# 嵌入式系统实验报告



|  |  |
| --- | --- |
| 实验名称： | STM32 I/O 编程—STM32 EXTI 中断 |
| 姓 名： | 王小龙 |
| 学 号： | 2020211502 |
| 学 院(系)： | 计算机学院 |
| 专 业： | 网络工程 |
| 指导教师： | 戴志涛、刘健培 |

2022年 12月 1 日

# 实验目的

学习STM32 中断的处理，有三种 I/O 操作技术，分别为轮询、中断和DMA操作，本次实验用中断的方式实现。

# 实验环境

* ST-Link 仿真器
* KeiluVision5 MDK集成开发软件
* PC机Window10 (64bit)

# 实验要求

1.按照实验过程的步骤，实现按键中断方式控制 LED 的功能。

2.完成步骤 3.2，将程序源码和程序输出截图贴在作业答卷里。

# 实验原理

**1.三种 I/O 操作技术——轮询、中断、DMA**

在 I/O 接口操作技术中，CPU 控制 I/O 接口的方法主要有：

一、直接程序控制：数据的输入输出是在 CPU 的执行程序的控制下完成的。

分两种：

（1）无条件传送：外设时刻准备就绪，无条件随时接受/传送 CPU 执行程

序发来的数据；

（2）程序查询方式：通过 CPU 执行程序查询外设状态，确定是否传输数据；

二、中断方式：CPU 根据接口是否发出中断请求来决定是否接受/传输数据；

中断处理方式的特点是 CPU 不再被动等待，而是可以执行其他程序，一旦

外设为数据交换准备就绪，可以向 CPU 提出服务请求，CPU 如果响应该请求，

便暂时停止当前程序的执行，转去执行与该请求对应的服务程序，完成后，再继

续执行原来被中断的程序。

三、直接存储器存取（DMA）：数据在内存和 I/O 设备中直接成块传送，不

需要经过 CPU 的控制干涉

2.STM32 的“中断”机制是比较复杂的。

首先，与“中断”相关的术语就有 exception, interrupt, event 三个。Cortex-m4

核中包含一个 NVIC 控制器，用于处理 exception。而 interrupt 是属于 exception之一种，其它 exception 类型包括 SysTick 等。interrupt 又叫作 IRQ。

STM32 之中、Cortex-m4 核之外的“中断”，即为 interrupt/IRQ。STM32 通

过 IRQ Channel 向 NVIC 请求处理 IRQ，而 NVIC 处理包括 IRQ 在内的各

种 exception，例如：优先级...等等。对于 IRQ，NVIC 将调用其“中断处理程

序” ISR。

有些 STM32 外围接口直接通过 IRQ Channel 与 NVIC 接口，而 GPIO

外部中断则要通过另一个控制器—EXTI—与 NVIC 接口。GPIO 与 EXTI 之间的接口称为 EXTI line；而 EXTI 与 NVIC 之间则为 IRQ Channel。GPIO pin 与

EXTI line 之间是 n:1 的关系，而 EXTI line 与 IRQ Channel 之间也是 n:1 的关系。基本上，PXn 对应 EXTI line n，这里 X=A, B, ... H，n=0, 1, 2 ... 15。例如，PX2（PA2, PB2 ...）都对应于 EXTI line 2。

