# 嵌入式系统实验报告



|  |  |
| --- | --- |
| 实验名称： | STM32 I/O 编程—操作 STM32 外设  的 4 种方式 |
| 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

验在 MDK 环境操作 STM32 的 4 种方式。观察和分析其不同的特点。

# 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

将 LED 引脚改为 PA7，按键引脚改为 PC3，LED 与按键状态改为反相输出。重做此 4 种方式，将程序源码和程序输出截图贴在作业答卷里。

# 实验原理

1.**程序操作 I/O 接口的基本原理——操作 I/O 接口寄存器**

计算机输入输出单元（简称计算机 I/O）通过 I/O 接口单元（也称为接口/interface 或控制器/controller）与计算机外围设备协作交换信息。

为了实现这种协作，处理器执行输入/输出指令，对 I/O 接口单元中选定的存储设备（通常是寄存器）执行读/写操作。另一方面，I/O 接口与外围设备协作，传输数据与交换控制信号。

I/O 接口单元通过数据、地址和控制总线连接到处理器。

每个 I/O 设备都会有一个专用的 I/O 地址，用来处理自己的输入输出信息。

每个 I/O 设备在其控制器芯片上都会有物理的寄存器（注意这里的寄存器不

是指的CPU内部的寄存器R1-R15等，而是指的硬件芯片上的存储单元），在ARM体系下，这些存储单元与内存进行统一编址（与此相对应的是 I/O 与内存地址空间分离的独立编址法），可以被 CPU 通过访存指令，像访问内存一样（使用指针）去访问，这些寄存器通常分为 3 种类型：命令寄存器、状态寄存器、数据寄存器。

程序控制 I/O 设备的办法通常是：程序通过 STR 指令向命令寄存器写入合适的内容，以完成对硬件进行配置的操作或者要求硬件进行某种物理操作。到此

为止，软件就完成了所有它该做的事情，之后硬件会自动完成相应操作，在硬件

完成操作后，程序又可以通过 LDR 指令从数据寄存器中获得想要的数据，或者

从状态寄存器中获知硬件的状态。注意，处理器与 I/O 设备是物理独立，二者的运行是并行的。

可见，程序控制硬件，简单地说，其实就是程序对 I/O 接口单元的寄存器进

行读写操作，命令设备完成操作，获取设备状态和数据，仅此而已。这里的关键

是：某个 I/O 接口寄存器的内存地址是多少？为使设备执行某个操作，应当向哪个寄存器写入什么值？如何有效的与硬件进行交互？如何有效的与应用程序交互？这些都是开发者需要解决的问题，而这些问题的解决，关键在于开发者能：

(1)理解要控制的硬件的运作机制。

(2)能熟练查阅硬件的手册（手册中会指明寄存器的内存地址以及寄存器各

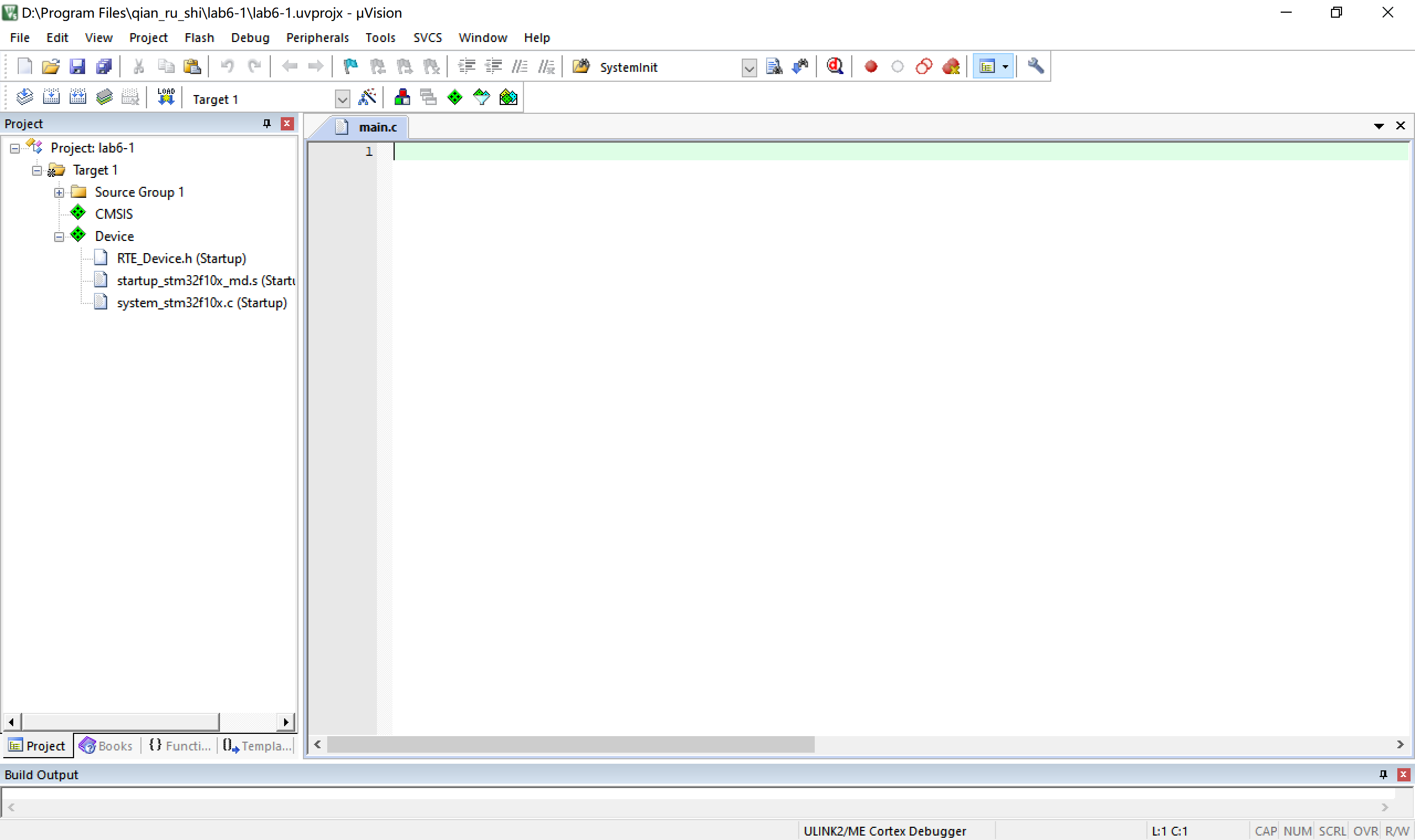
种取值的含义）。

(3)能看懂硬件的连线原理图。

# 实验步骤

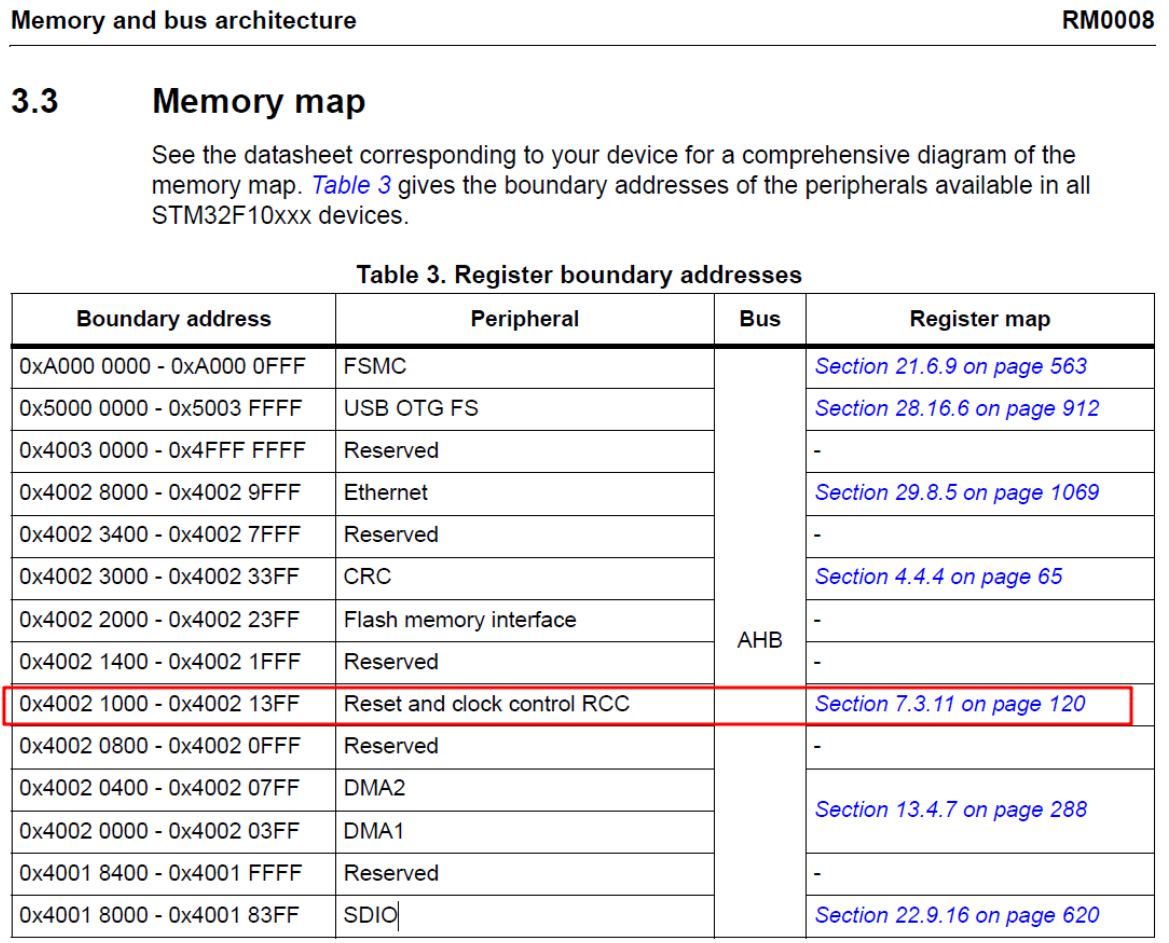
**(1)不使用 STM32 标准外设库+直接操作寄存器**

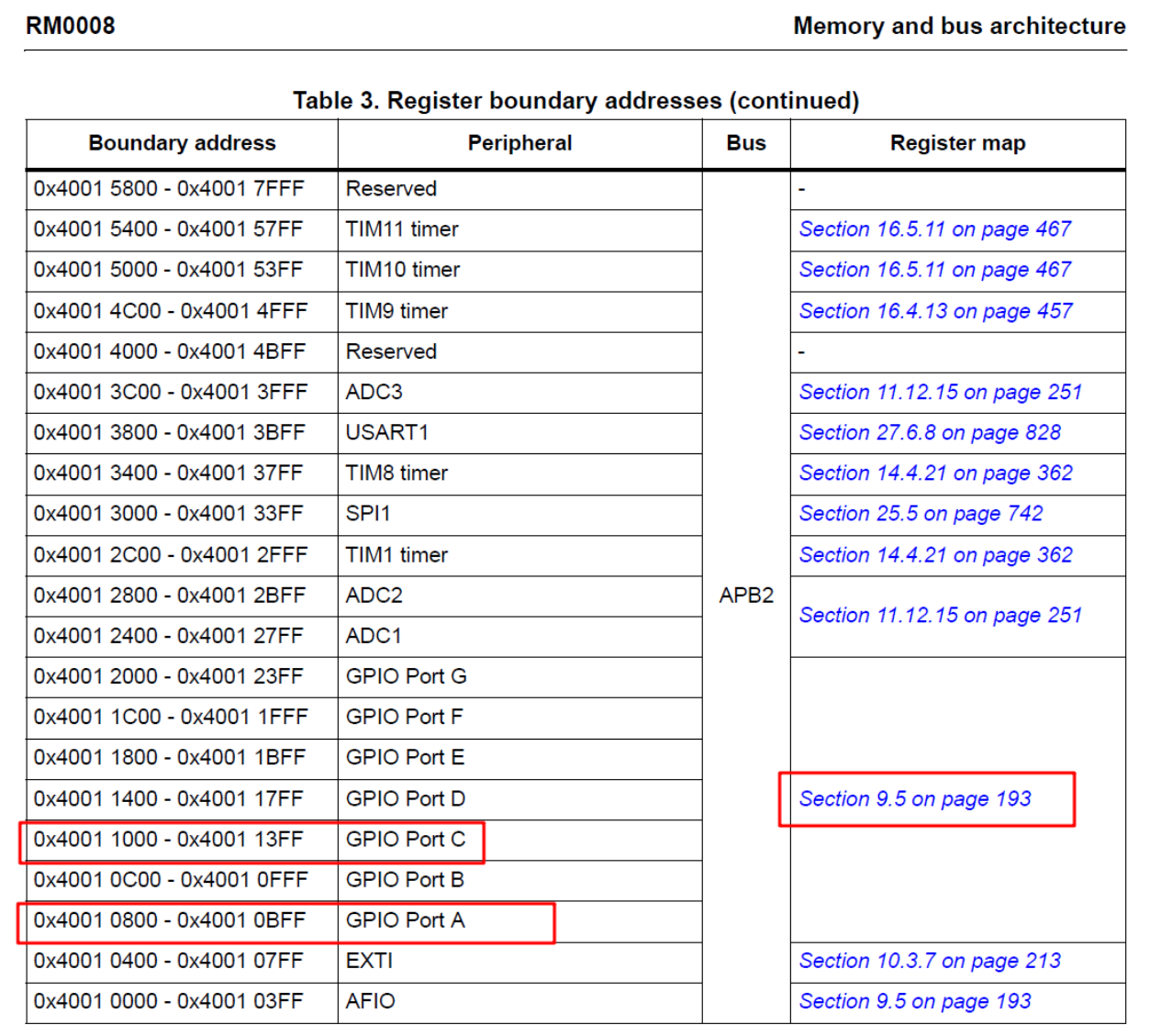
1.首先，创建 MDK 新工程 lab6-1，选择芯片 STM32F103RB。 并添加main.c



