**微机原理与嵌入式系统实验报告**

**雷迪 PB17061153**

**实验1 基于ASM的PROJECT**

* 1. **实验目的**

1. 掌握µVision IDE基本使用、了解一个项目编译、连接、调试的工作过程
2. 汇编代码编写的一般语法，掌握编写子程序的方法
3. 掌握常规代码调试技巧

4、理解编程者模型

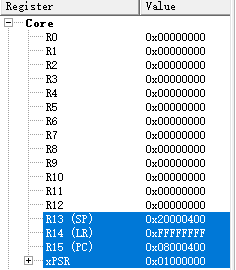
**1.2 实验内容**

1、Project的建立、编译、连接

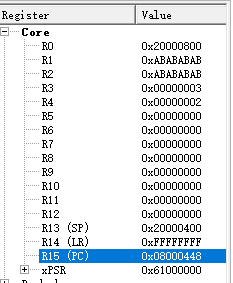
2、Project的调试（Debug）

（4）观察调试过程中通用寄存器的变化

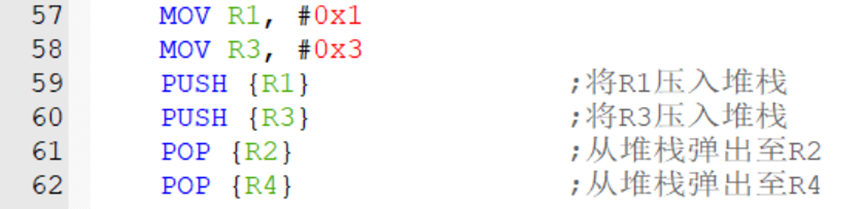
运行之前：



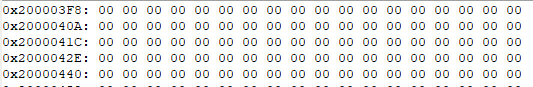
运行之后：

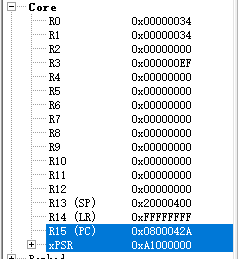


（5）观察如下代码执行前后栈指针的变化：

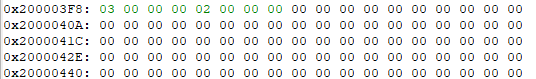


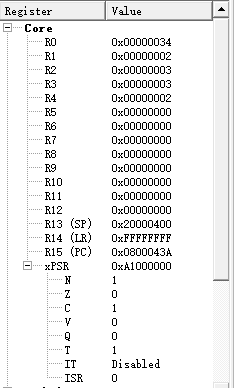
由实验软件可以得出下面的变化：（其中我把#0x1改成了#0x2）

运行后：



运行后：

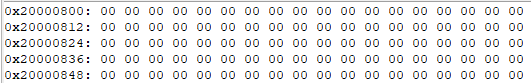




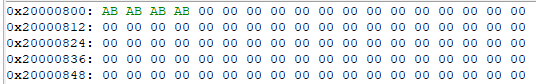
但是这个里面看不出栈的指针变化，但是可以看出来，指针是递减的，同时指向栈顶的元素。