EXTI line 与 IRQ Channel 之间的对应关系则稍微复杂，16 个 EXTI line 占

用 7 个 IRQ：

EXTI line 0 - 4 分别对应一个 IRQ，因此，共有 5 个 IRQ

EXTI line 5 - 9 共用一个 IRQ

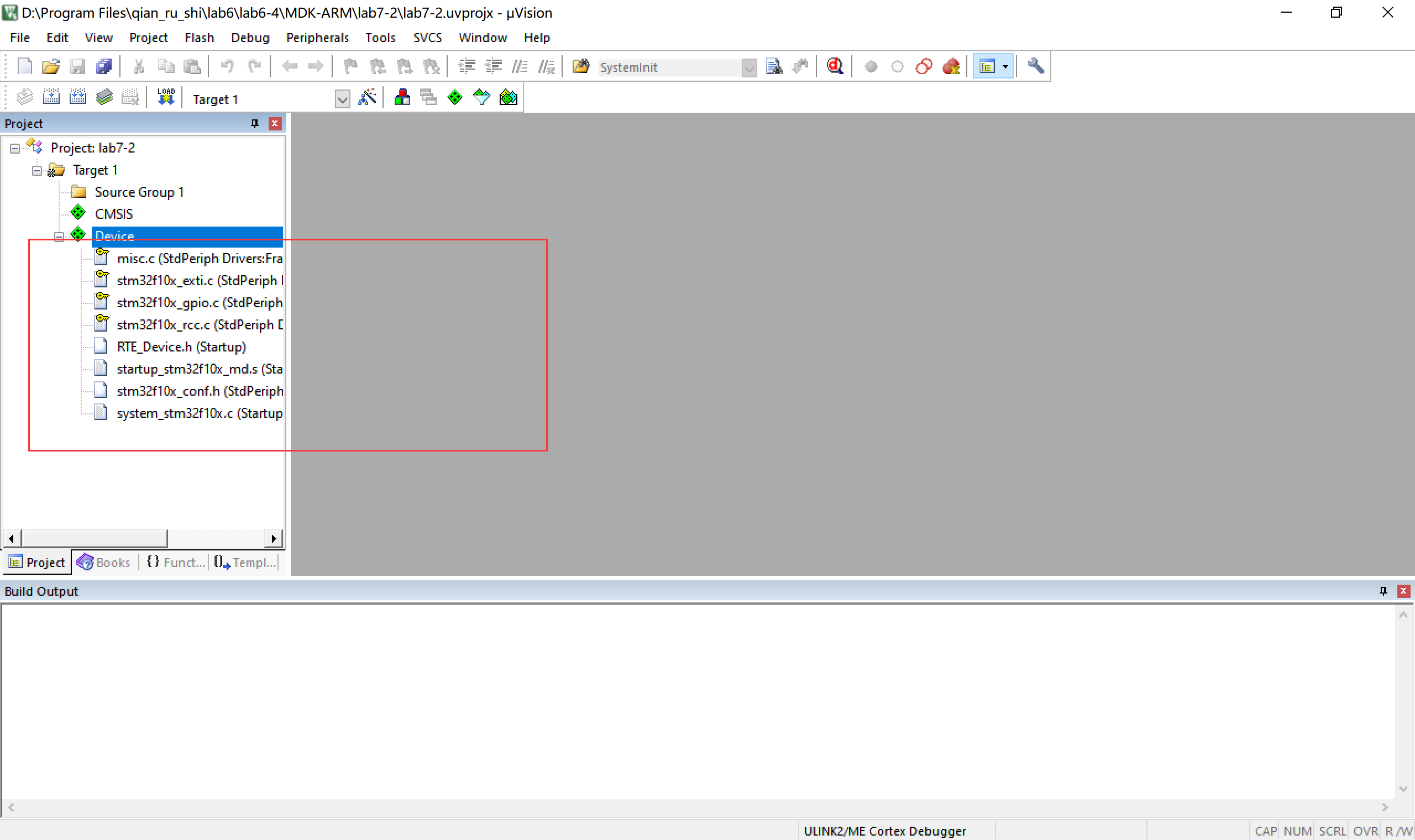
EXTI line 10 - 15 共用一个 IRQ

此外，EXTI line 上除了支持 interrupt 之外，还支持 event。event 被触发

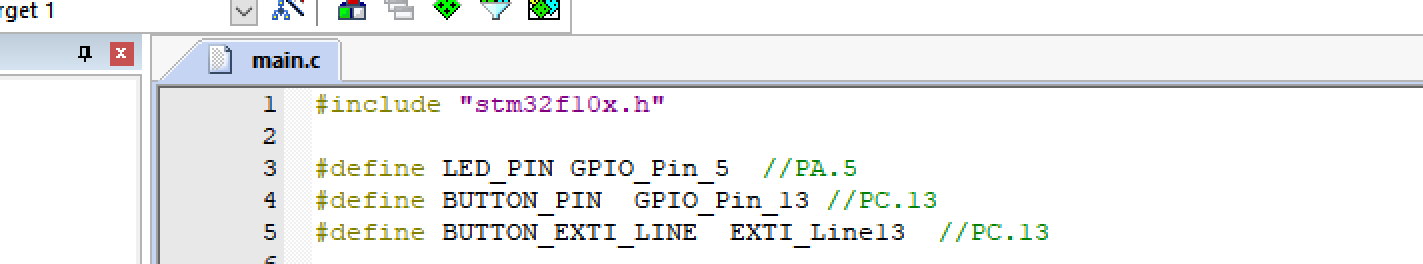
之后，并不传递给 NVIC 去处理（像 IRQ 那样），而是发送一个脉冲给电源管理模块，可以用来实现唤醒功能。

# 实验步骤

1.首先，创建 MDK 新工程 lab7-2，选择芯片 STM32F103RB。选择 Startup 启动代码和必须的 CMSIS::CORE 代码外，还将选择外设库 Framework、GPIO、RCC 和 EXTI 模块代码。因为我们将使用 GPIO、 RCC 和 EXTI 模块提供的 API 操作寄存器。



