**微嵌自主实验 实验报告**

PB17061193 孙晨曦

# 实验一 基于ASM的PROJECT

## 1.1 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、掌握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

### 1.2.1 Project的建立、编译、连接

（1）创建Project，并添加ustc\_sample\_asm.s。

（2）编译、连接Project——Bulid Target。

### 1.2.2 Project的调试

（3）调试：电脑上模拟，尝试单步运行、运行到光标位置、设置断点等功能

（4）观察调试过程中通用寄存器的变化

（5）观察16-21条指令执行前后栈指针的变化

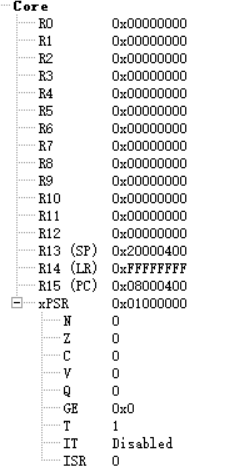
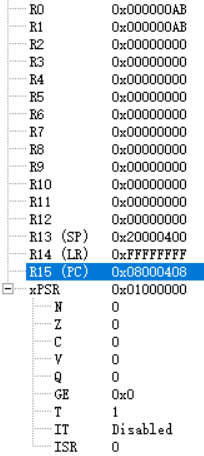
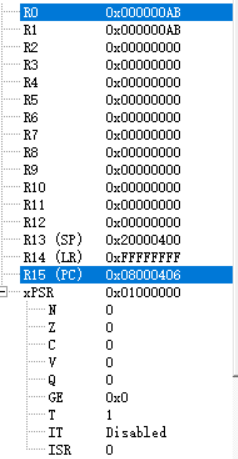
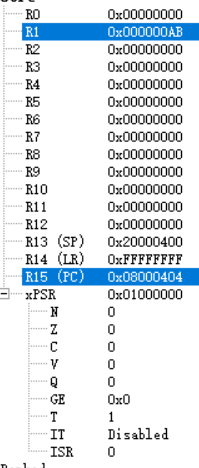
（6）观察在如下代码执行前后观察存储器0x20000800地址的内容

记录如下：三条代码—三个变化为一组

1 MOV R1, #0xAB ;立即数需要满足特定的规则

2 MOV R0, R1 ;寄存器R1中的数值装载到寄存器R0

3 MOVS R0, R1 ;寄存器R1中的数值装载到寄存器R0，更新APSR

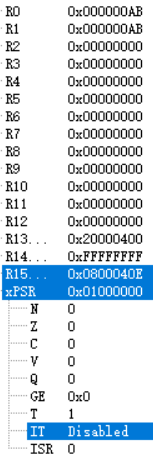
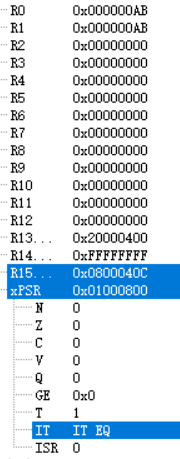
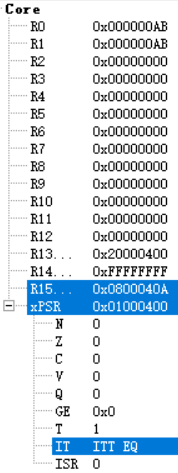
 

（Reset->1-3条指令）

4 ITT EQ ;随后的两条指令是条件执行的

5 MOVEQ R0, R1 ;当APSR中标志位“Z == 1”时执行，否则不执行

6 MOVEQ R2, R1 ;当APSR中标志位“Z == 1”时执行，否则不执行



（4-6条指令）

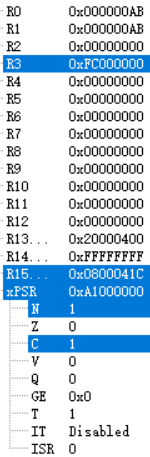
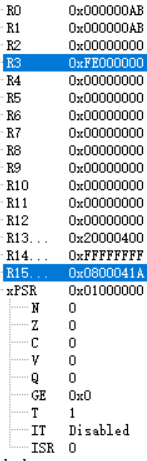
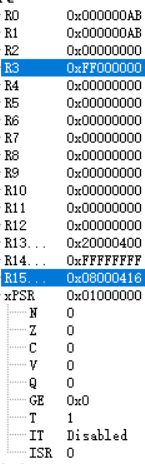
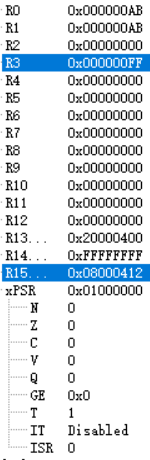
;观察PSR

7 MOV R3, #0xFF ;立即数需要满足特定的规则

8 LSL R3, #24

9 LSL R3, #1

10 LSLS R3, #1



（7-10条指令）

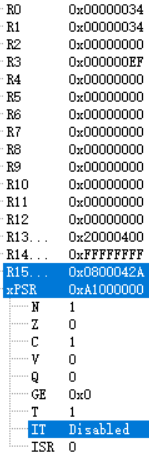
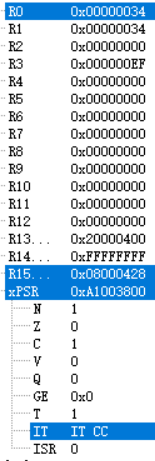
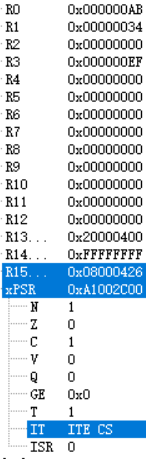
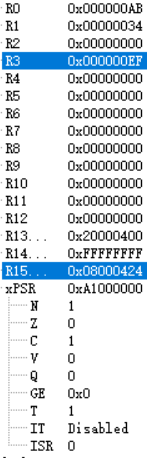
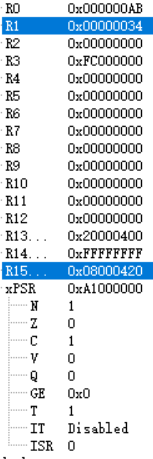
11 MOV R1,#0x34