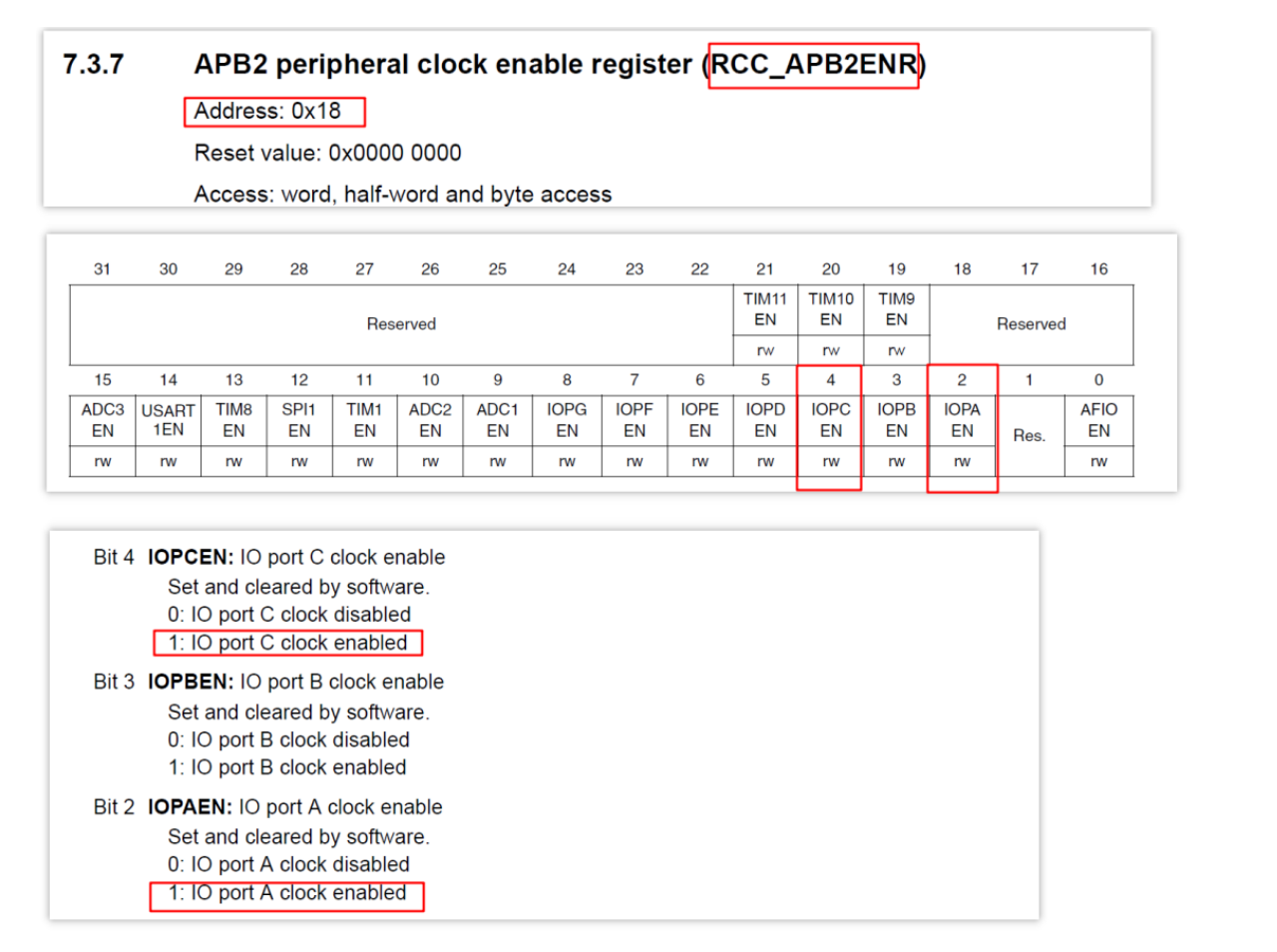
2.查找控制 GPIO 时钟的 RCC 模块寄存器组基地址，与 GPIOA、GPIOC

的寄存器组基地址。

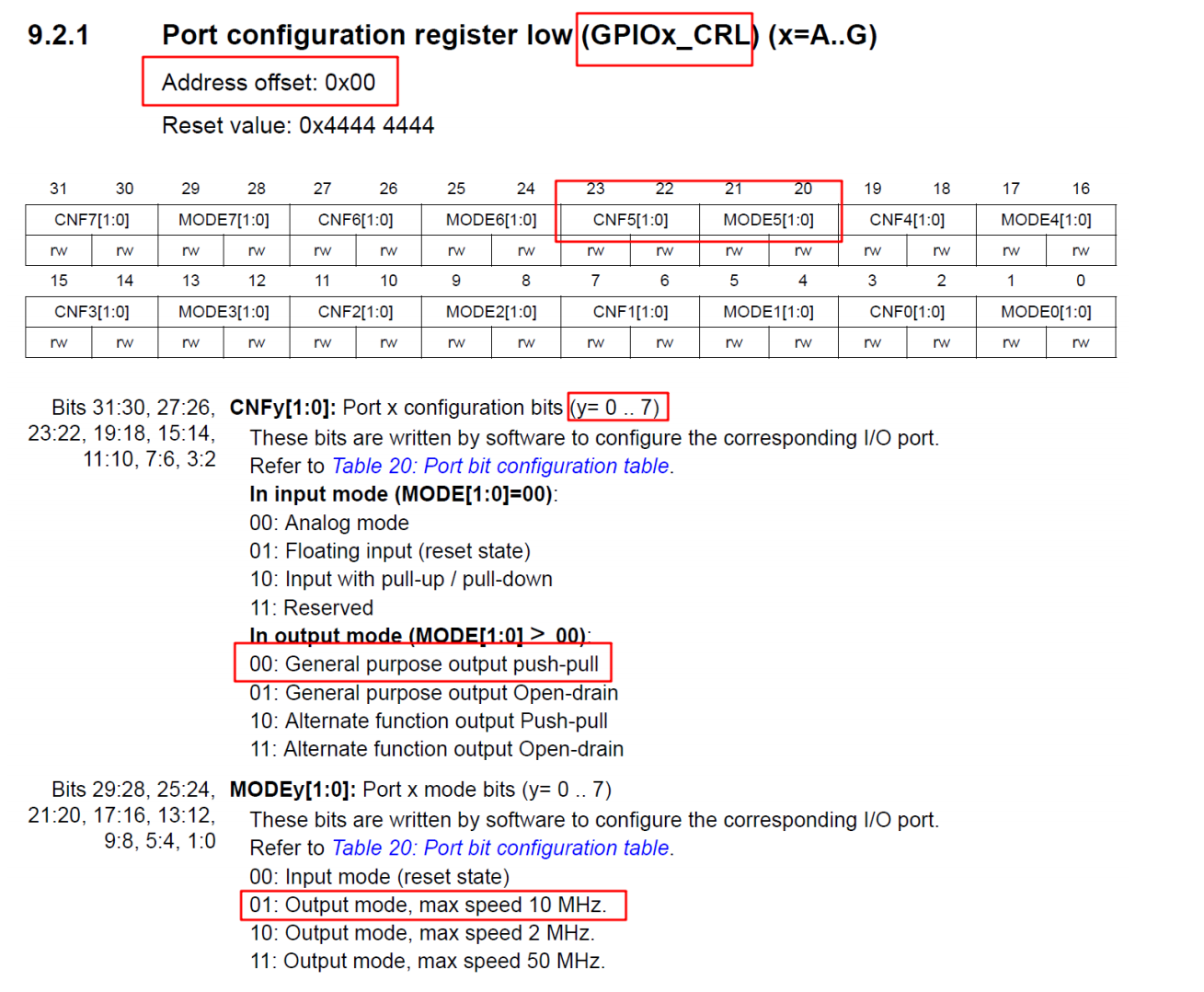




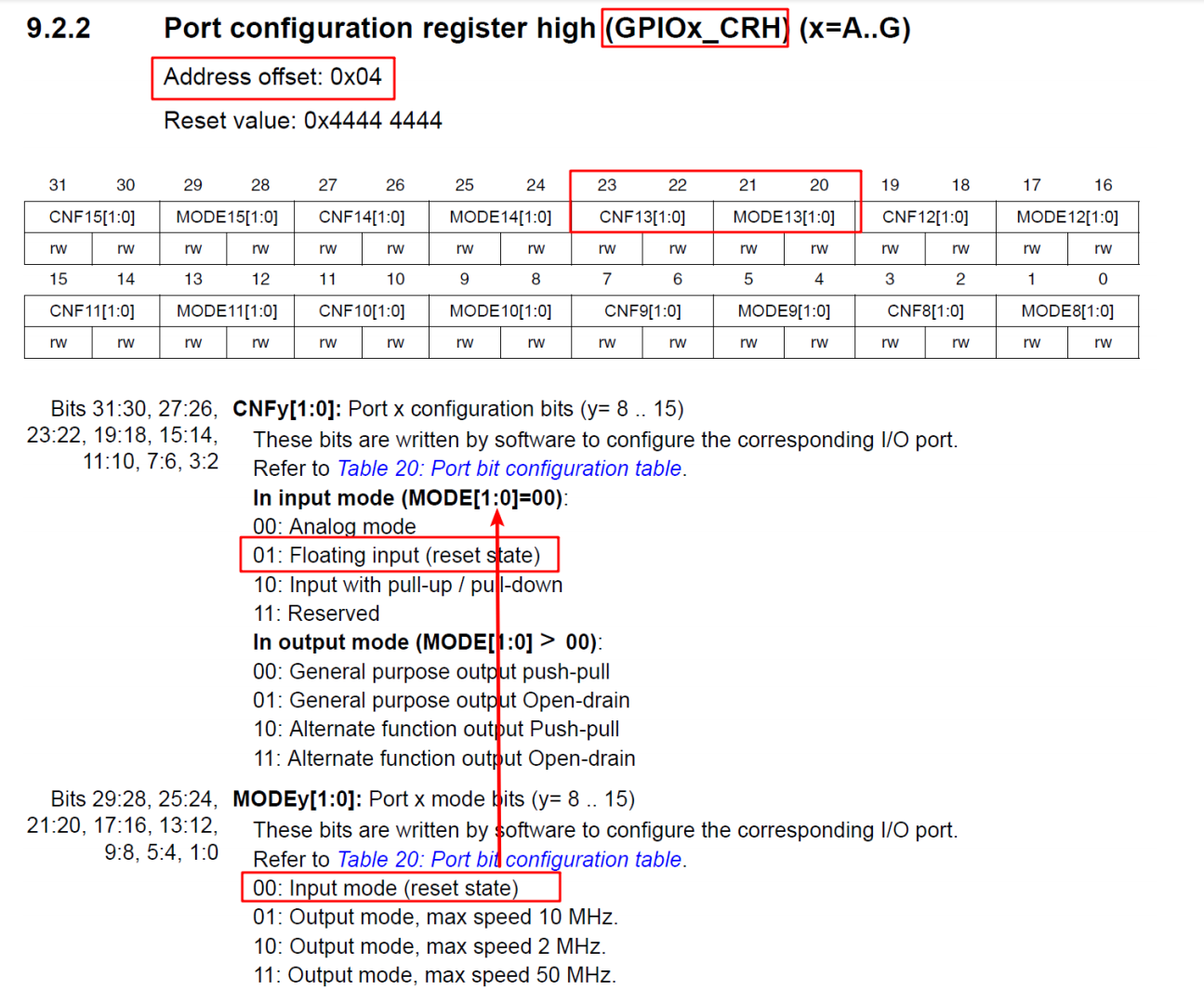
3.找到控制 APB2 上 GPIOA、GPIOC 的寄存器、偏移地址以及控制 bit。