（6）在如下代码执行前后观察存储器0x20000800地址的内容

运行之前：



运行之后：



* 1. **思考题**

（7）异常处理子程序Reset\_Handler的入口地址是？

**答：0x08000400**

（8）添加一行代码，使xPSR寄存器的Z标志位为1。

**答：**

**MOV R1,#0x34**

**CMP R1,#0x34**

（9）示例代码中，为何使用的是MSP而不是PSP。

**答：因为是工作在handler模式下，只允许使用主堆栈指针MSP。**

（10）请解释执行至Reset\_Handler中第一行代码时，为何MSP为“0x20000400”？

**答：因为在程序开头，定义了栈的大小为0x00000400，然后开始执行时，会指向栈的顶部，即为0x20000400，同时为递减堆栈。**

（11）请依据代码调试中观察到的机器指令解释伪指令“LDR R0, =0x20000800”被翻译为机器指令的执行过程？

**答：首先执行机器码4803，LDR，然后将pc寄存器向下偏移12个字节的数据存入R0中。**

（12）伪指令“LDR R0, =0x20000800”中数值“0x20000800”，被存放在哪个地址？

**答：0x08000448。**

（13）解释指示符（伪指令）“EXPORT”和“DCD”的作用？

**答：export伪指令用于程序中声明一个全局的标号，该标号可在其他的文件中引用。DCD伪指令用于分配一片连续的字存储单元并用伪指令中指定的表达式初始化。**

（14）观察代码执行过程中，PC变化的规律。并写出一条使PC递增2的指令，再写出一条使PC递增4的指令。

**答：递增2的指令：MOV R0, R1 递增4的指令：MOV R1, #0xAB。**

**实验2 基于C的PROJECT**

**2.1 实验目的**

1、掌握µVision IDE下创建C语言工程的基本步骤

2、了解µVision IDE自带CMSIS库和device的启动文件

3、掌握联机帮助查询技巧

4、掌握代码分析技巧

5、理解ARM汇编程序中的伪指令（指示符，Directive）

6、掌握C和汇编混合编程方法

**2.2实验内容**

**1、建立基于C程序的Project**

1. **代码功能验证**

（16）阅读启动文件“startup\_ARMCM3.s”和“startup\_ARMCM3.c”，分析启动的大致过程。

**答：首先设置了栈和堆的大小并进行设置，然后设置了向量表，然后调用reset handler，引导程序进入\_\_main，最后进行用户堆栈的初始化，完成启动的过程。**

（17）分析示例代码中C调用汇编子程序的过程。

**答：示例中，c程序调用的是strcopy子程序，strcopy首先将R1指向源字符串地址，取出字符内容存入R2，然后R0指向目的字符串地址，R2中内容存入R0指向内存单元，然后比较R2里面是不是全空，就可以实现复制字符串的功能。**

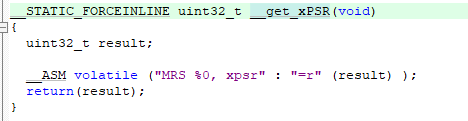
（18）分析示例代码中汇编调用C子程序的过程。

**答：示例中，汇编调用的是MY\_C\_FUNCTION子程序，通过传入两个int数，然后返回两个数的和。**

**3、代码分析技巧**

（20）分析示例代码中所调用的CMSIS-core函数的定义。浏览所打开的“cmsis\_armcc.h”文件。

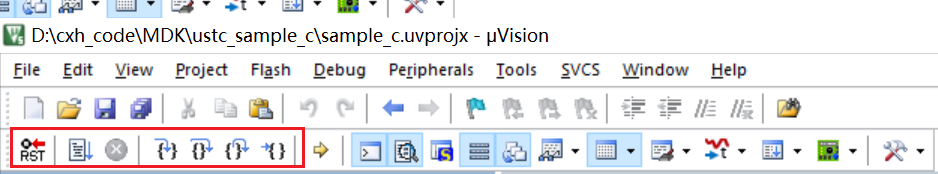
**答：访问一些汇编指令和特殊寄存器。**



**4、μVision联机资源使用**

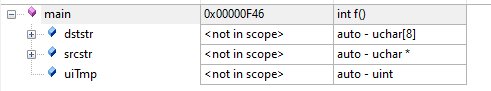
**5、Project的调试（Debug）**

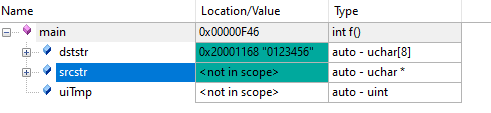
（25）通过试验分析下图所示Debug工具栏（红色框内）各个图标对应功能的区别。

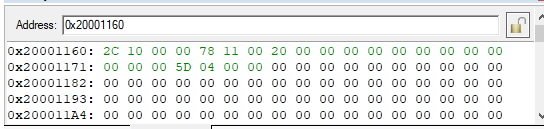


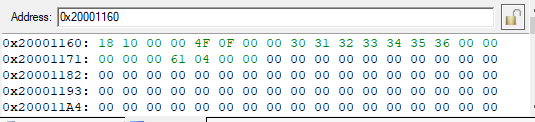
**答：第一个是重新设置cpu，相当于清零大部分寄存器；第二个是向下运行；第三个是停止代码的执行；第四个是进入函数的单步执行；第五个是不进入函数的单步执行；第六个是跳过当前函数执行；第七个是执行到光标位置。**