12 MOV R3,#0xEF

13 ITE CS ;随后的两条指令是条件执行的

14 MOVCS R0, R1 ;当APSR中标志位“C == 1”时执行，否则不执行

15 MOVCC R2, R3 ;当APSR中标志位“C == 0”时执行，否则不执行



（11-15条指令）

;观察SP

16 MOV R1, #0x1

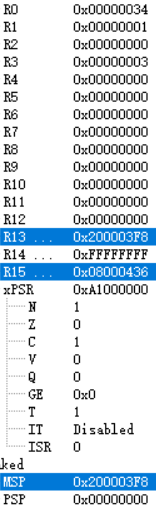
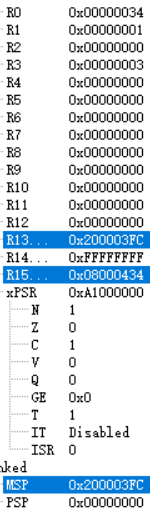
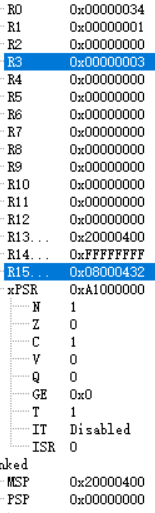
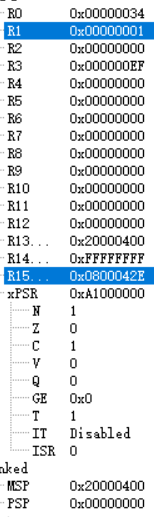
17 MOV R3, #0x3

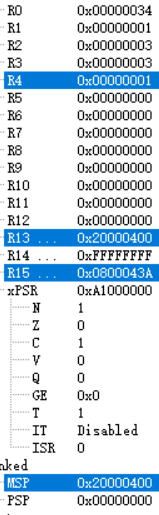
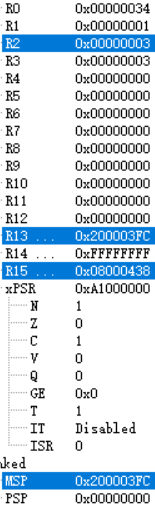
18 PUSH {R1} ;将R1压入堆栈

19 PUSH {R3} ;将R3压入堆栈

20 POP {R2} ;从堆栈弹出至R2

21 POP {R4} ;从堆栈弹出至R4





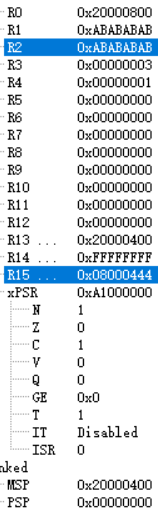
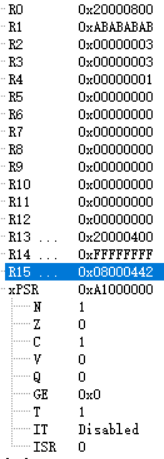
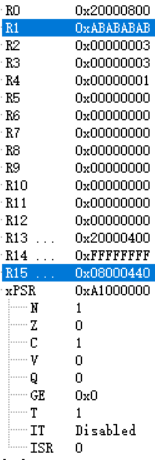
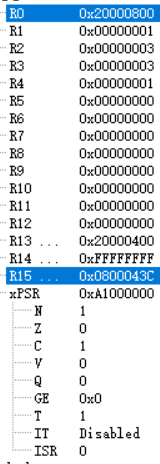
（16-21条指令）

22 LDR R0, =0x20000800 ;把0x12345678装载到R0

23 MOV R1, #0xABABABAB ;立即数需要满足特定的规则

24 STR R1, [R0]

25 LDR R2, [R0]



（22-25条指令）



（24条对寄存器的影响）

### 1.2.3 思考题

（7）异常处理子程序Reset\_Handler的入口地址是？

答：0x0800 0400

（8）添加一行代码，使xPSR寄存器的Z标志位为1。

答：在

21 POP {R4} ;从堆栈弹出至R4

后加上CMP R3,R2

（9）示例代码中，为何使用的是MSP而不是PSP。

答：复位后处于线程模式特权级，默认使用MSP，且程序中没有MSR/MRS指令进行指定。

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

答：因为设置的栈大小为0x0000 0400，而默认的SRAM区从0x2000 0000开始。此设定下，0x8000 0000处指令为DCW 0x4000，0x8000 0002处指令为DCW 0x2000

（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：被翻译为：

LDR r0,[pc,#12] ; @0x0800044C

即：从当前PC值+12得到的地址的操作数送入R0。当前指令的PC值为0x0800 043E，字对齐后为0x8000 0440查看0x8000 044C，有：，小端模式，为0x2000 0800。

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

答：0x0800 044C

（13）解释指示符（伪指令）“EXPORT”和“DCD”的作用？

答：

EXPORT：用于在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。EXPORT可用GLOBAL代替，标号在程序中区分大小写，[WEAK]选项声明其他的同名标号优先于该标号被引用。

DCD：用于分配一段连续的字存储单元并用伪指令中指定的表达式进行初始化。

（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：PC有时+2，有时+4，这和上一条指令的长度有关，指令为16位时为前者，为32位时为后者。

递增2：MOV r0,r1

递增4：MOV r0,#0x1

# 实验二 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

### 2.2.1 建立基于C程序的Project

（15）通过IDE环境的主菜单“Project”“New µVision Project”创建，选择ARMCM3板，使用自带的CMSIS库和对应的Device的StartUp，使用Simulator调试，用Compiler v5，并勾选C99选项。