2.添加main.c，定义引脚的初始值：

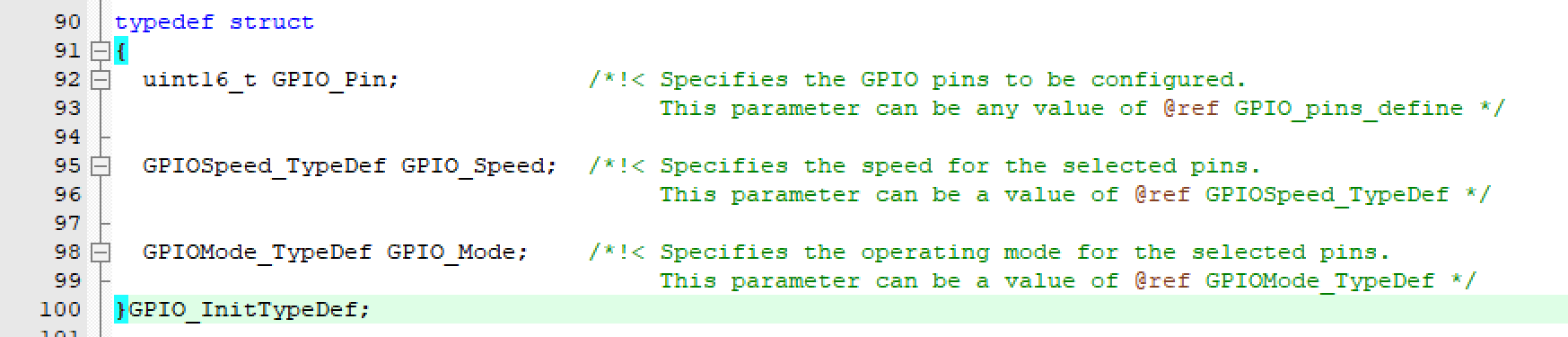


3.在Button\_Init()函数中实现GPIOC.13的初始化：

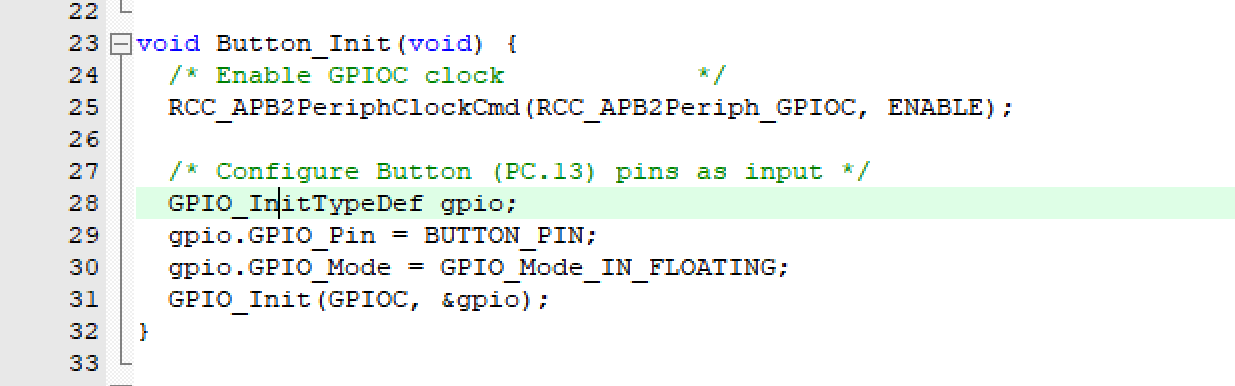
第一步，配置GPIOC.13的引脚，通过RCC\_APB2PeriphClockCmd函数初始化GPIOC的时钟，再用GPIO\_Init函数完成初始化。