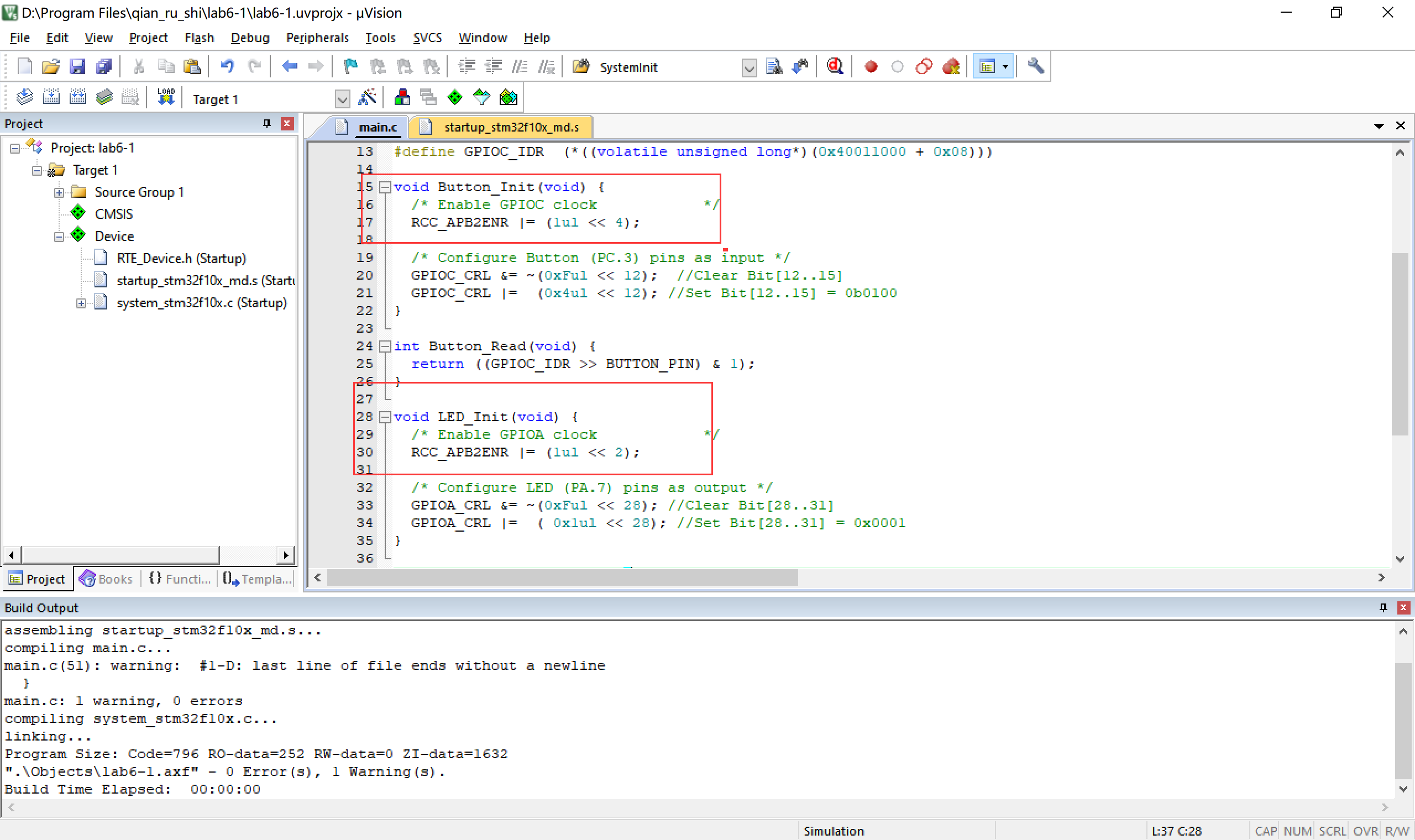
4.找到设置 GPIOA5（LED）模式的寄存器、偏移地址以及控制 bit。



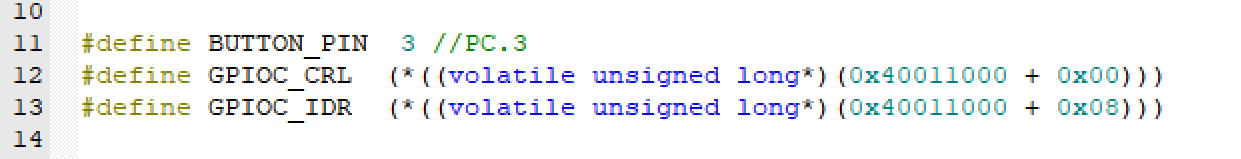
5.找到设置 GPIOC13（按键）模式的寄存器、偏移地址以及控制 bit。



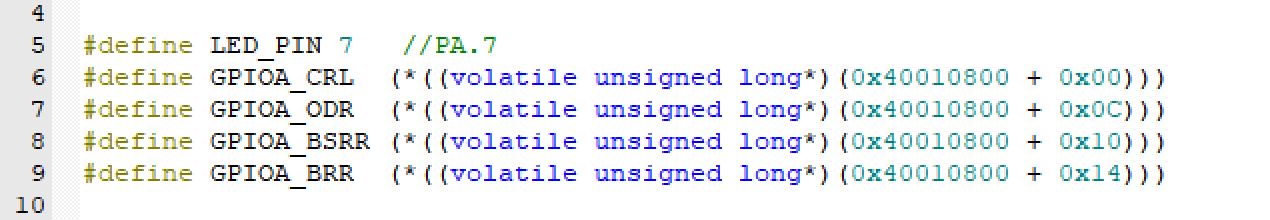
6.在APB2起始位置后2和4个bit 的位置，用位操作激活GPIOA和GPIOC端口。



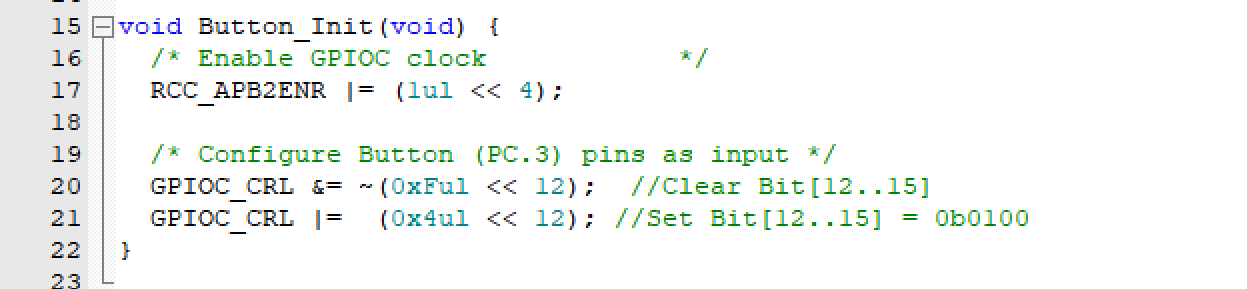
7.实验中需要将GPIOC.13换成GPIOC.3，如下操作：

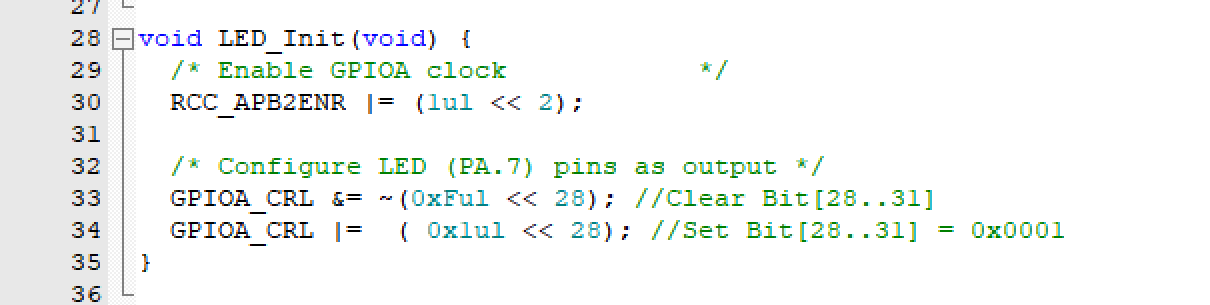


GPIOA各寄存器的位置：



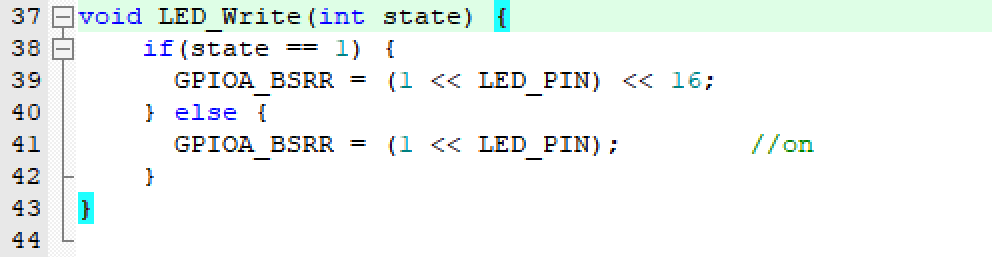
8.在Button\_Init中，初始化GPIOC.3，位置为12~15位，先清零再置成0b0100如下：



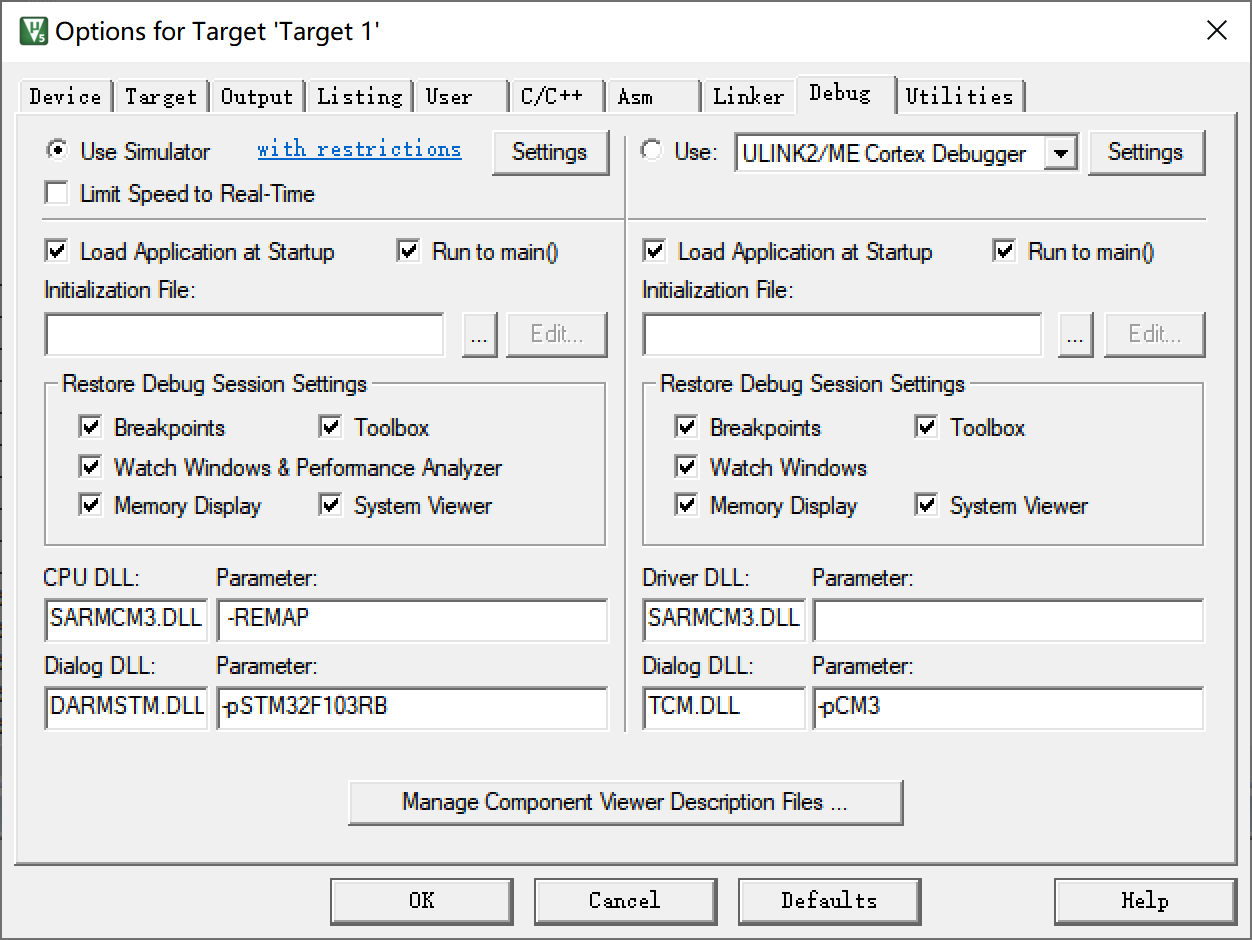
9.在LED\_Init中，初始化GPIOA.7，对应的位置为28~31位，先清零再置成0b0001，