### 2.2.2 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：大致为：

汇编： S1 栈初始化——指定栈的大小，根据大小初始化。

S2 堆初始化——指定堆的大小，若不为0，则根据起始和结束位置初始化。

S3 指示编译器8字节对齐，接下来的指令为THUMB指令

S4 定义中断向量表——声明向量表起始、结束、大小标号，后为按优先级给中断向量分配空间，并用相应标号初始化，并预留10-224中断仅分配空间不定义。

S5 定义Reset ISR——声明向量标号，引入系统初始化标号，引入main函数标号，首先跳转至系统初始化执行返回，再跳转至main函数。

S6 定义HardFault ISR——声明标号，并跳转至当前行，即没有任何操作。

S7 用宏定义定义其他中断ISR——硬件错误不用宏是因为会引发错误

S8 全局声明堆栈标号，用于用户自定义

C： S1 包含头文件<ARMCM3.h>

S2 定义系统时钟频率

S3 声明中断向量表

S4 设置系统时钟频率

S5 系统初始化函数——若\_\_VTOR\_PRESENT已定义且值为1，则进行向量表重定义；并初始化时钟。

（17）分析示例代码中C调用汇编子程序的过程。

答：strcopy(dststr,srcstr);

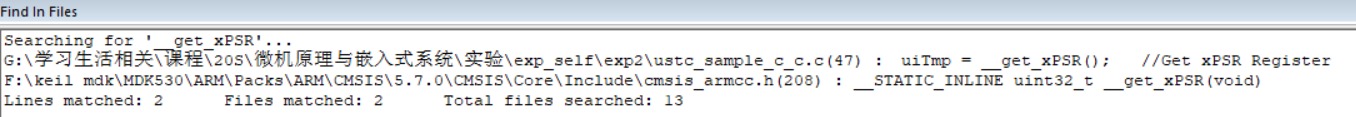
有两个参数，dststr的起始地址放在R0中，srcstr的起始地址放在R1中，当前程序的地址被放在LR中，随后PC更新至strcopy标号的位置，执行子程序，子程序中最后跳转至LR，结束调用。

（18）分析示例代码中汇编调用C子程序的过程。

答：汇编程序call\_C首先将参数1赋给R0，参数2赋给R1，然后BL跳转至MY\_C\_FUNCTION标号处，C程序执行结束后，将return 值保存在R0中，而后跳回LR的地址，结束调用。

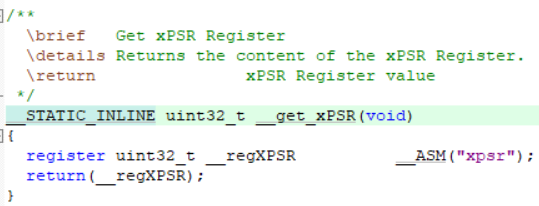
### 2.2.3 代码分析技巧

（19）在当前Project内所有文件中查找“\_\_get\_xPSR”关键字。



在ustc\_sample\_c\_c.c和cmsis\_armcc.h中有该关键字。





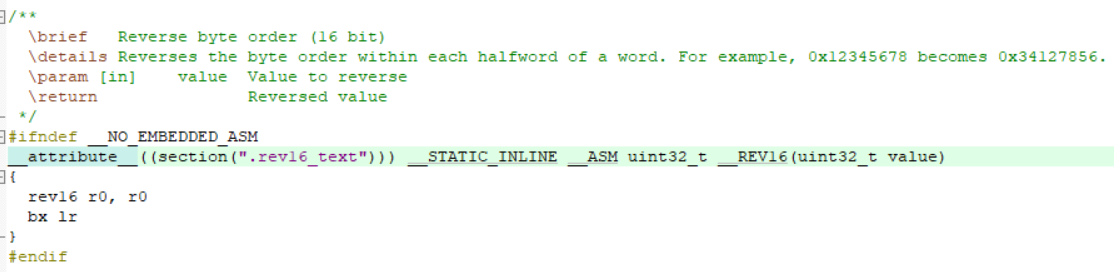
（20）分析示例代码中所调用的CMSIS-core函数的定义。如下图所示在函数位置点击鼠标右键（或热键F12），跳转至\_get\_xPSR()的定义文件。浏览所打开的“cmsis\_armcc.h”文件。