（26）观察变量“srcstr”和“dststr”，同时观察Memory窗口对应地址的值

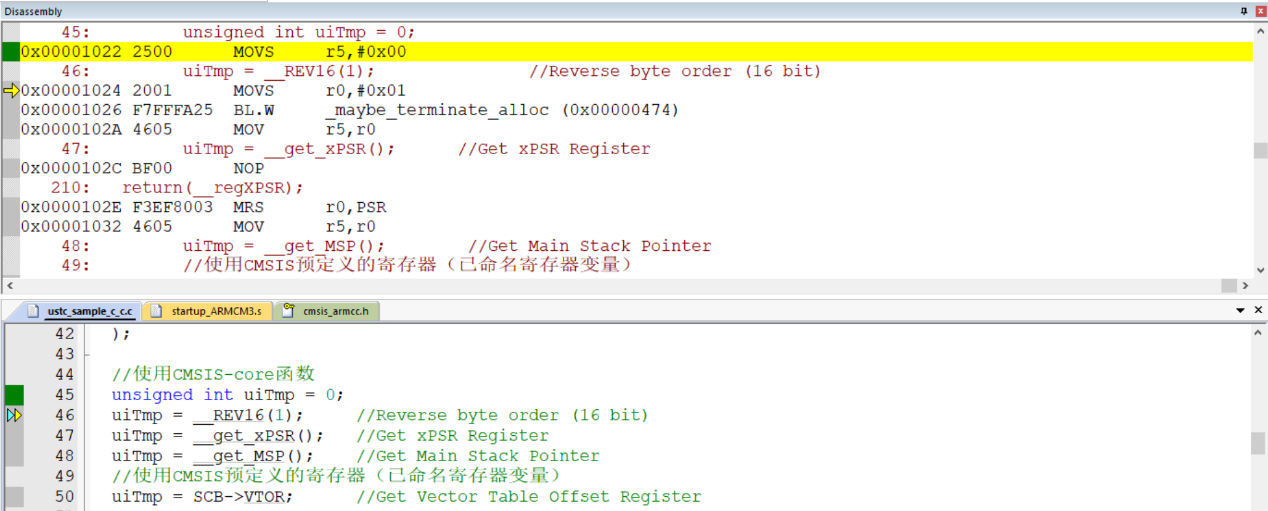








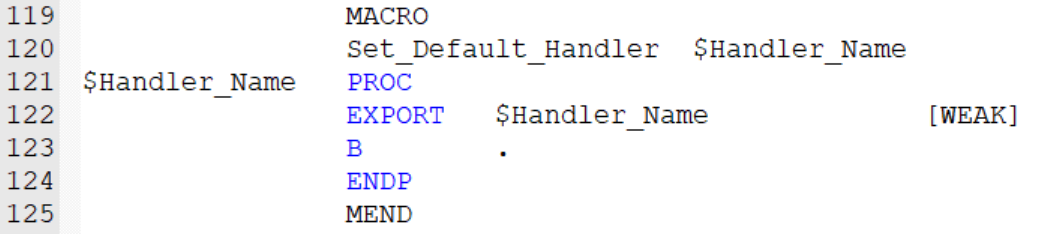
（27）分析下图所示反汇编窗口中机器指令与源代码窗口中C代码的对应关系。



**答：第45行对应第一条机器指令，第46行对应后面三条机器指令，第47行对应后面四条机器指令。**

* 1. **思考题**

（28）分析启动文件“startup\_ARMCM3.s”中图下宏定义的含义？并写出$Handler\_Name等于NMI\_Handler时，该宏定义展开后的代码。