10.实现LED 与按键状态改为反相输出：

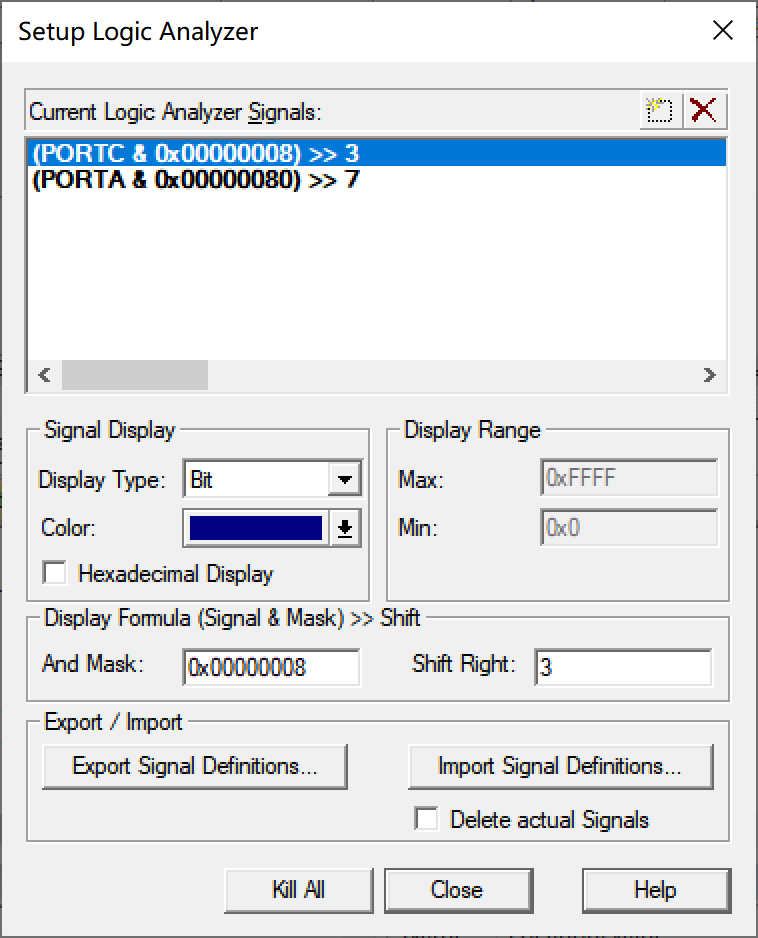
反向处理一下状态即可，state置1：



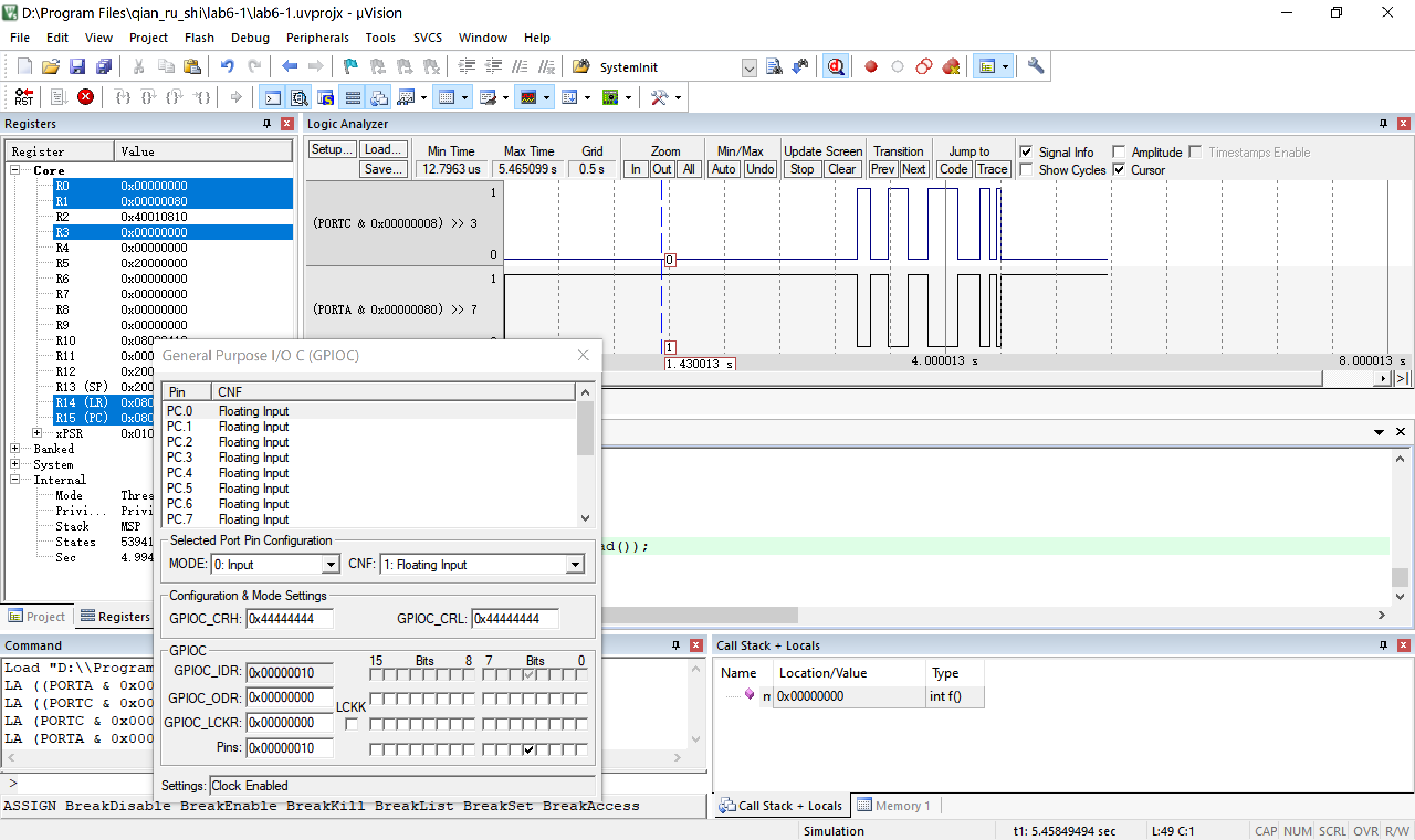
7.编译成功后，设置 debug 属性为使用模拟器



8.启动调试。然后打开 logic analyzer，增加 PORTA.7 与 PORTC.3 2 个观察点:

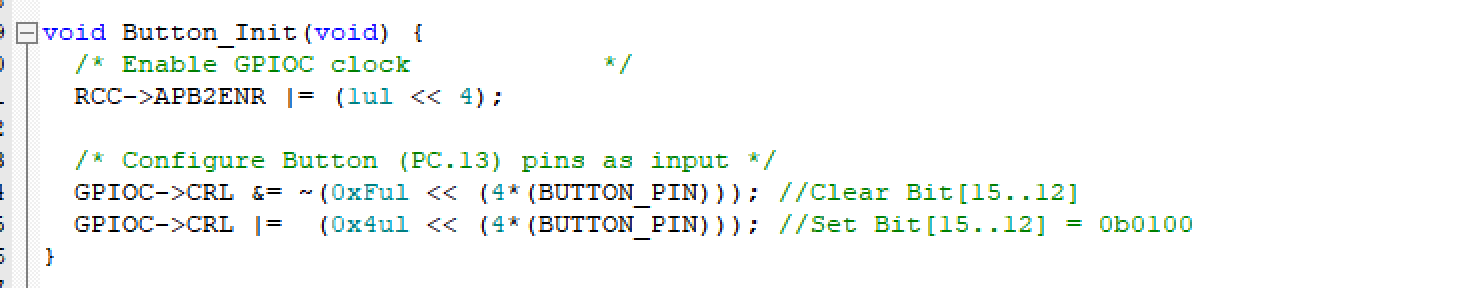


9.运行之后，可以看到实现了LED 与按键状态改为反相输出：

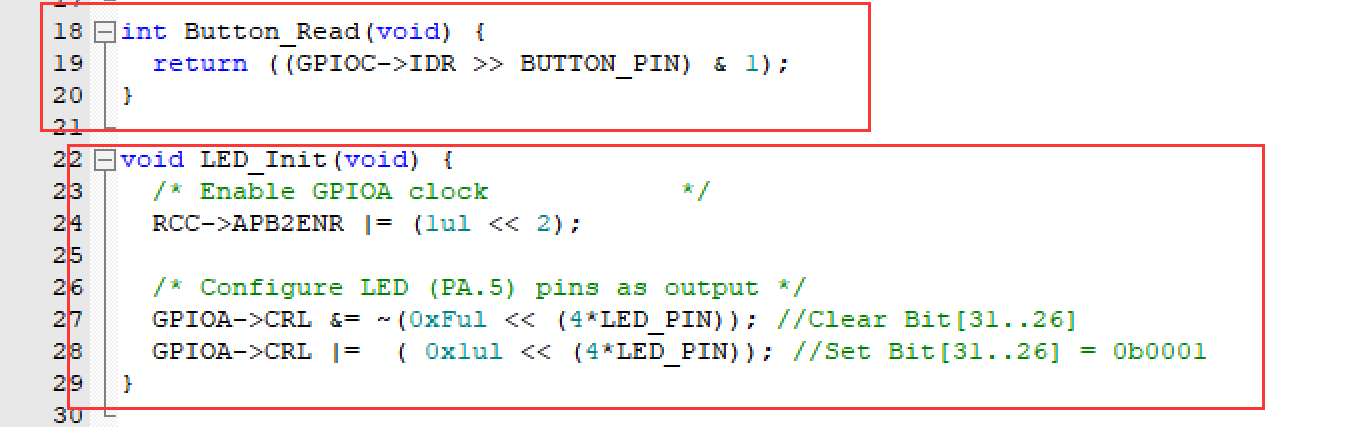


（2）**使用 STM32 标准外设库+直接操作寄存器**

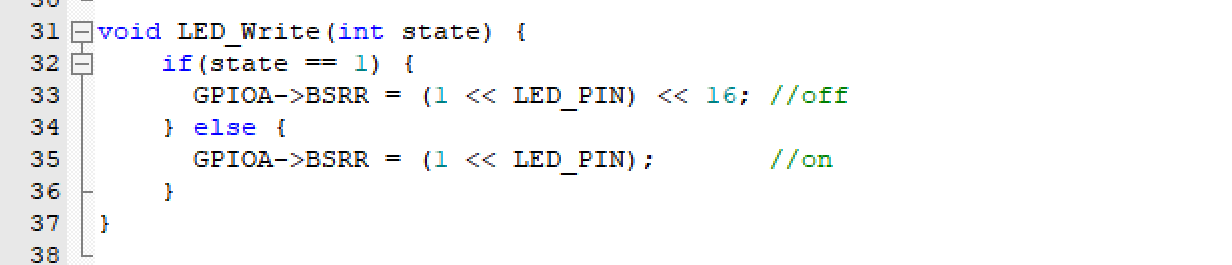
1.在Button\_Init函数中，把CRH换成CRL，以及BUTTON\_PIN – 8改成BUTTON\_PIN。