通过查找GPIO\_Init函数的定义，可以看到其功能是完成GPIO的初始化，需要GPIOx, GPIO\_InitStruct两个参数。

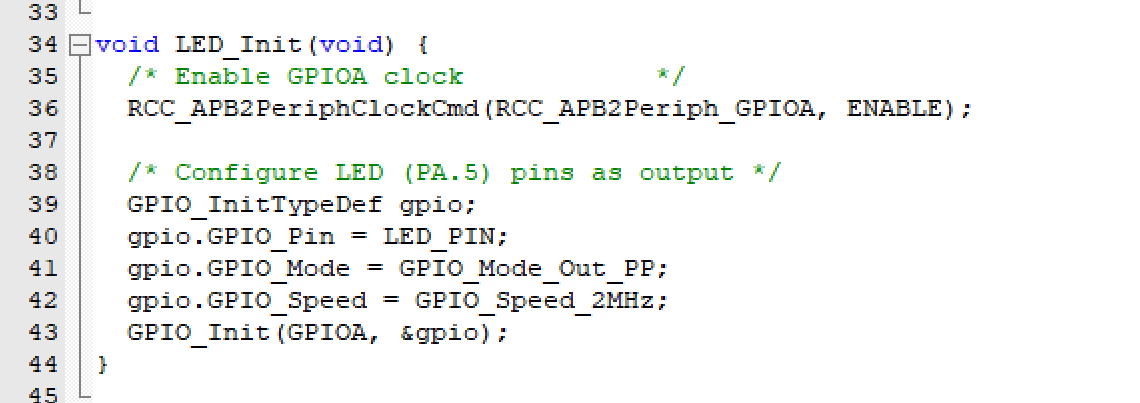
查看参数GPIO\_InitStruct的类型GPIO\_InitTypeDef定义：