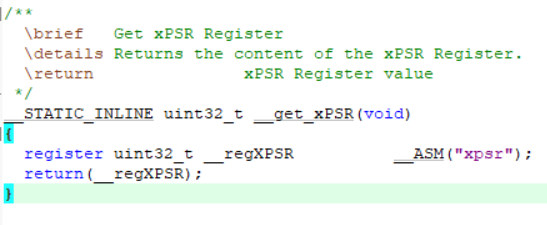
答：

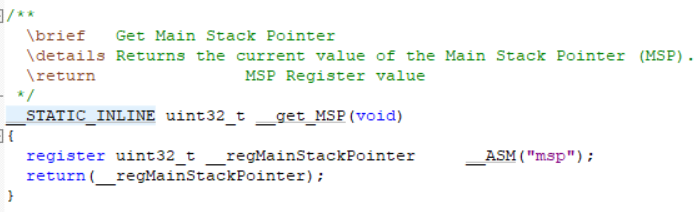
\_\_REV16(1)：将一个字的每个半字以字节为单位翻转。

\_\_get\_xPSR()：返回XPSR的内容。

\_\_get\_MSP()：返回MSP的值。

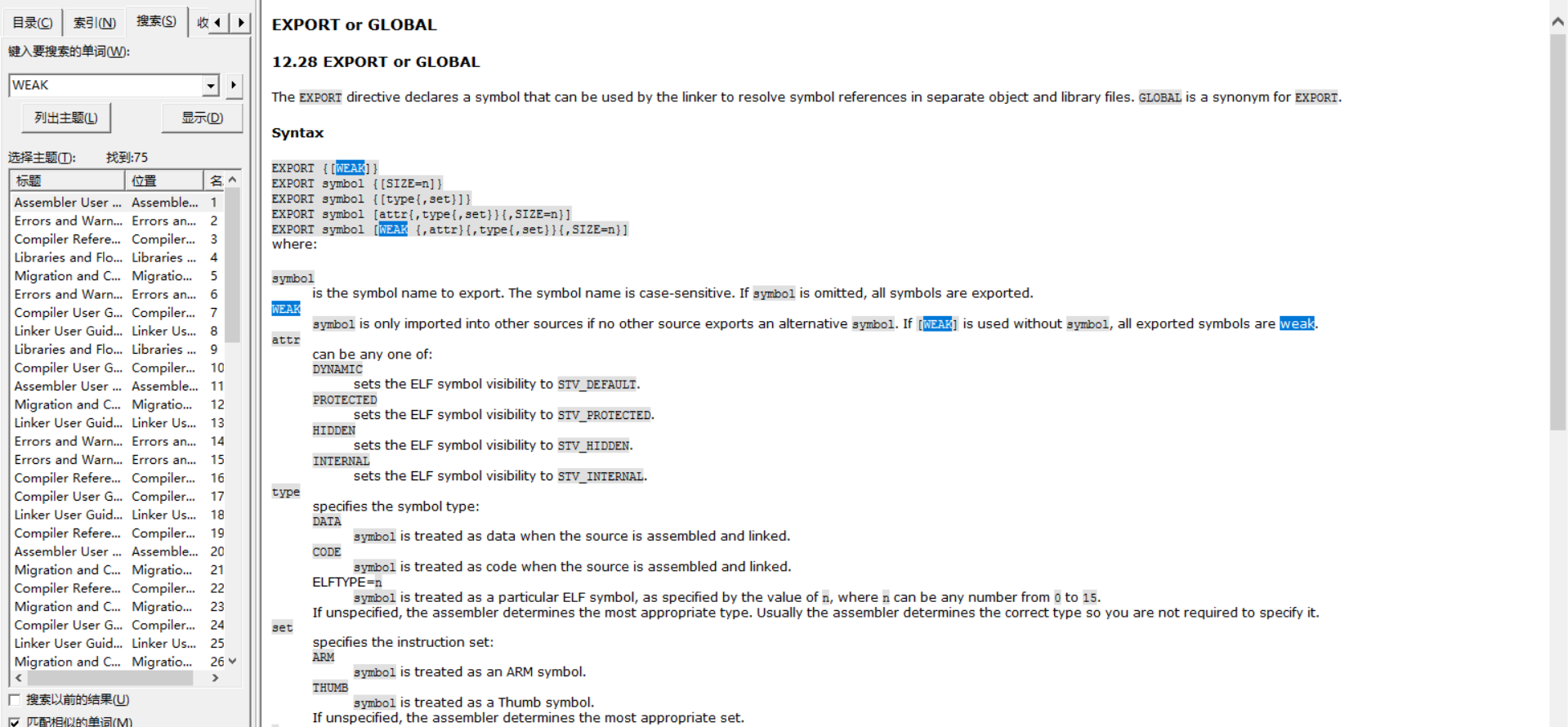






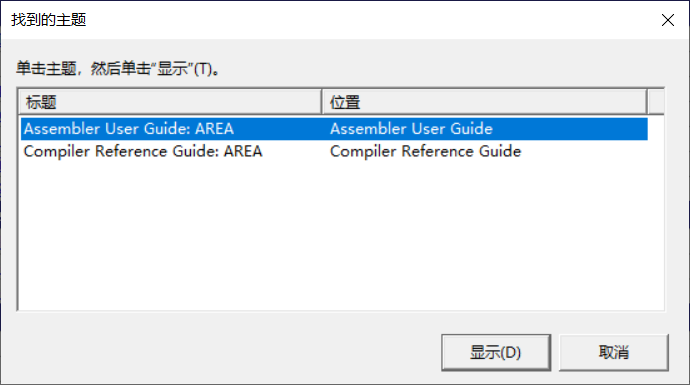