**答：表示如果有多个地方出现了$Handler\_Name，那么就不改变原来的量的定义。**

**NMI\_Handler PROC**

**EXPORT NMI\_Handler [WEAK]**

**B .**

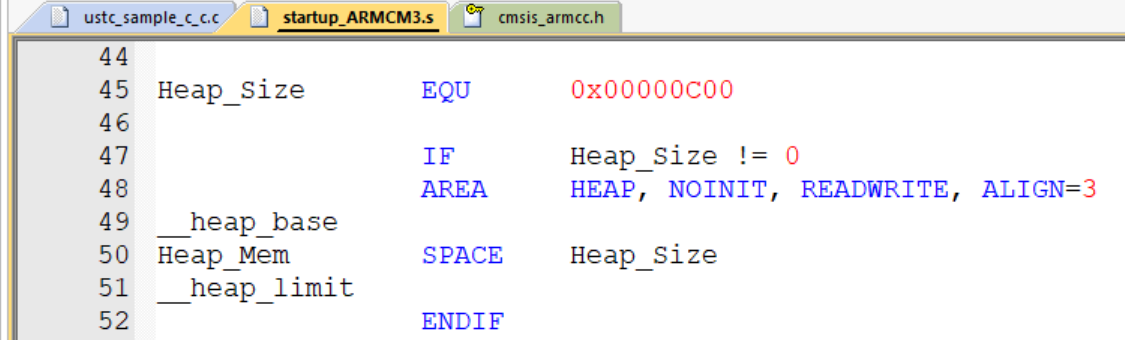
**ENDP**

（29）依据调试结果，示例c程序中如下行中字符串"Hello USTCer\n"被保存在存储器的什么位置（写出存储器地址）？



**答：0x00000F80**

（30）请结合“startup\_ARMCM3.s”文件中如下代码分析示例中“dststr”的地址为何是“0x20001160”？



**答：初始化的堆栈头地址为0x20000178，堆空间为0x00000C00，栈空间为0x00000400，初始化的栈顶为0x20001178.栈是向下增长的，所以dststr存放的地址是0x20001168.**

**实验3 基于STM32库的GPIO与定时器**

* 1. **实验目的**

1. 掌握µVision IDE中基于ST公司STM32库建立project的流程
2. 了解ST公司提供的TIM、GPIO相关库函数
3. 了解STM32F10X系列芯片定时器相关的寄存器功能。
4. 掌握µVision IDE中外设仿真模块（GPIO）的使用
   * 1. 学会利用外设仿真模块（GPIO）观察I/O引脚输出
     2. 学会利用外设仿真模块（GPIO）模拟I/O引脚的输入
5. 掌握µVision IDE逻辑分析模块（Logic Analyzer）的使用
   1. **实验内容**