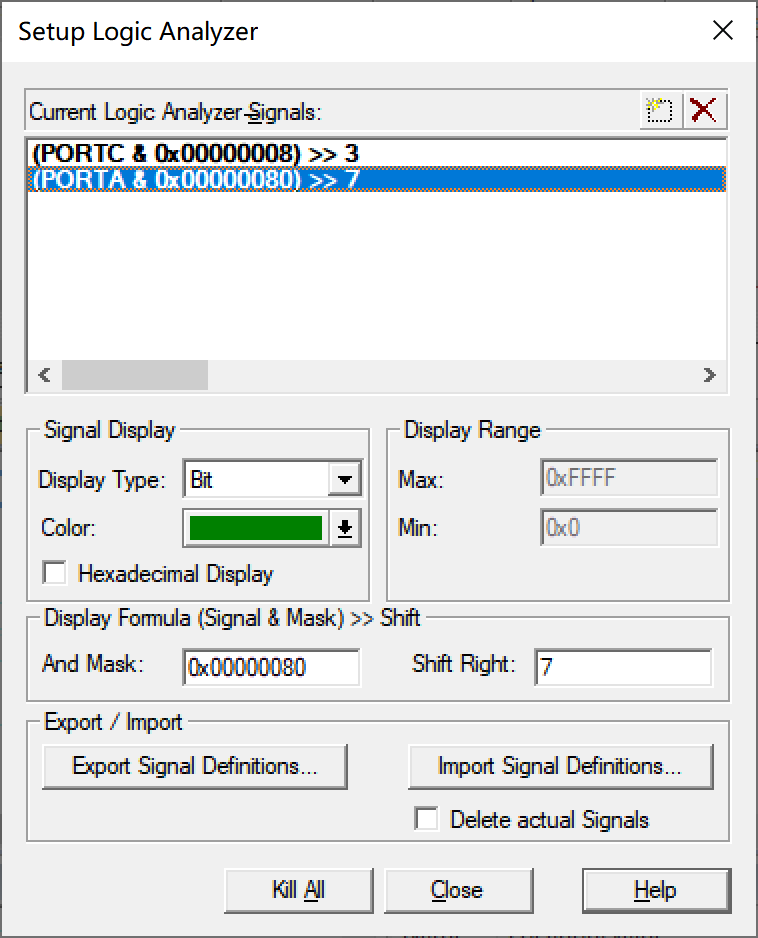
2.初始化



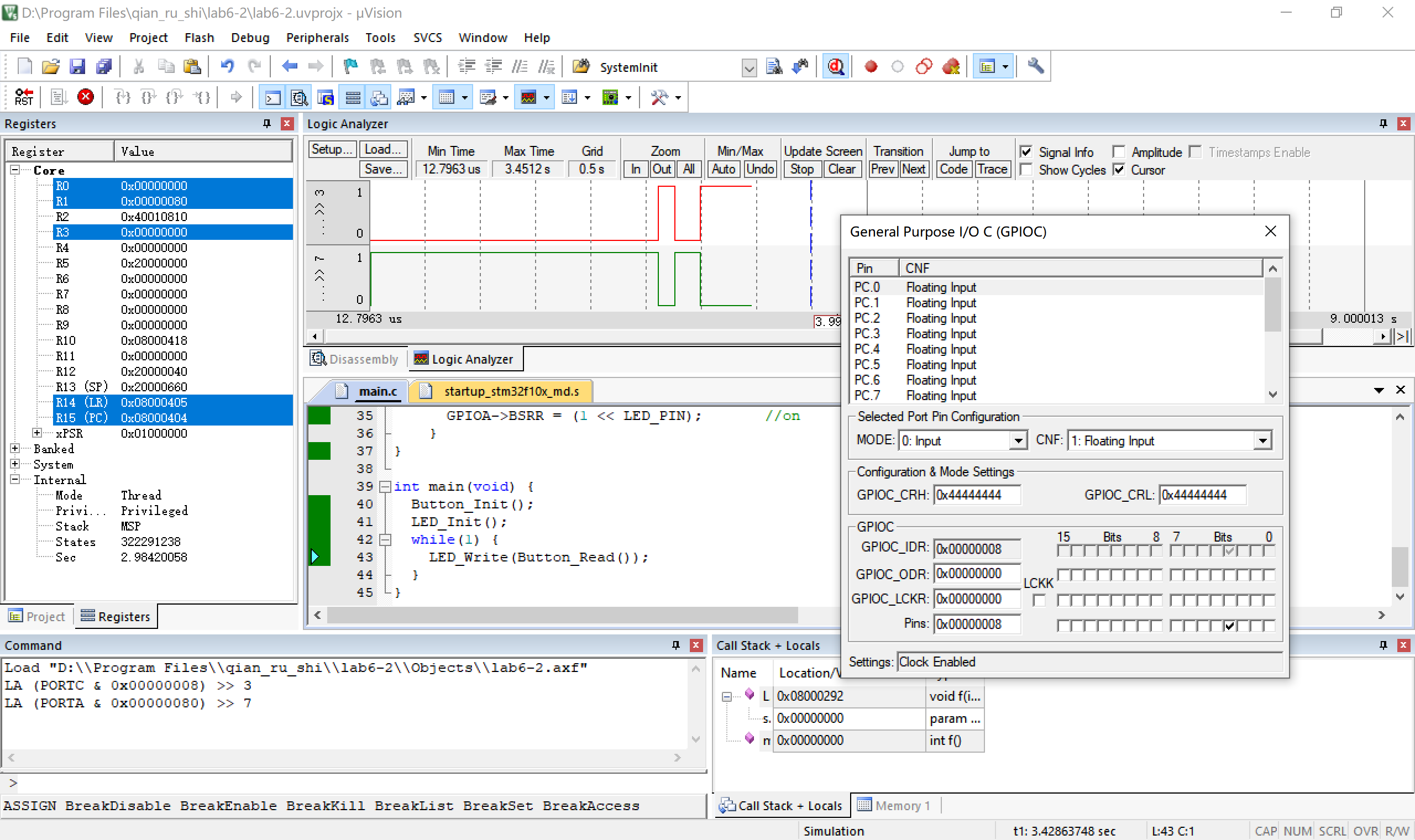
3.实现反向输出：



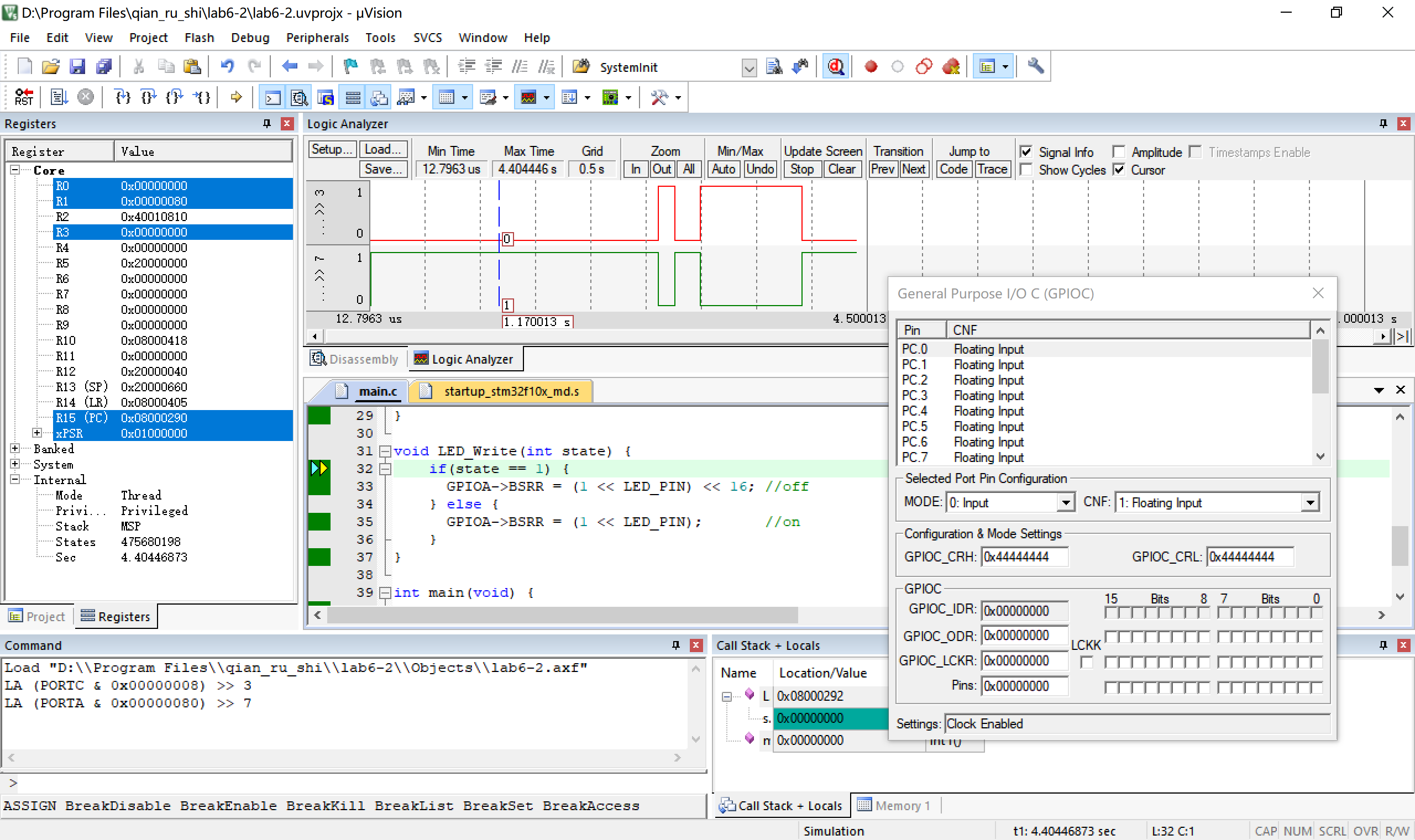
4.编译调试，添加查看：



5.运行：



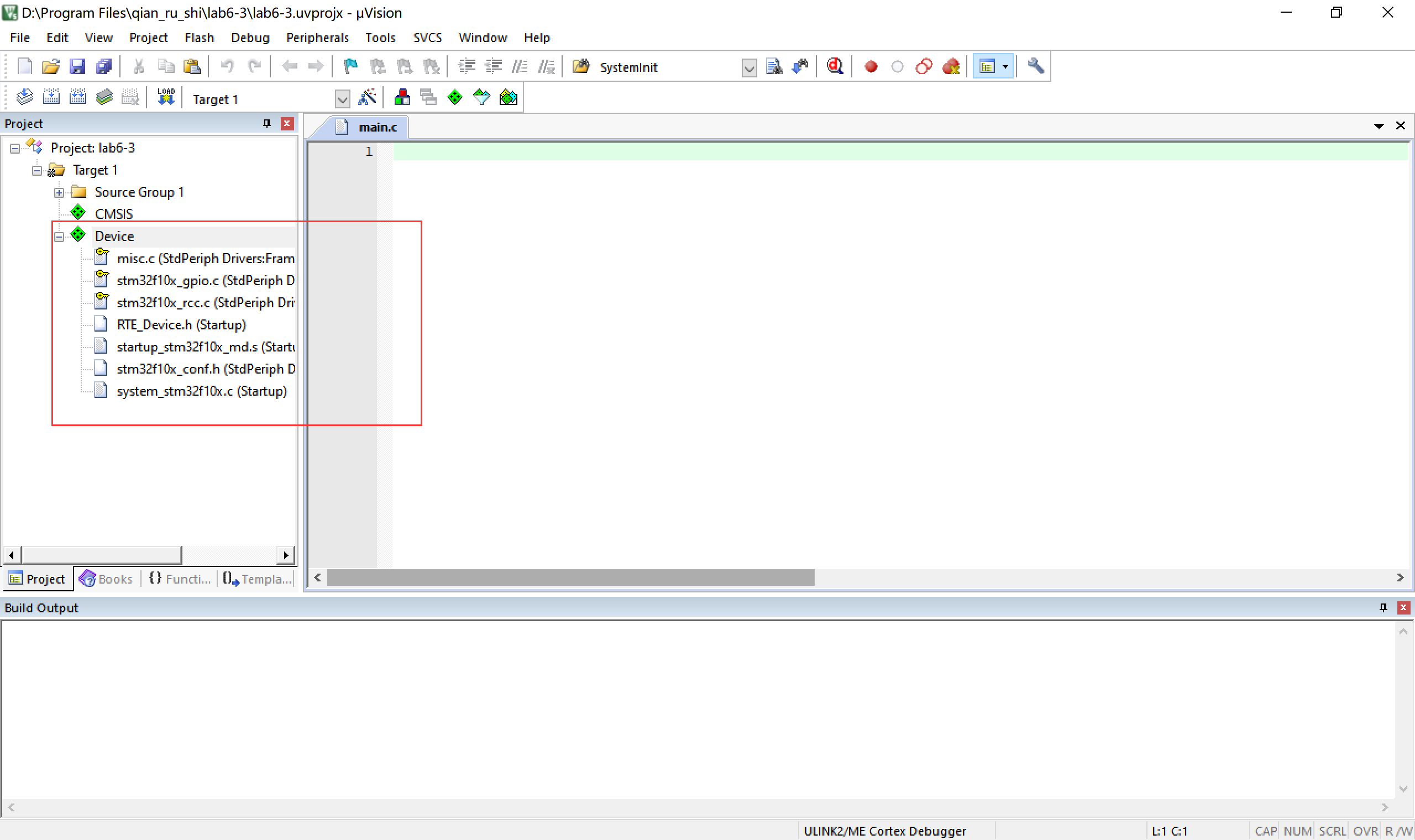
按键是由pins3操作，可以看到实现了反向输出：



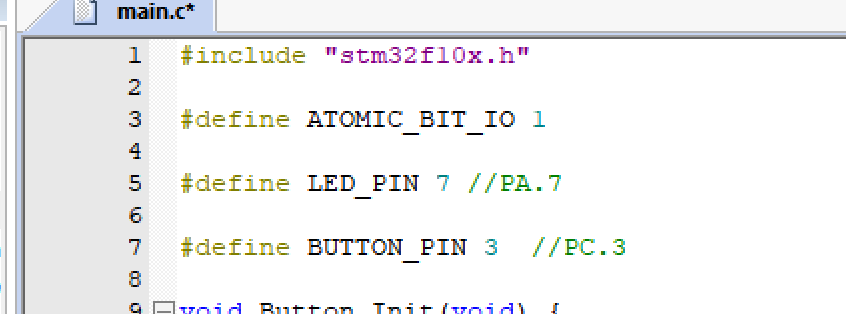
（3）**使用 STM32 标准外设库+API 操作寄存器**

1.创建 MDK 新工程 lab6-3，选择芯片 STM32F103RB。

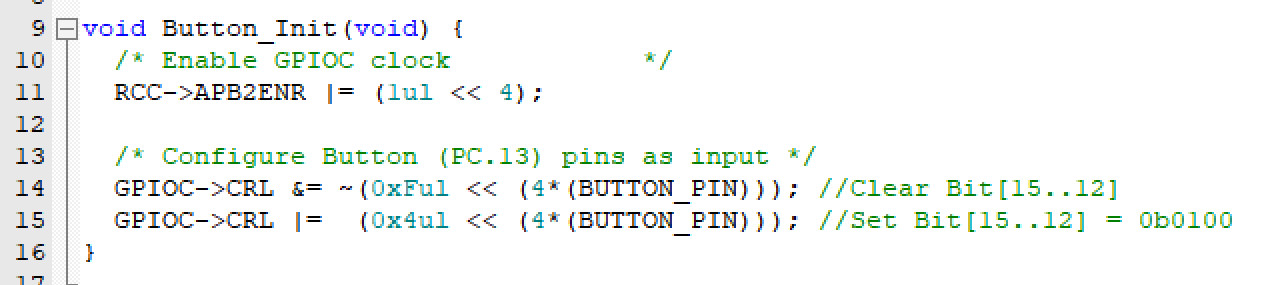
选择 Startup 启动代码和必须的CMSIS::CORE代码，还将选择外设库Framework、GPIO 和 RCC 模块代码。