### 2.2.4 µVision联机资源使用

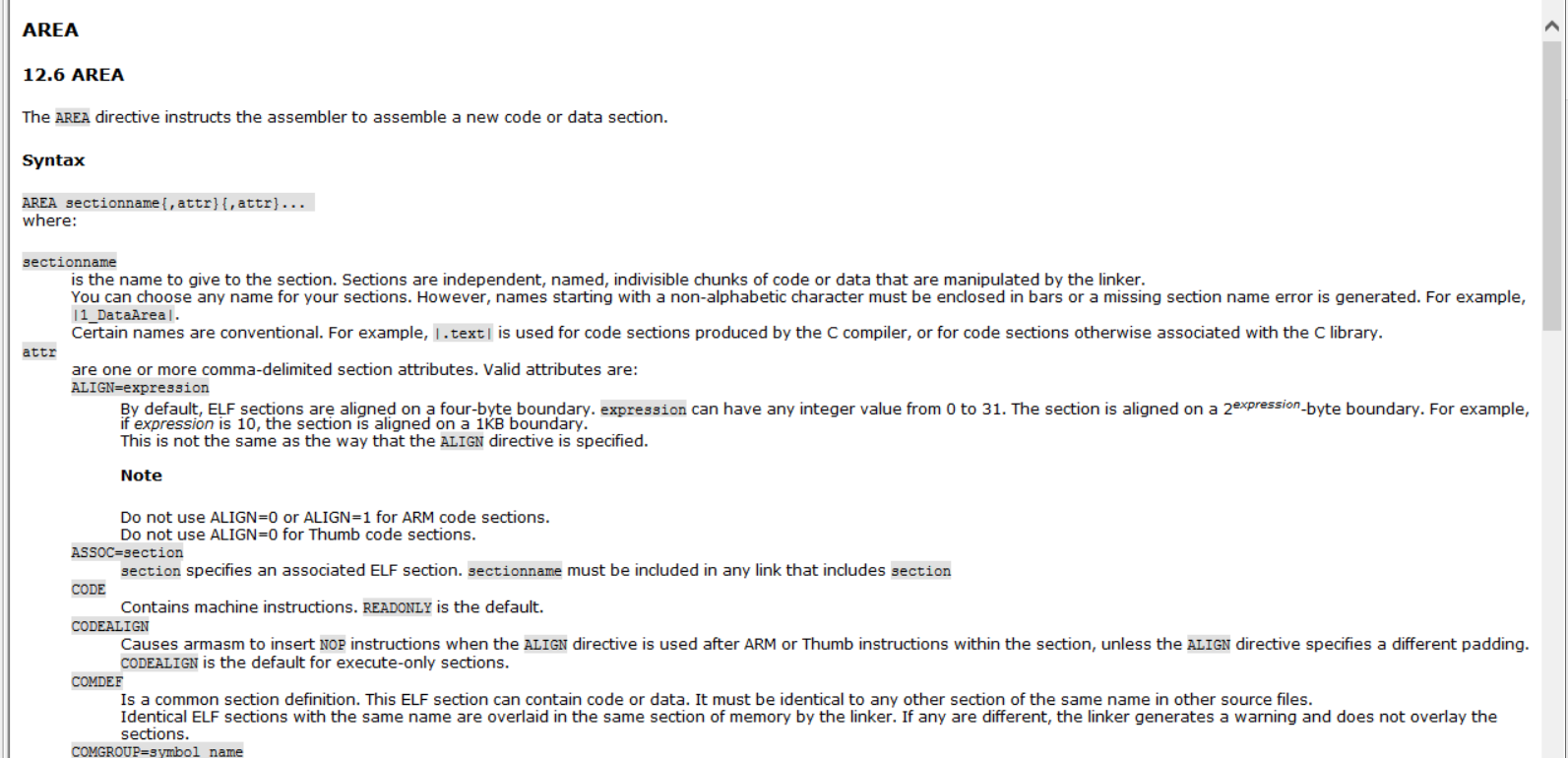
（21）打开联机帮助文档，搜索指示符“WEAK”的作用。



即：当该声明的标号在其他地方没有定义时，此处有效。

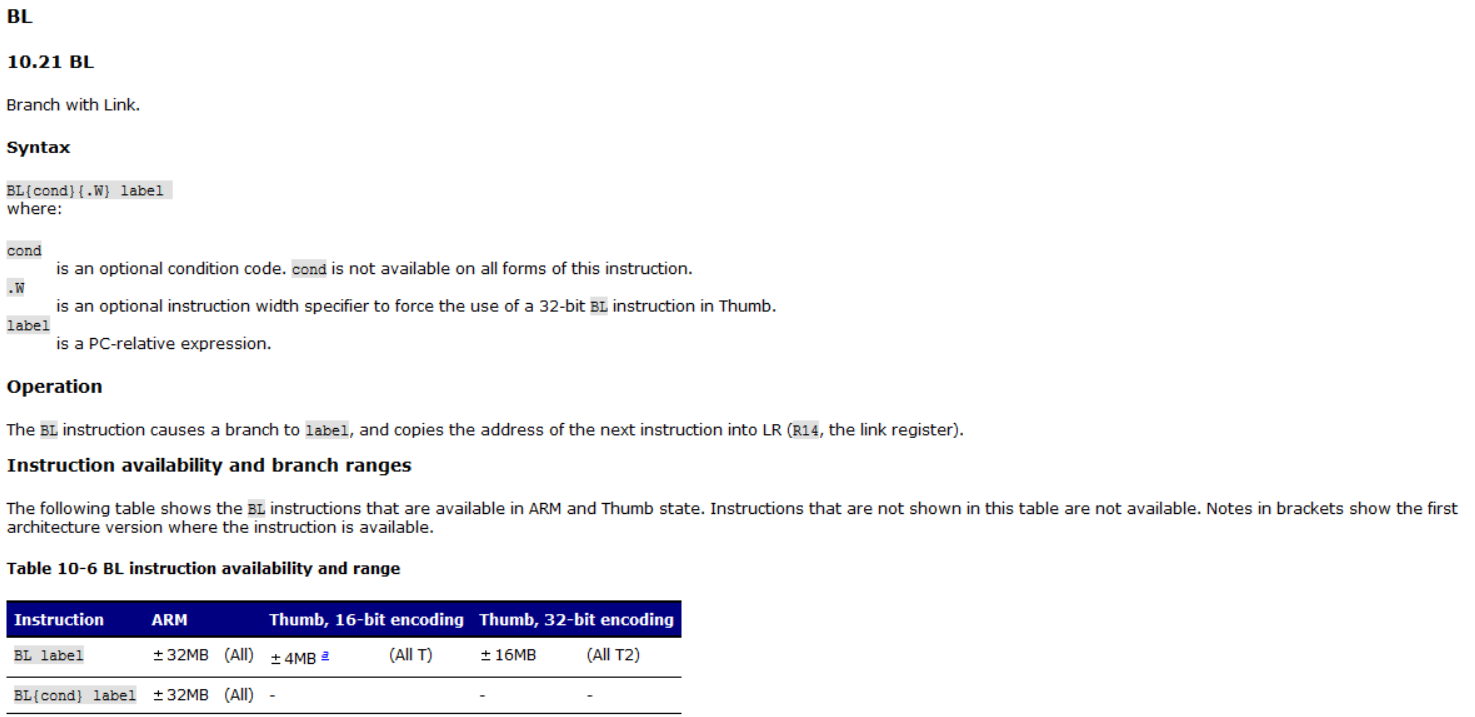
（22）将光标停留在某个指示符（伪指令关键字）上，如“AREA”，按键“F1”，查阅自动弹出的帮助信息。