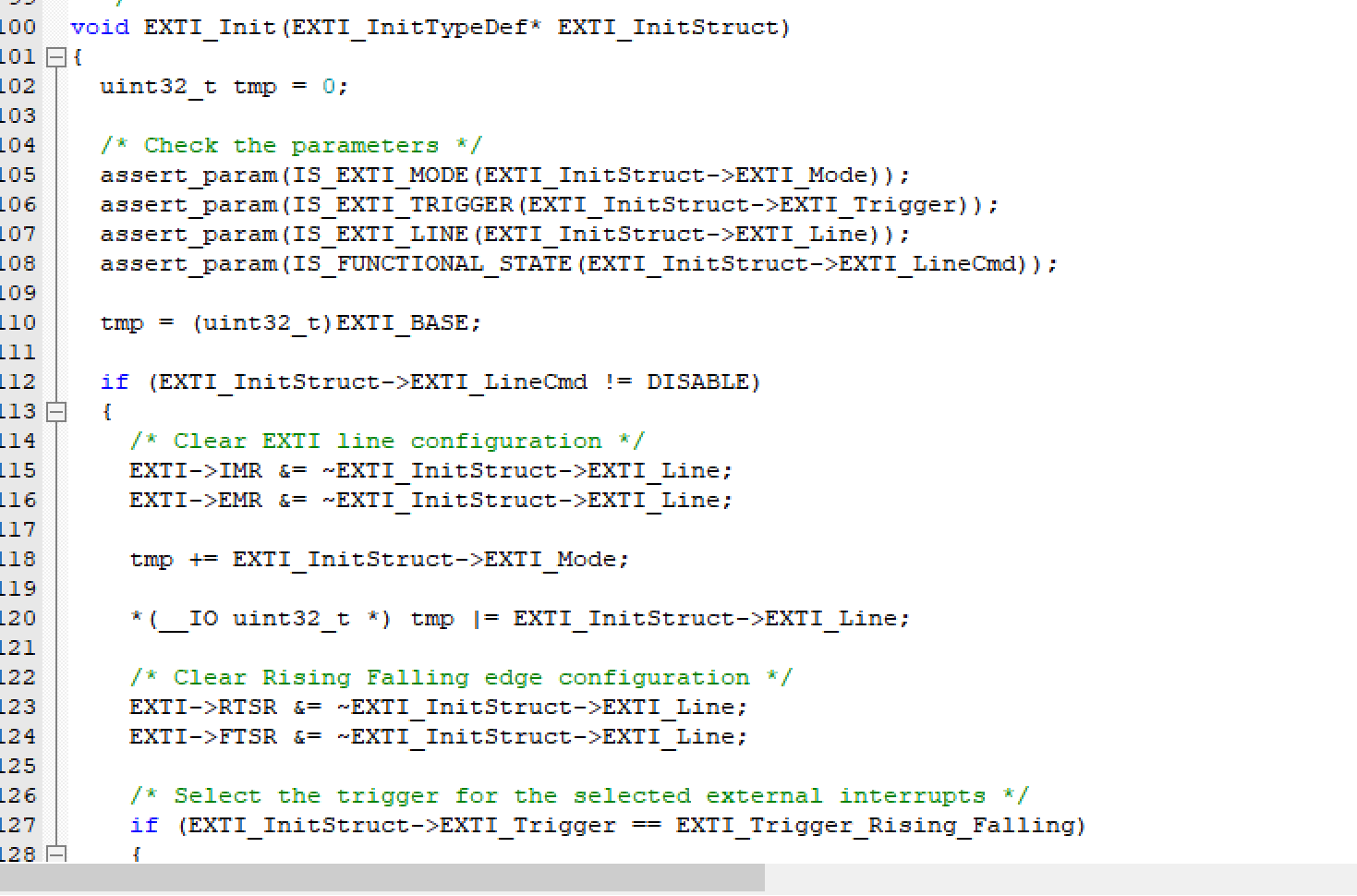
将其置为我们需要的模式：



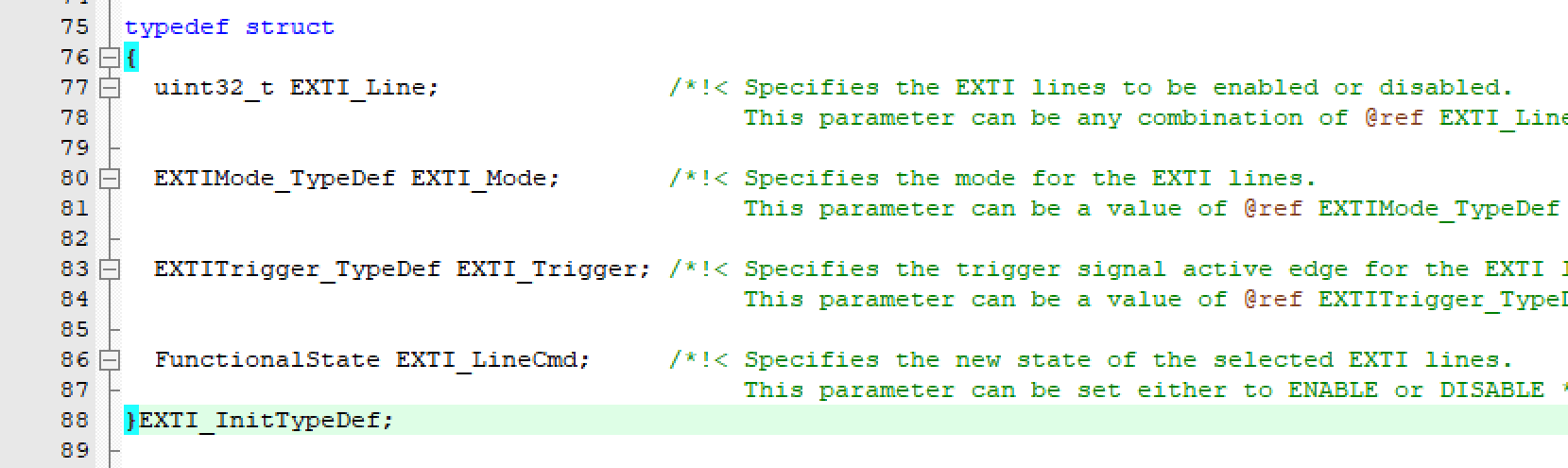
4.接下来初始化GPIOA.5端口，和上面一样：



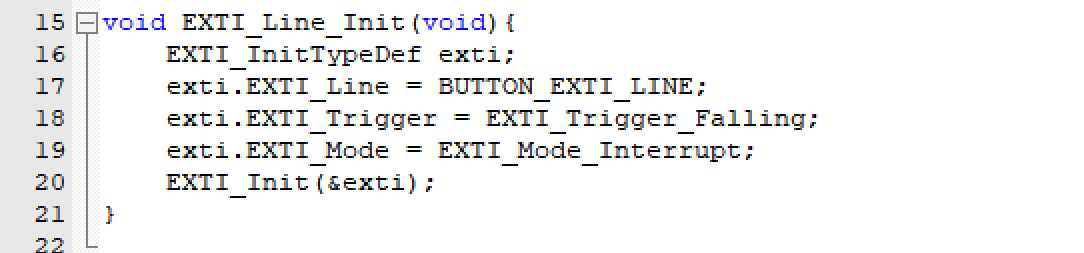
5.阅读stm32f10x\_exti.c，发现EXTI\_Init()函数可以根据传入的EXTI\_InitStruct配置EXTI。