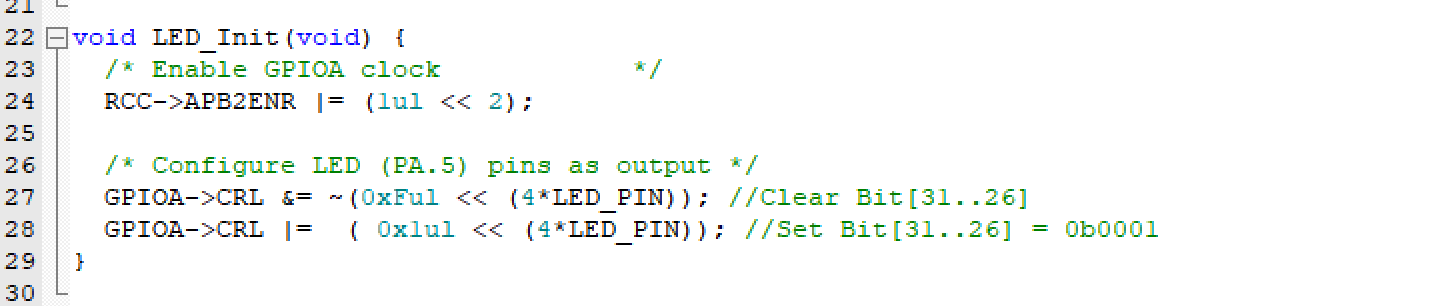


2.定义需要操作的端口号

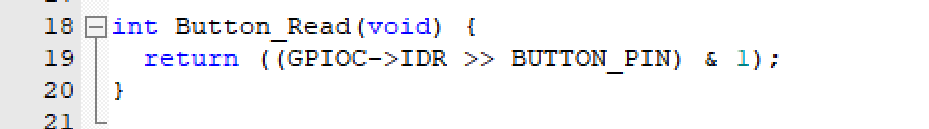


3.初始化要操作的端口

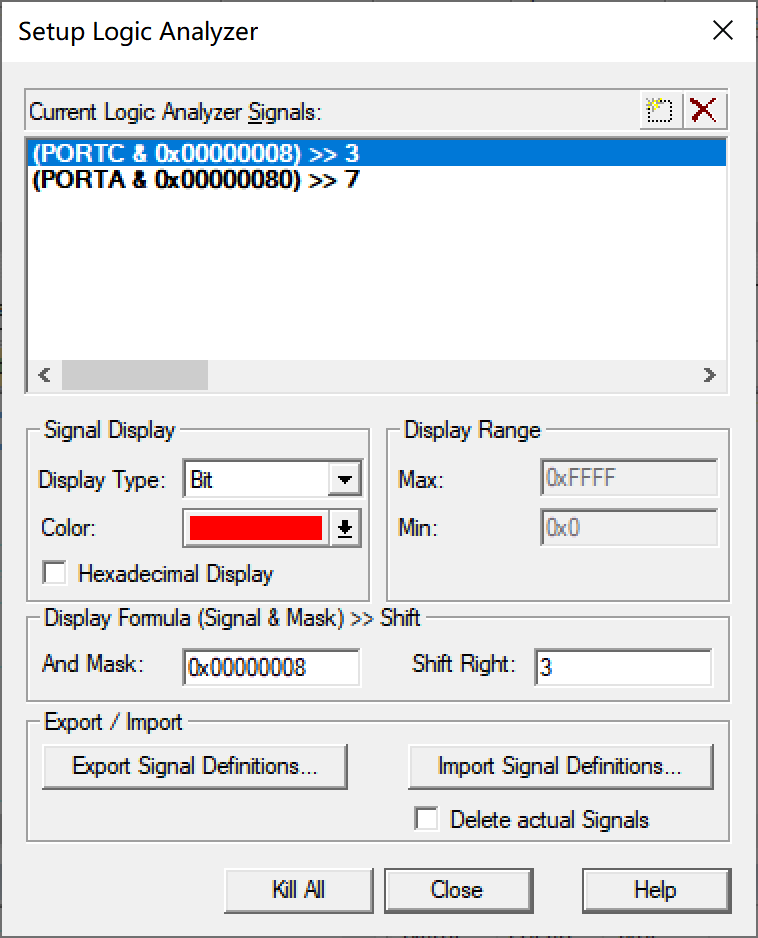




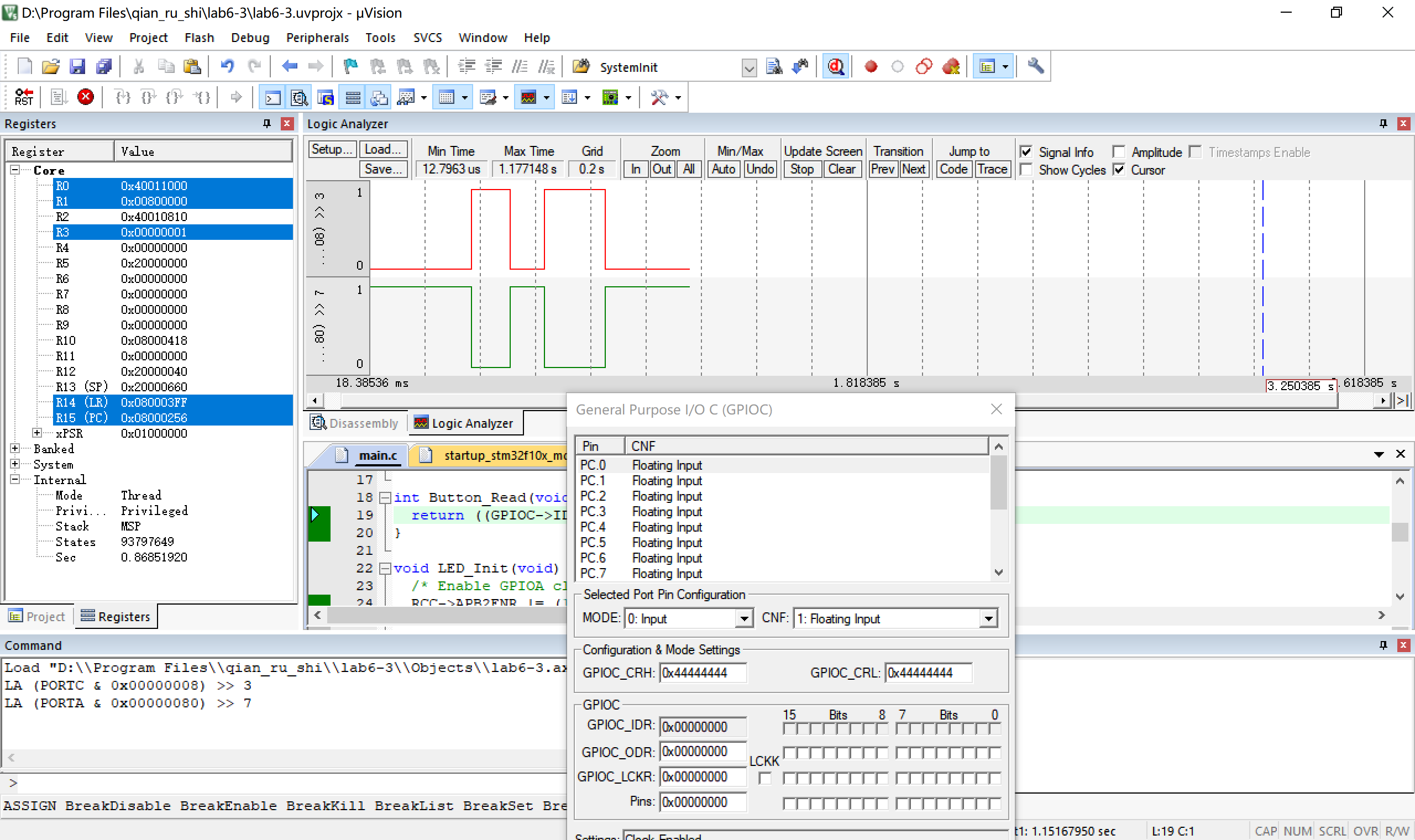
4.实现读取按钮值的函数：

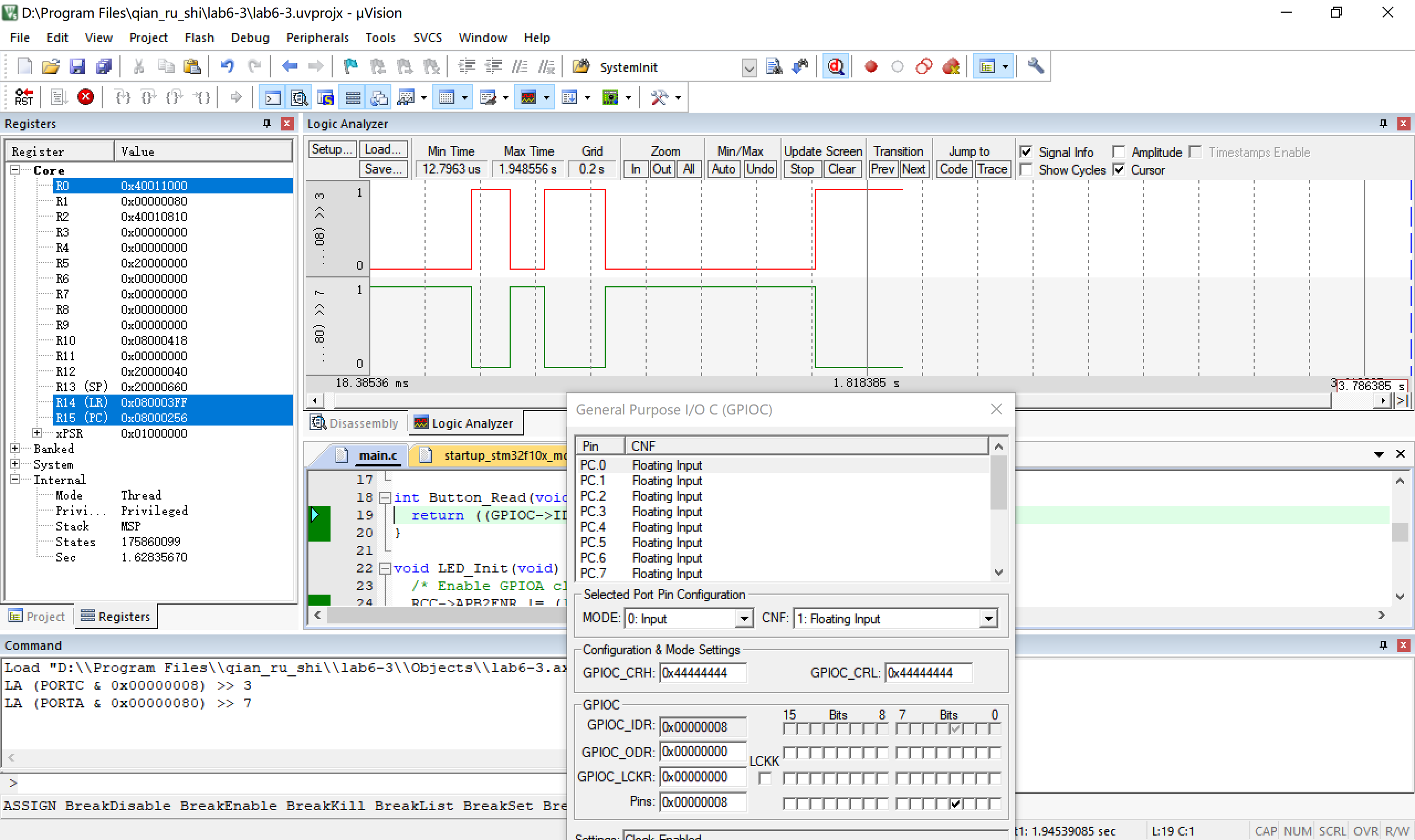


5.编译调试，设置查看：



6.运行，可以看到反向输出成功实现，由pins3操作