即：用于声明一个代码段或数据段。

（23）将光标停留在某条指令上，如“BLX”，按键“F1”，查阅自动弹出的帮助信息。

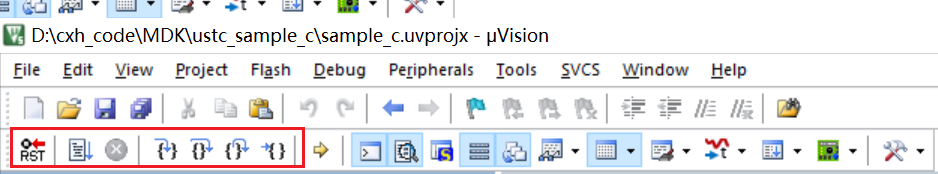


（24）了解包管理器“Pack installer”的基本功能。

用来安装、管理包以获得对相应芯片的支持。

### 2.2.5 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。



从左往右：

1 复位；

2 运行；

3 停止运行；

4 运行一行；

5 步越，如果当前语句包含函数调用，则执行完该函数并返回到下一条语句；

6 步出，执行至当前函数结束；

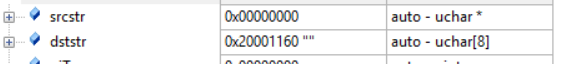
7 运行到光标所在行。

但由于不清楚keil中的调用是何种形式，5、6虽然能看到不是逐步进行的，但运行结束并非以C程序中的一行或一个函数为单位，使用起来并不方便。另一方面，这两个按钮在汇编文件中可以较好的发挥其应有功能，可能原因是其子函数的概念是汇编的子函数，而C语句中含有多个汇编子函数，因而不好使用。

（26）通过Watch窗口观察一下代码执行前后变量值。观察变量“srcstr”和“dststr”，同时观察Memory窗口对应地址的值。

从watch窗口上可以看到：

1、两者未定义时



2、经过定义后

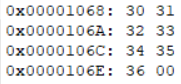
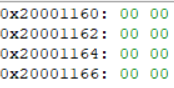


3、步进strcopy命令，dststr的字符逐个变为srcstr的字符

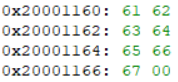


从Memory窗口来看：

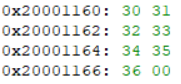
1、两者未定义时，0x00001068-0x0000106F中存储”0123456\0”，0x20001160-0x20001167中什么也没有。

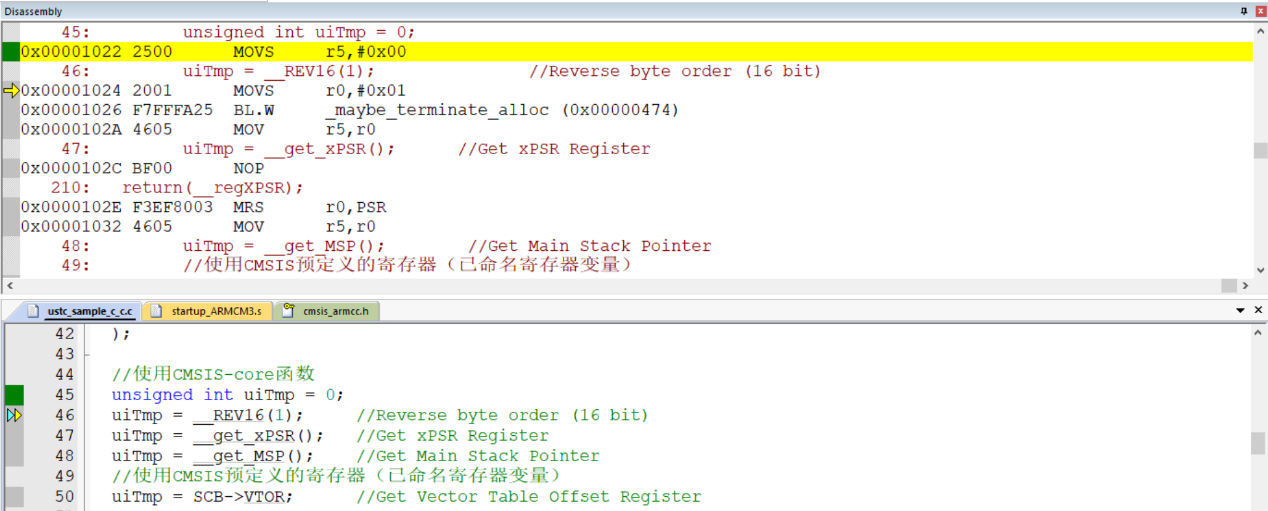
2、两者定义后，前者不变，后者变为“abcdefg\0”



3、strcopy后，前者不变，后者变为”0123456\0”。



（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：“汇编-C语言”方式记录

S 0x00001022 2500 MOVS r5,#0x00

C unsigned int uiTmp = 0;

S 0x00001024 2001 MOVS r0,#0x01

0x00001026 F7FFFA25 BL.W \_maybe\_terminate\_alloc (0x00000474)

0x0000102A 4605 MOV r5,r0

C uiTmp = \_\_REV16(1); //Reverse byte order (16 bit)

S 0x0000102E F3EF8003 MRS r0,PSR

0x00001032 4605 MOV r5,r0

C uiTmp = \_\_get\_xPSR(); //Get xPSR Register

S 0x00001036 F3EF8008 MRS r0,MSP

0x0000103A 4605 MOV r5,r0

C uiTmp = \_\_get\_MSP(); //Get Main Stack Pointer

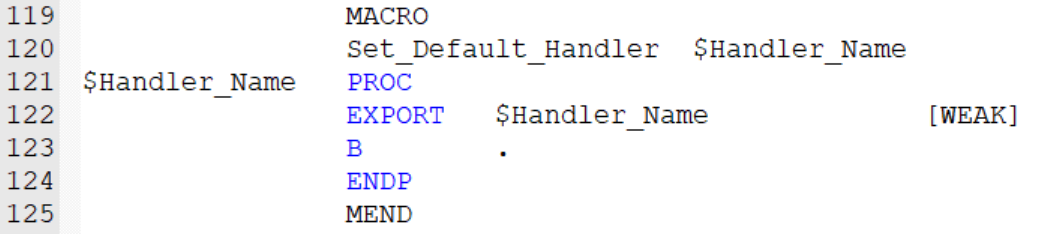
S 0x0000103C 480E LDR r0,[pc,#56] ; @0x00001078

0x0000103E 6805 LDR r5,[r0,#0x00]

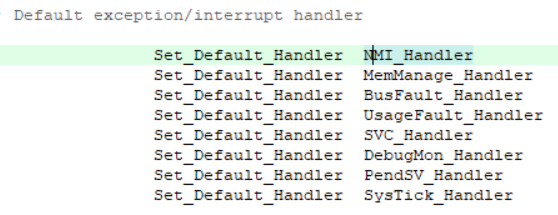
C uiTmp = SCB->VTOR; //Get Vector Table Offset Register

## 2.3 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：用于减小工作量，结合下方的语句可以快速定义默认ISR。



定义NMI\_Handler时：

NMI\_Handler PROC

EXPORT NMI\_Handler [WEAK]

B .