1、下载ST公司STM32库及芯片手册

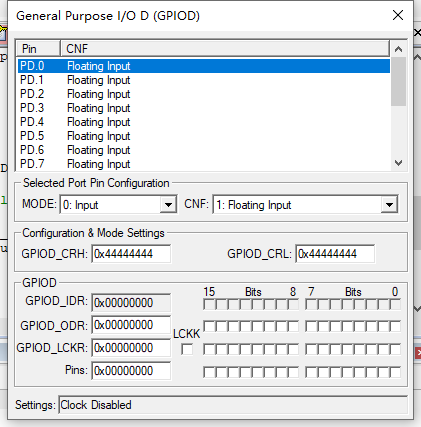
2、建立基于STM32库的Project

3、配置Project的头文件目录、预编译参数、Simulator

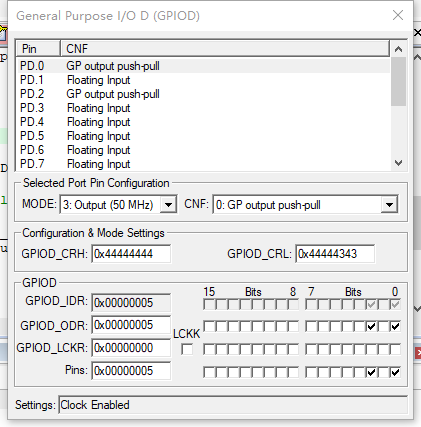
4、Debug时使用外设仿真功能验证GPIO输入和输出

（40）观察示例代码中一下代码行执行前后GPIOD的变化

执行之前：

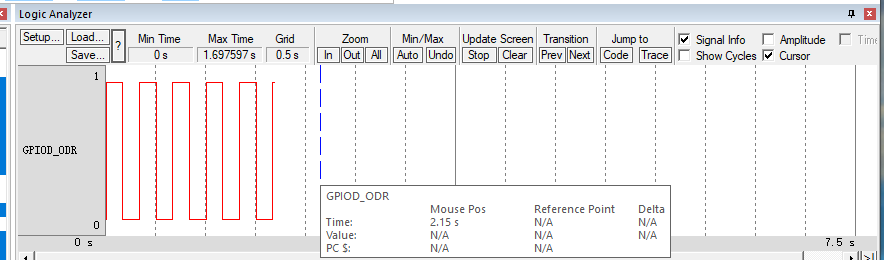


执行之后

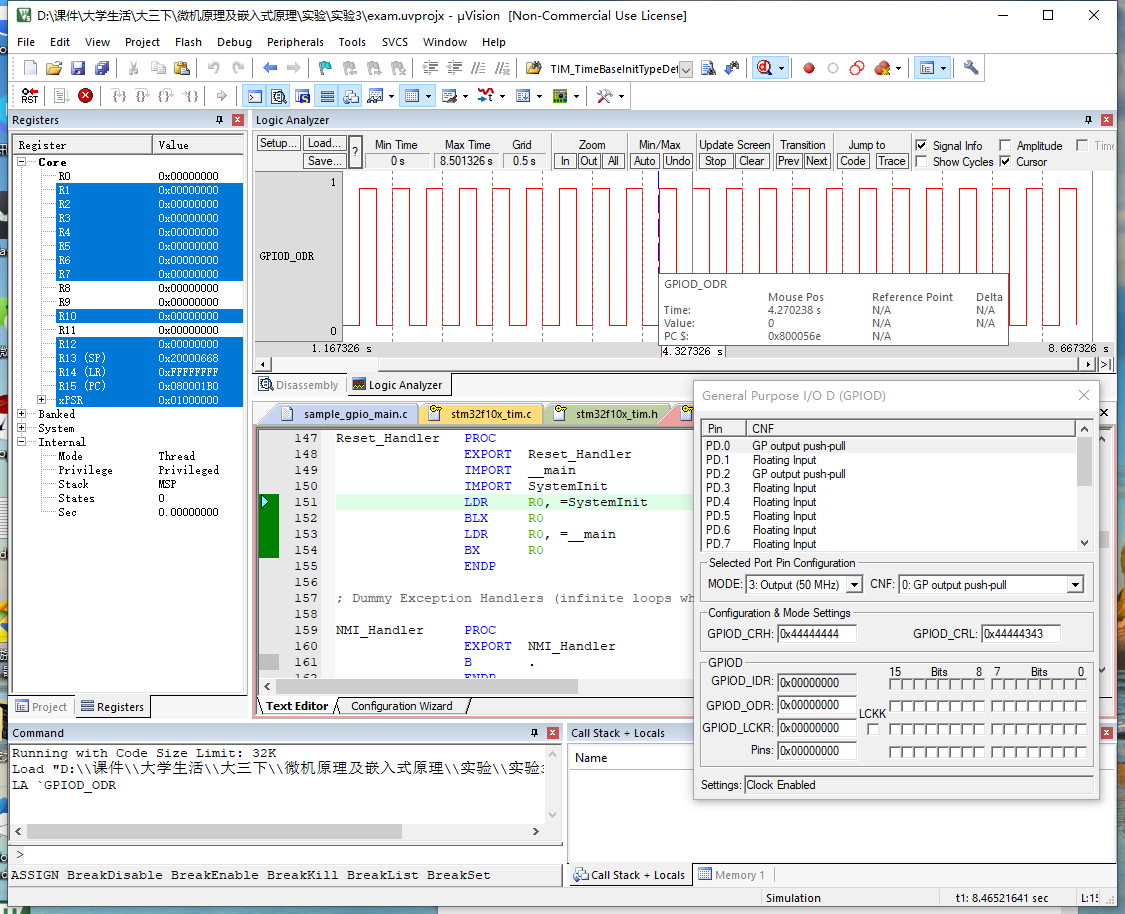


1. Debug时使用Logic Analyzer观察GPIO输出

（41）通过µVision IDE自带的逻辑分析模块（Logic Analyzer）观察GPIOD的变化。

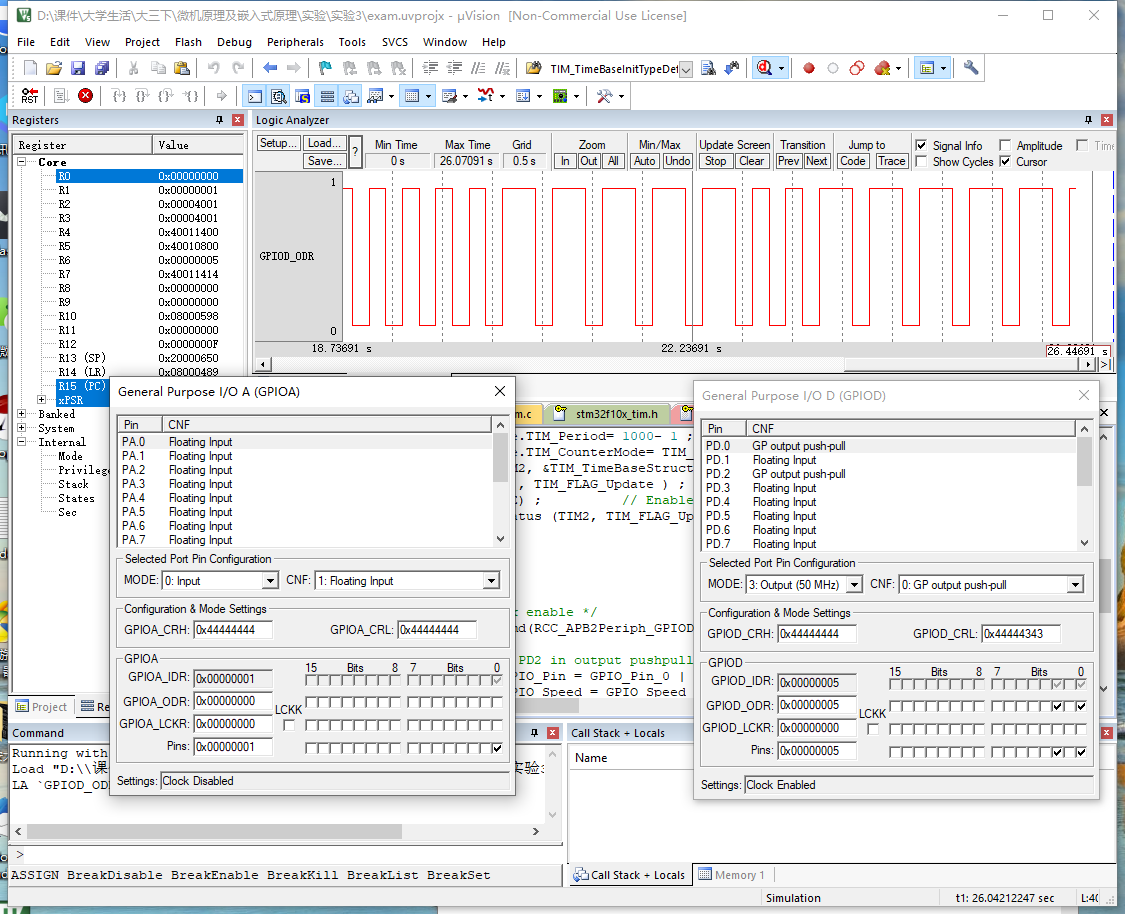


在Logic Analyzer窗口和GPIOD窗口同时观察到PD0和PD5位的变化。



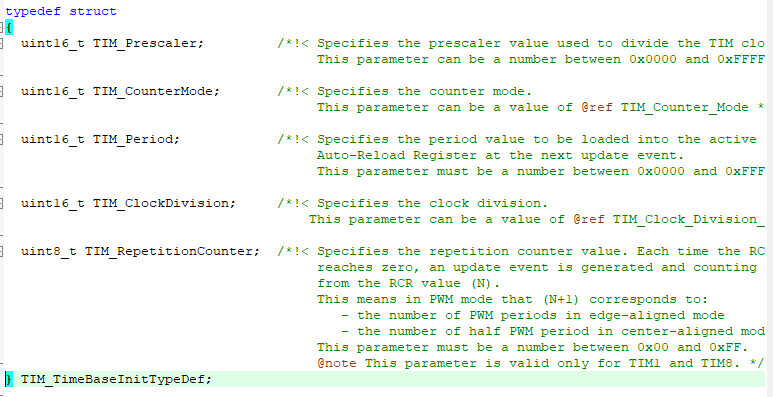
6、使用外设仿真和Logic Analyzer验证GPIO输入和输出

（42）通过再GPIOA窗口单击PA0引脚位置，模拟PA0输入“1”，观察Logic Analyzer的变化，并与示例代码进行验证。



7、定时器的配置

（43）阅读代码的注释信息，了解该结构体各成员的含义。



typedef struct

{

uint16\_t TIM\_Prescaler;// 指定用于分频TIM时钟的预分频器值。该参数可以是0x0000和0xFFFF之间的数字

uint16\_t TIM\_CounterMode;// 指定计数器模式。该参数可以是@ ref TIM \_计数器\_模式的值

uint16\_t TIM\_Period; //指定要加载到活动中的周期值。下一次更新事件时自动重新加载寄存器。该参数必须是0x0000和0xFFFF之间的数字。

uint16\_t TIM\_ClockDivision;// 指定时钟划分。该参数可以是@ ref TIM \_ Clock \_除法\_CKD的值

uint8\_t TIM\_RepetitionCounter;// 指定重复计数器值。每次RCR向下计数。达到零时，会生成更新事件并重新开始计数。来自RCR值(N)。

这意味着在脉宽调制模式下，(N 1)对应于:

-边沿对齐模式下的脉宽调制周期数

-中间对齐模式下半个脉宽调制周期的数量

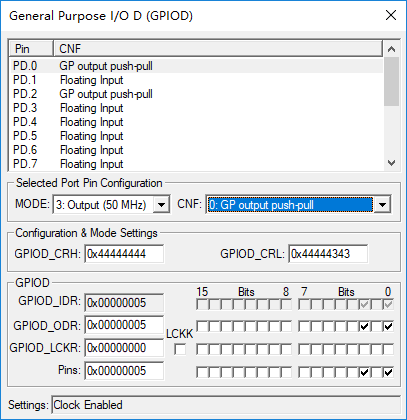
该参数必须是0x00和0xFF之间的数字。

@注意该参数仅对TIM1和TIM8有效。

} TIM\_TimeBaseInitTypeDef;

* 1. **思考题**

（44）阅读讲义8.5.3小节，及STM32芯片手册，解释下图中“CRL、CRH、IDR、ODR、LCKR”几个寄存器的作用。



**答：CRL寄存器：端口配置寄存器，配置GPIO工作模式，用于控制GPIOX（X表示A—G）的低8位（Pin7-Pin0）；**

**CRH寄存器：端口配置寄存器，配置GPIO工作模式，用于控制GPIOX（X表示A—G）的高8位（Pin15-Pin8）；**

**IDR寄存器：端口输入数据寄存器，只用了低16位。该寄存器为只读寄存器，并且只能以16位的形式读出。读出的值为对应IO口的状态。**

**ODR寄存器：端口输出数据寄存器，也只用了低16位。该寄存器虽然为可读写，但是从该寄存器读出来的数据都是0。只有写是有效的。其作用就是控制端口的输出。**

**LCKR寄存器：端口配置锁定寄存器，端口锁定后下次系统复位之前将不能再更改端口位的配置。**

（45）解释代码“GPIOD->BSRR = 0x00000085; ”的作用。

**答：设置I/O端口7，端口2和端口0为高，保持其他I/O端口不变。**

（46）解释代码“GPIOD->BRR = 0x00000080;”的作用。

**答：设置I/O端口7为低，保持其他I/O端口不变。**

（47）解释库函数“GPIO\_ReadInputDataBit (GPIOD,GPIO\_Pin\_0)”的作用。

**答：用来读取GPIOD的I/O端口0状态的速率。**

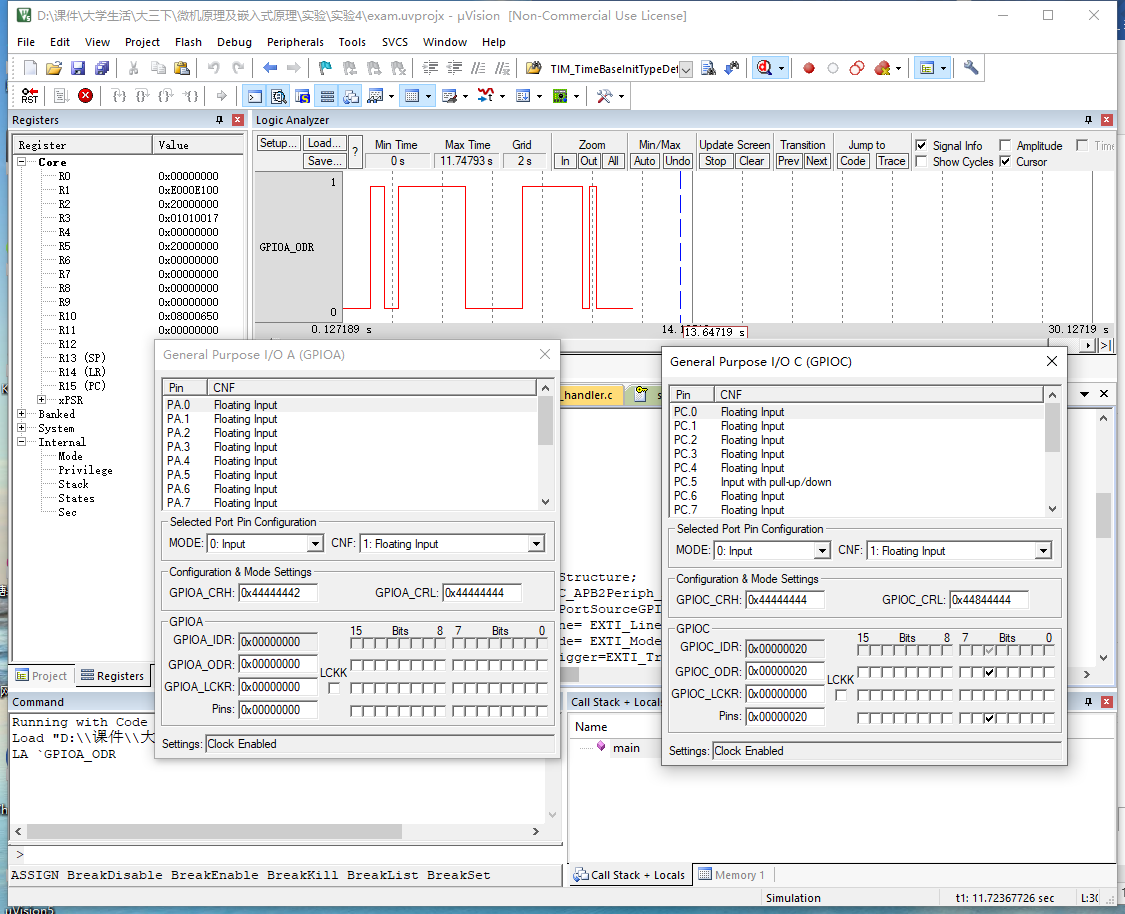
**实验4 基于STM32库的中断**

* 1. **实验目的**

1、掌握EXTI中断配置流程

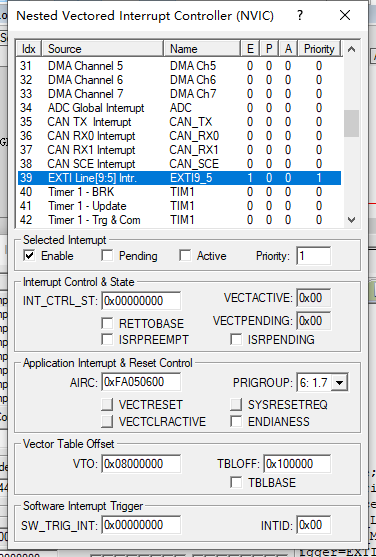