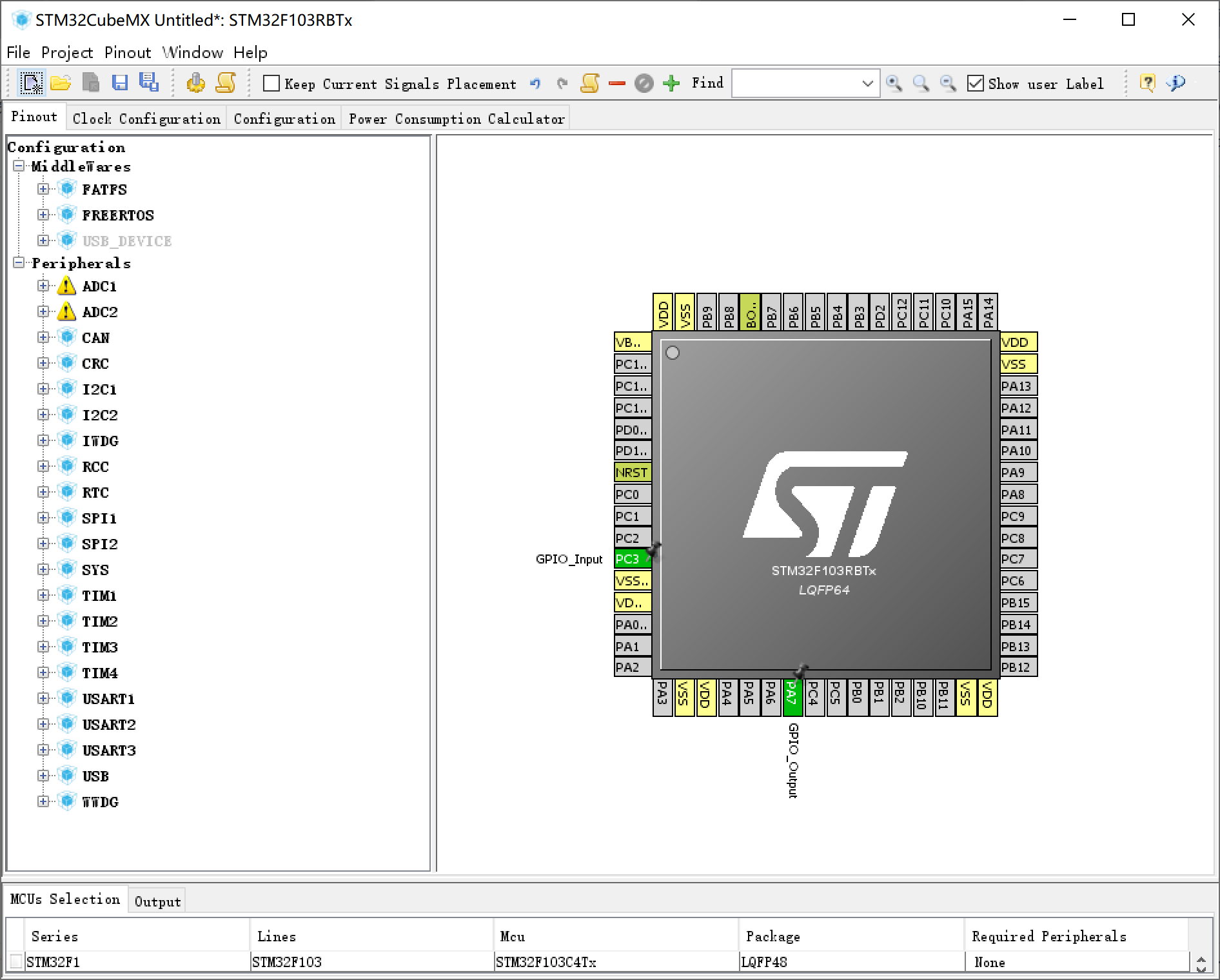




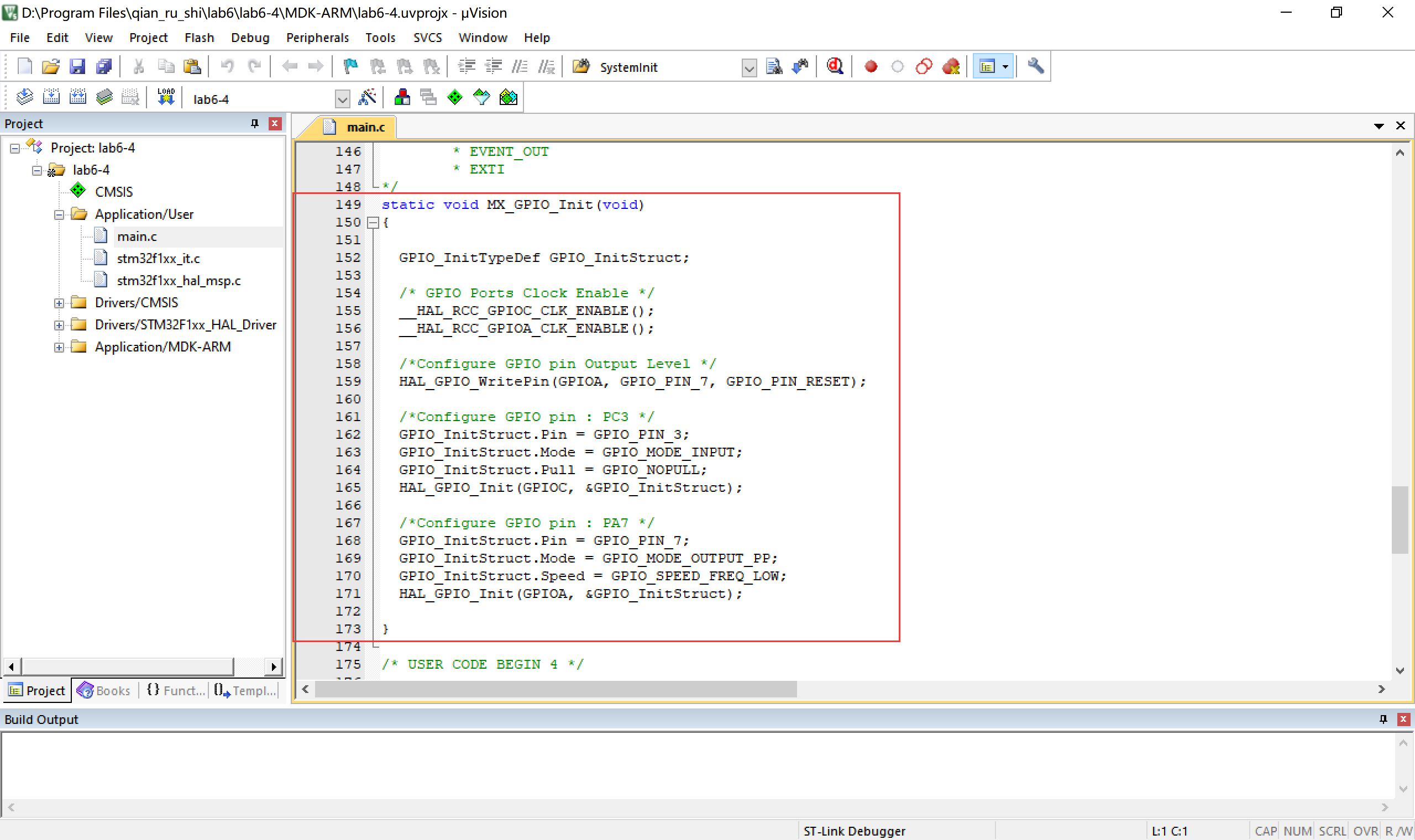
（4）**使用 STM32CubeMX 生成代码+API 操作寄存器**

1.打开 STM32CubeMX ，选择 New Project，创建工程，选择芯片：STM32F103RBTx ；

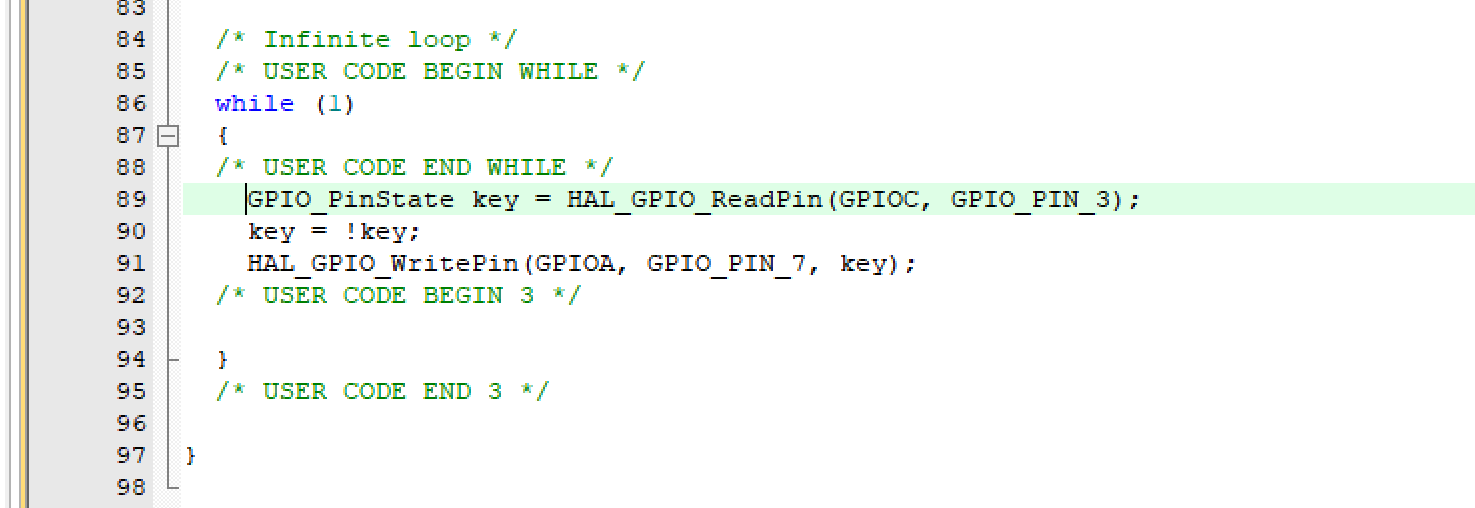
配置芯片引脚。将PA7设为输出模式，PC3为输入模式。



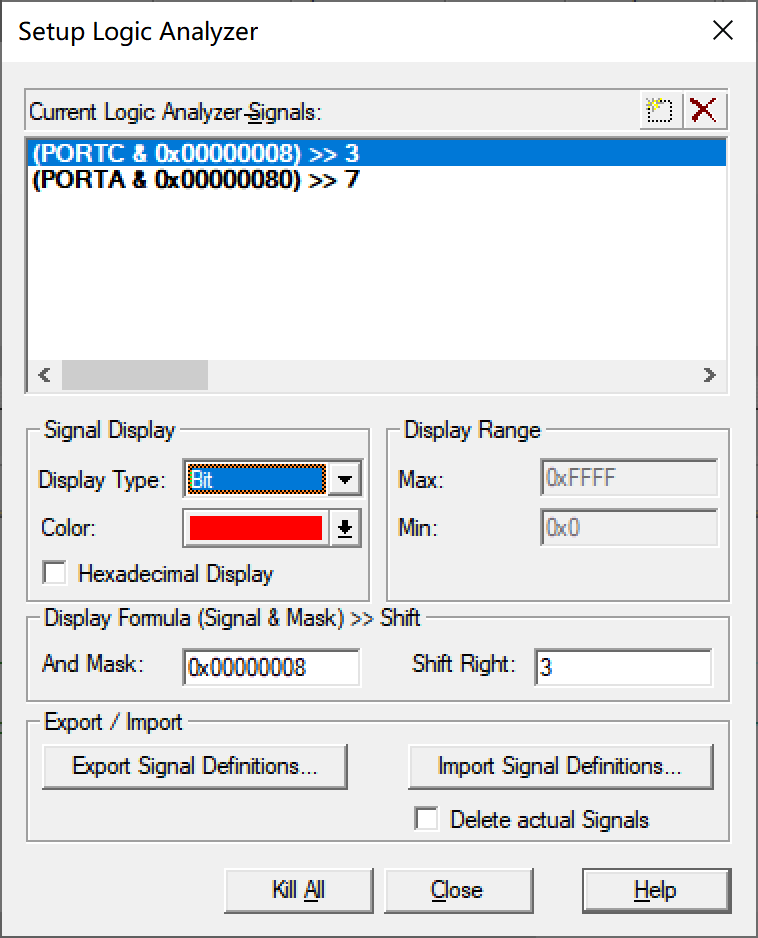