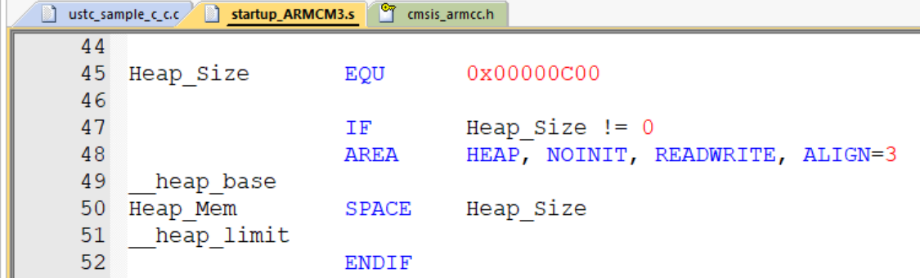
ENDP

（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）？



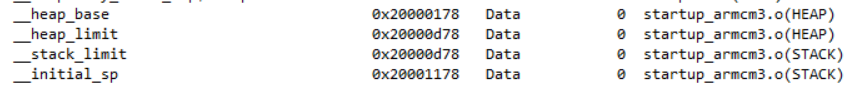
答：0x0000 1058 – 0x0000 1065（包括末尾的”\0”）

（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”？

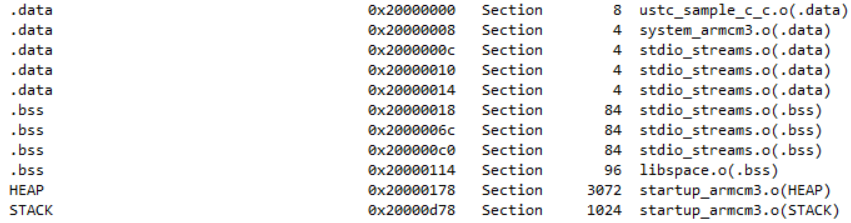


答：考虑首先SRAM从0x2000 0000起始，考虑栈的初始化将使0x2000 0000 - 0x2000 0400为栈的区域。而后堆的大小设为0xC00，则堆的末尾为0x2000 1000。但这么想解释不了。

查看输出文件\*.map发现，栈顶为0x20001178，即指针在此处。栈和堆的大小如定义，但之前有一些未知的数据。而调试发现，在正式运行程序之前，有几段初始化程序在使用栈，个人认为是正好初始化完成时，恰好数据指针在0x20001160，因此dststr变量被分配到该处。



（全局定义的字符）



（局部定义字符）

# 实验三 基于STM32库的GPIO与定时器

## 3.1 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

## 3.2 实验内容

### 3.2.1 下载ST公司STM32库及芯片手册

（31）下载ST公司关于STM32F103系列芯片的库文件

（32）下载ST公司关于STM32F103系列芯片的文档

STM32F10x参考手册

STM32F103ZET6数据手册

STM32F10x固件库用户手册

### 3.2.2 建立基于STM32库的Project

（33）建立基于STM32F103ZE的项目

（34）拷贝库文件

（35）添加库文件

### 3.2.3 配置Project的头文件目录、预编译参数、Simulator

（36）配置头文件路径

（37）配置预编译选项“Preprocessor Symbols”

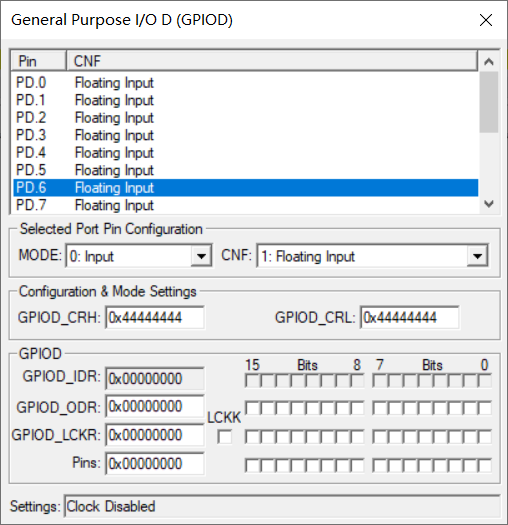
（38）设置采用Simulator方式进行调试，配置为STM32F103ZE芯片的simulator