1. 理解异常向量表
2. 理解异常优先级配置
3. 了解ST公司提供的TIM、NVIC相关库函数
4. 掌握µVision IDE中外设仿真模块（NVIC）的使用
   1. **实验内容**
5. **建立基于STM32库的Project**
6. **配置Project的头文件目录、预编译参数、Simulator**
7. **使用Logic Analyzer和外设仿真功能验证EXIT及GPIO输出**

（59）F5运行示例代码，在GPIOC窗口可以输入PC5（模拟中断信号）。随着PC5信号的变化，可以在Logic Analyzer窗口和GPIOA窗口同时观察到PA8的变化。



1. **观察NVIC寄存器组**

（60）观察与外部中断#5有关的寄存器信息。



**4.3思考题**

（61）为什么通过GPIOC的窗口（Peripherals🡪General purpose I/O🡪GPIOC）模拟PC5（中断信号）输入的时候，改变两次PC5后PA8才会发生变化？

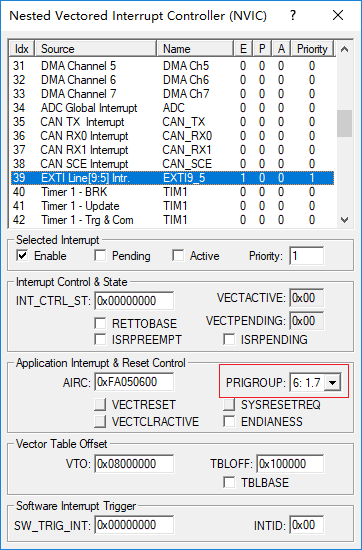
**答：因为在实际操作中，每当产生中断信息后，芯片会自动复位，但是按照软件进行模拟的时候，并没有自动产生复位，所以需要改变两次PC5，相当于复位一次后，才能发生变化。**

（62）请通过调试获得EXTI9\_5\_IRQHandler()的入口地址（应该是0x800026A），这个地址保存在在异常向量表什么位置？

**答：入口地址为0x8000026A**

**通过查询芯片手册可以知道，会保存在0x0000009C的位置，即向量表开始地址往后偏移9C的位置。**

（63）解释下图中PRIGROUP的含义（请查阅讲义5.5.1小节），PRIGROUP和AIRC（Application Interrupt & Reset Control寄存器）是什么关系？



**答：这是复位控制寄存器的PRIGROUP域，通过设置该域来完成优先级配置。此时表示抢占优先级域是Bit[7]，子优先级域Bit[6:0]。**

**PRIGROUP是32位寄存器AIRC中的第[10:8]位。**