2.生成代码后，发现已经完成了初始化，查看初始化函数，可以看到在这里面完成了对GPIOA.7和GPIOC.3的配置。



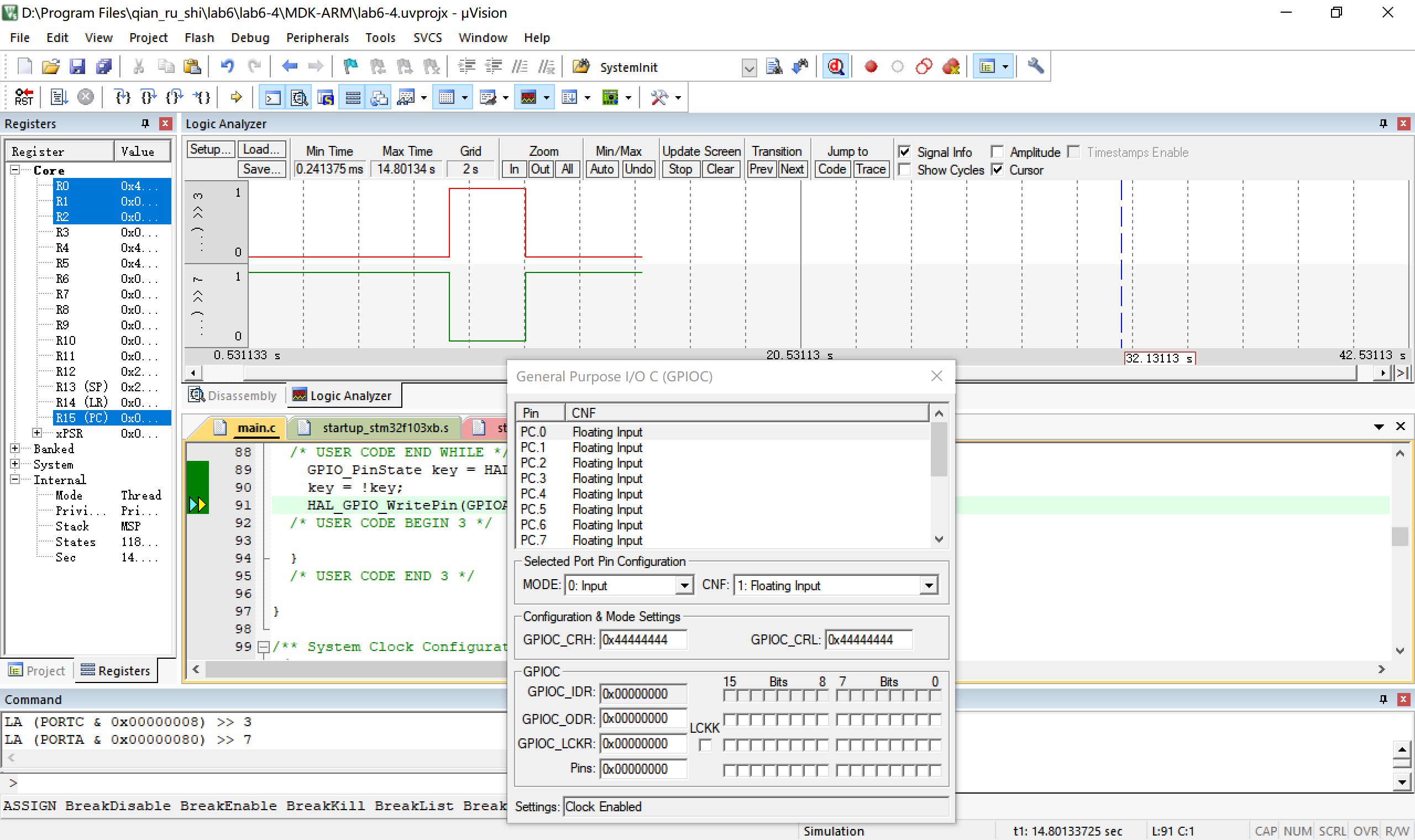
3.在 main.c 的 while(1)循环里增加下述代码：

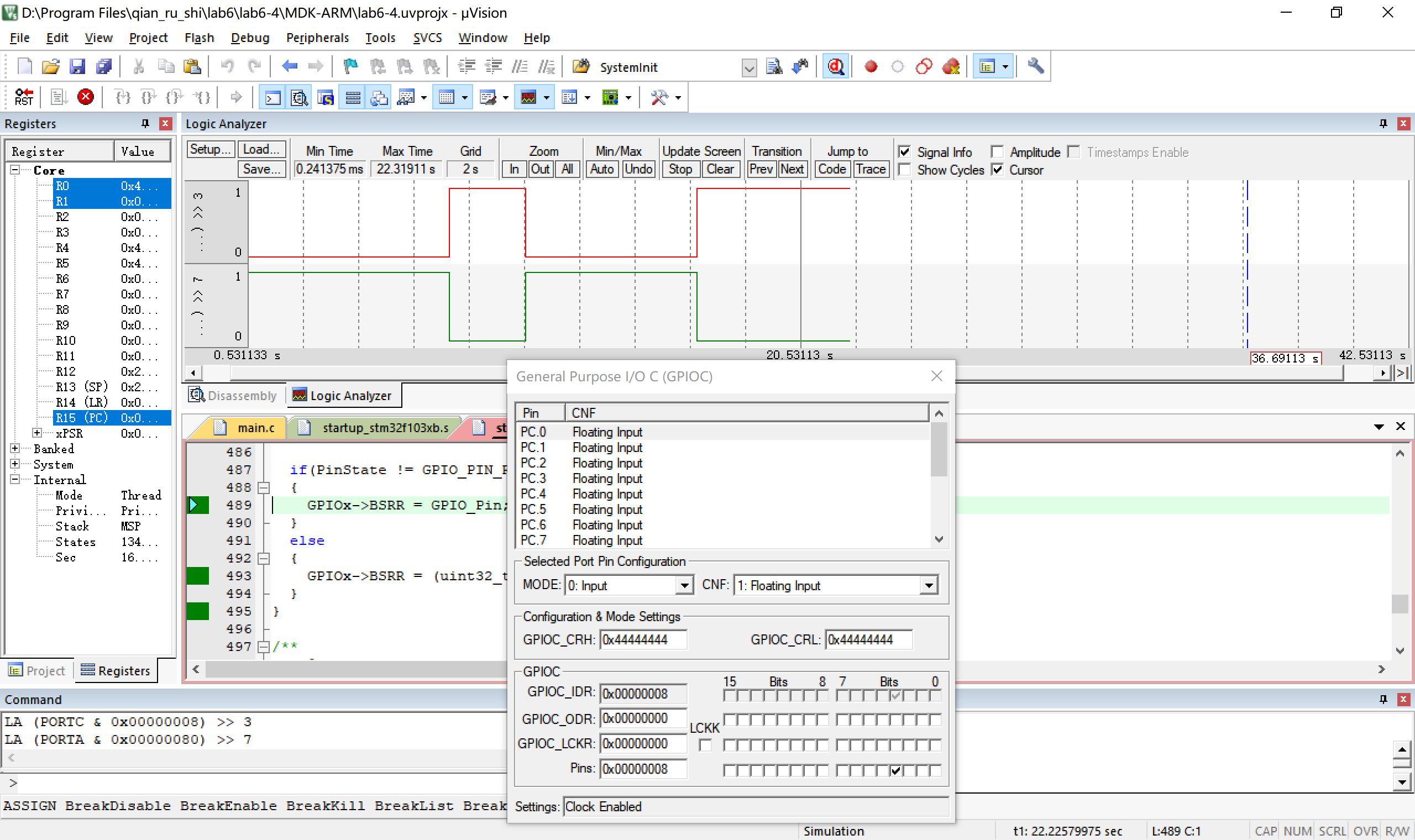


4.编译调试，添加查看：



5.运行，可以看到实现了反向输出





。

# 实验总结

在本次实验中，实现了在 MDK 环境操作 STM32 的 4 种方式，观察和分析了其不同的特点。本次实验的4种方式从上到下，自动化和抽象化程度越高，编程越容易，但前几种方法让我了解到了更多的信息，收获颇多。