用的 Keil、IAR 等）提供的一种扩展修饰符。下面详细介绍其用途、特点及使用示例。

用途

\_\_weak 修饰符主要用于函数或变量声明，它允许用户在不修改原有库文件或者其他代码的情况下，对某些函数或变量进行重定义。在嵌入式开发中，这个特性非常实用，特别是在使用一些官方提供的库或者框架时，用户可能需要根据具体需求对某些默认函数进行修改。

特点

弱符号特性：被 \_\_weak 修饰的函数或变量属于弱符号。在链接过程中，如果存在强符号（即没有使用 \_\_weak 修饰的同名函数或变量），链接器会优先使用强符号；只有当不存在强符号时，才会使用弱符号。

默认实现：通常情况下，\_\_weak 修饰的函数会作为默认实现存在。当用户没有提供自定义的实现时，程序会使用这个默认实现；而当用户提供了自定义的实现后，程序就会使用自定义的实现。

void HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin)  
{  
 if(GPIO\_Pin==GPIO\_PIN\_12)  
 {  
 if(HAL\_GPIO\_ReadPin(GPIOA,GPIO\_PIN\_12)==GPIO\_PIN\_RESET)  
 {  
 flag=1;  
 }  
 if(HAL\_GPIO\_ReadPin(GPIOA,GPIO\_PIN\_12)==GPIO\_PIN\_SET)  
 {  
 flag=0;  
 }  
 }  
}

在回调函数中再加一层保险，再来判断一下是不是对应的GPIO口触发的，因为是不同的触发条件。

如果是下降沿触发：会检测到低电平，置一个标志位。

如果是上升沿触发：会检测到高电平，置一个标志位。

再在main.c中根据不同的标志位来处理不同的程序。实现不同的功能。

上面的好像不是加一层保险而是，10-15会触发同一个中断句柄所以才要判断一下。