查看EXTI\_InitStruct变量的类型定义EXTI\_InitTypeDef：

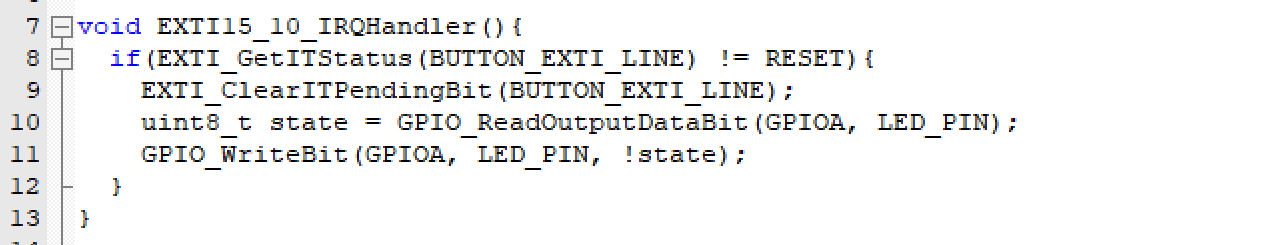


根据该结构，写出如下代码：

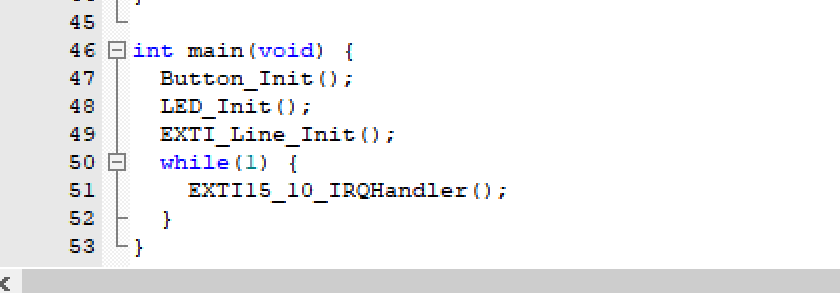


6.通过GPIO\_ReadOutputDataBit函数读取GPIO口的状态，再使用GPIO\_WriteBit写回返回的值，写回的状态为原来状态的相反状态以实现LED灯的跳变。

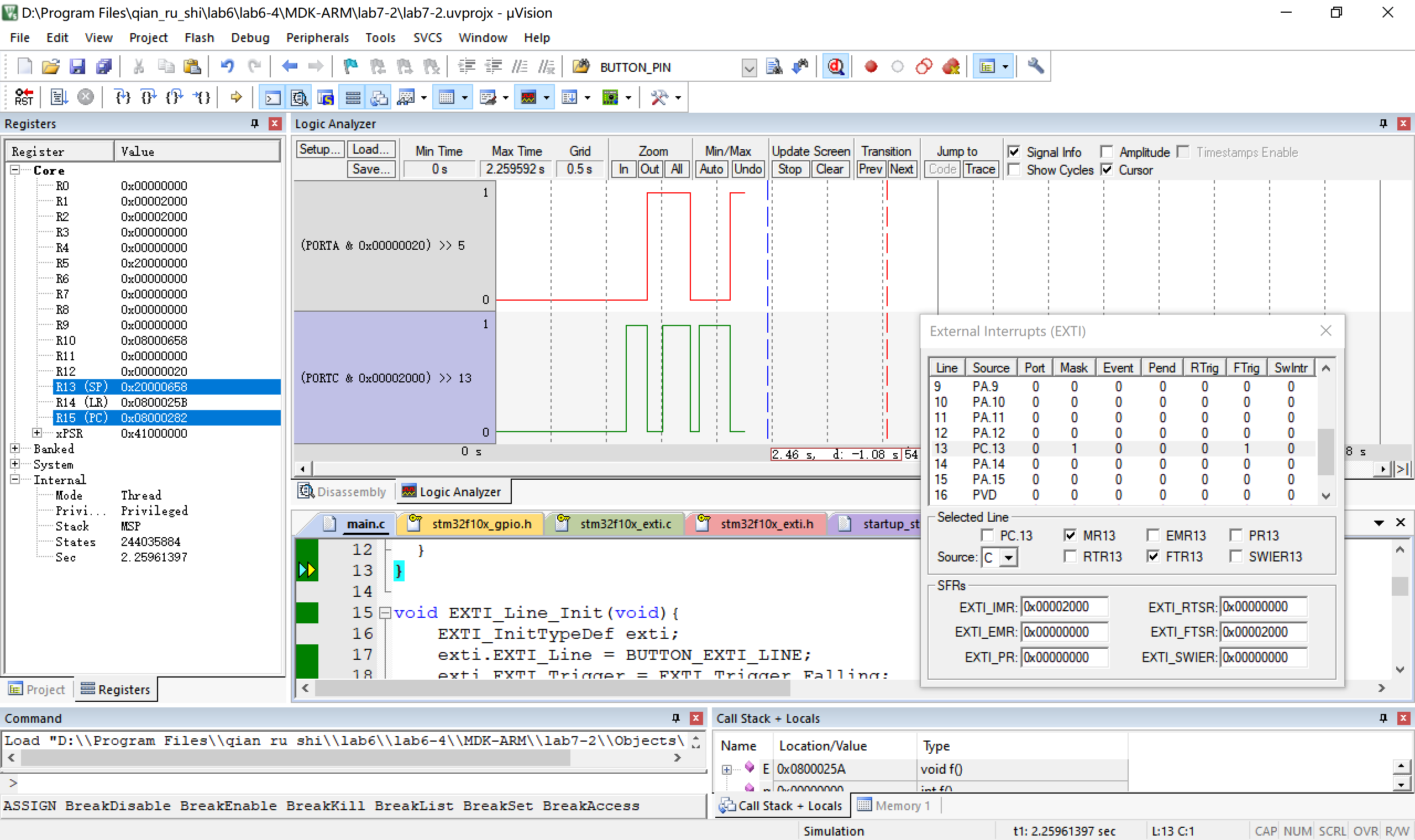
则中断函数如下：



7.编写main函数：

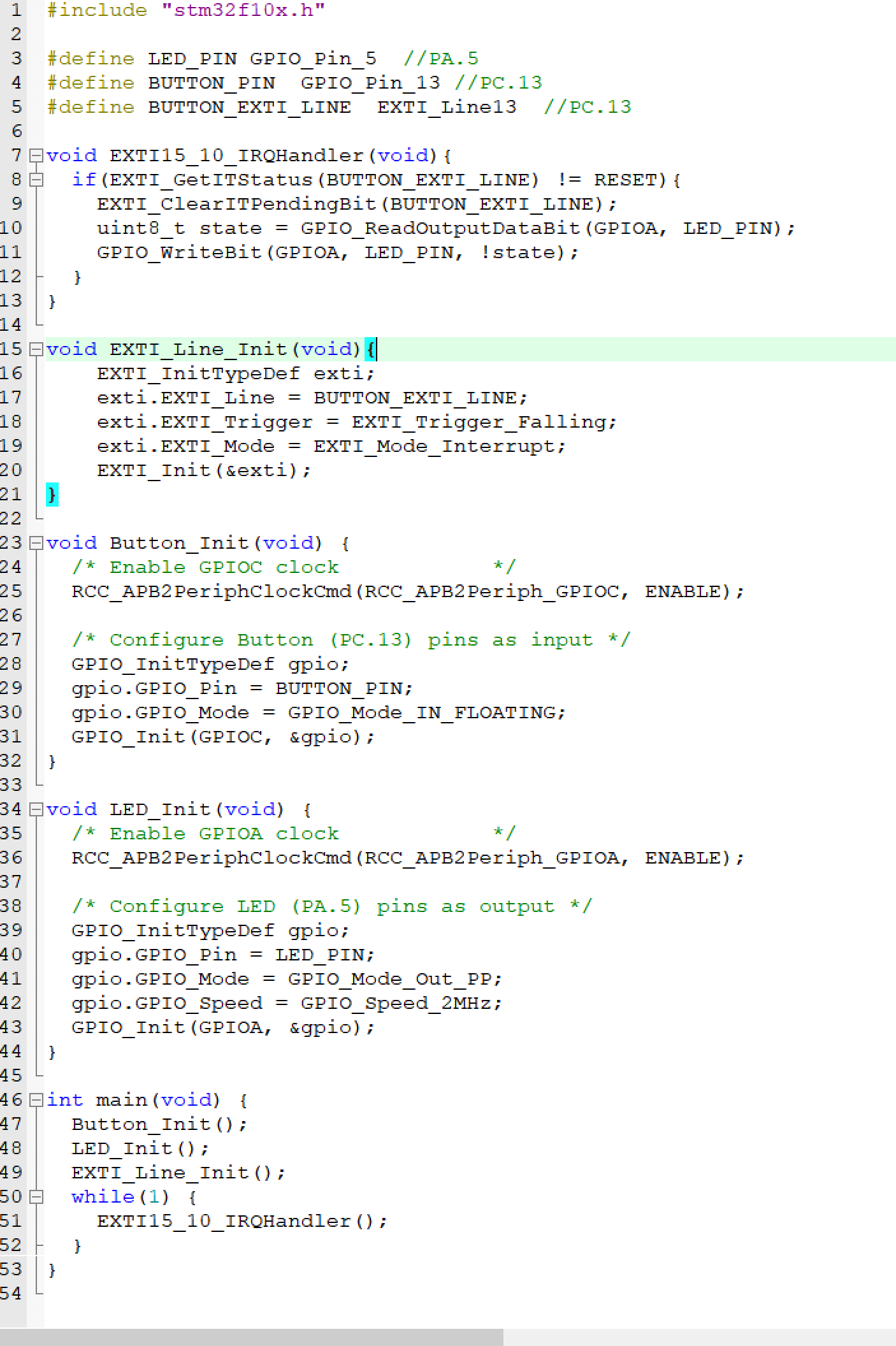


8.编译运行，可以看到PORTA的输出状态在PORTC的状态为下降沿时会发生跳变：



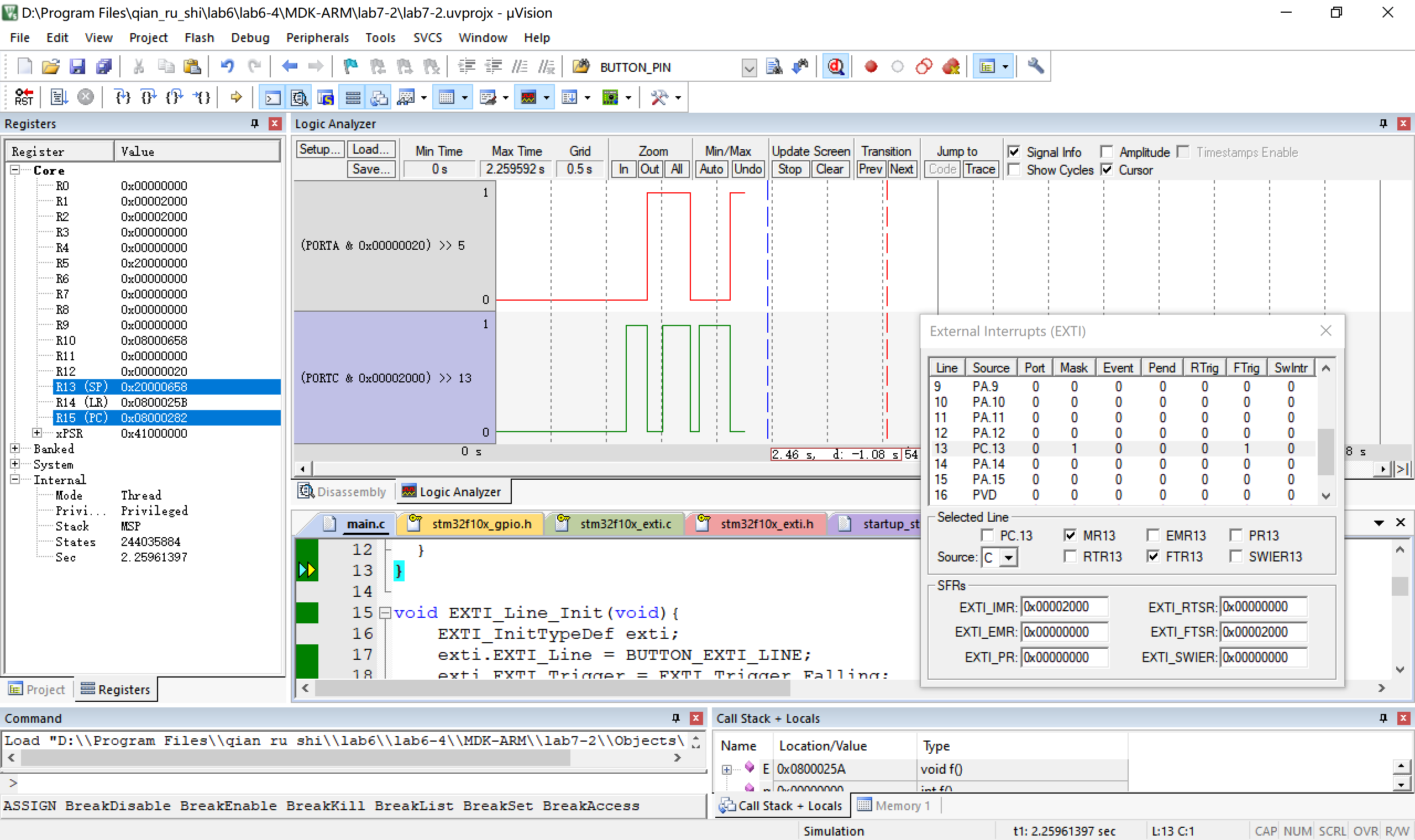
# 实验方案与实现

## 源代码



# 实验结果与分析

编译运行，可以看到PORTA的输出状态在PORTC的状态为下降沿时会发生跳变，结果正确。



# 实验总结

通过本次实验，学习了中断的有关知识，本次实验也是作业的最后一个实验，相比于以前的生疏，现在对MDK软件的使用也更加熟练，总之，在这些作业中，收获颇多。