（39）编译、链接

### 3.2.4 Debug，用外设仿真功能验证GPIO输入和输出

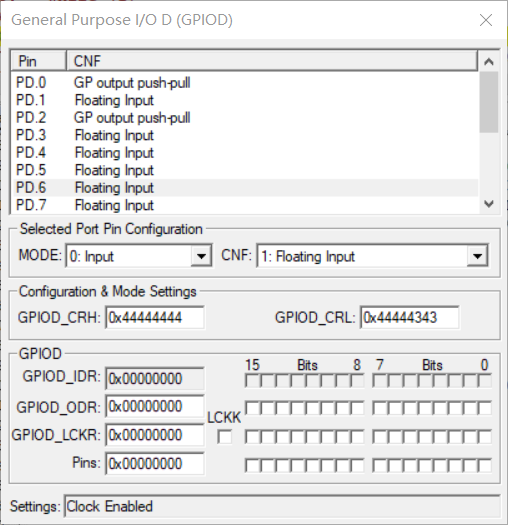
（40）打开GPIO窗口，观察执行前后GPIOD的变化。

1 未执行：

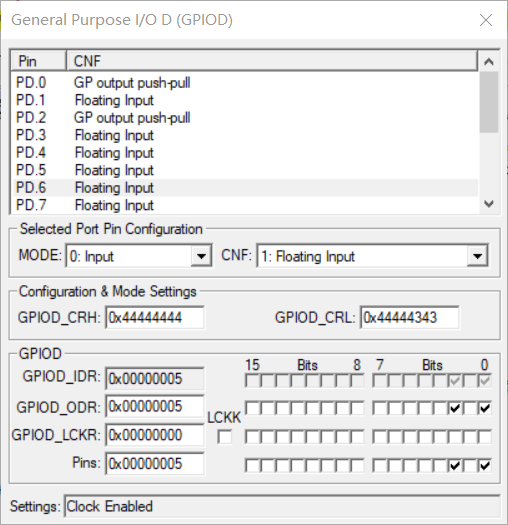


2 开启时钟：不变

3 设置管脚：



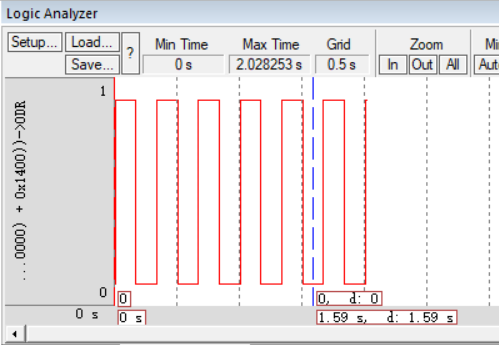
4 while循环：

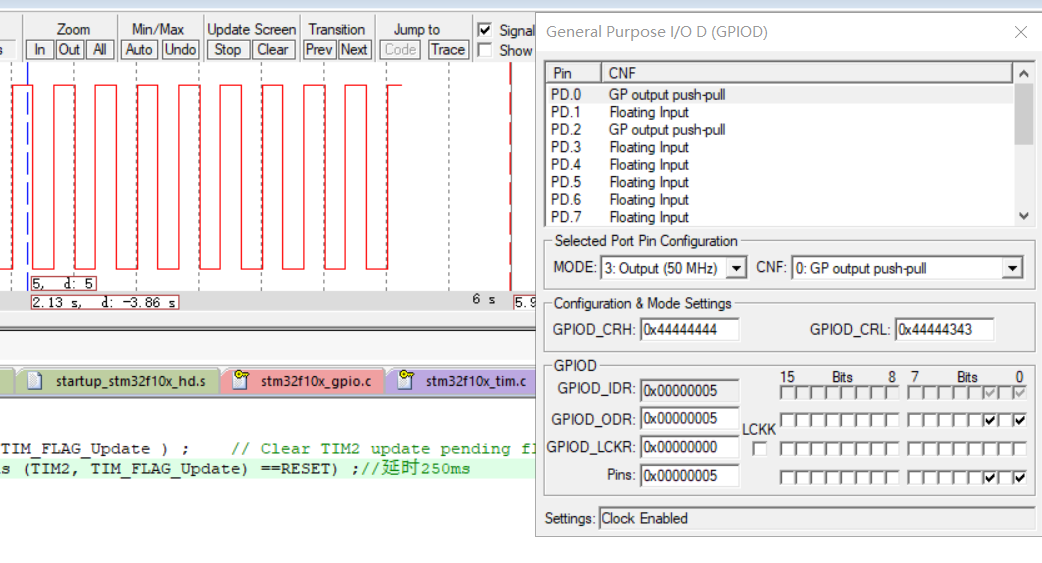


Pins、ODR的0、2位交替为0，1。

### 3.2.5 Debug，用Logic Analyzer观察GPIO输出

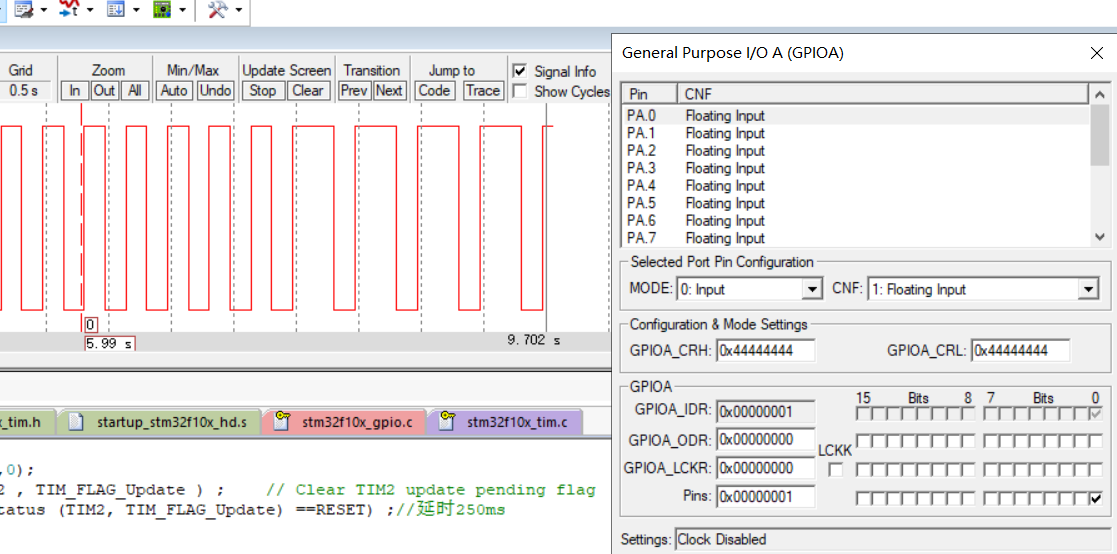
（41）通过µVision IDE自带的逻辑分析模块（Logic Analyzer）观察GPIOD的变化。





### 3.2.6 使用外设仿真和Logic Analyzer验证GPIO输入和输出

（42）打开GPIOA的窗口，通过在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。



与代码中

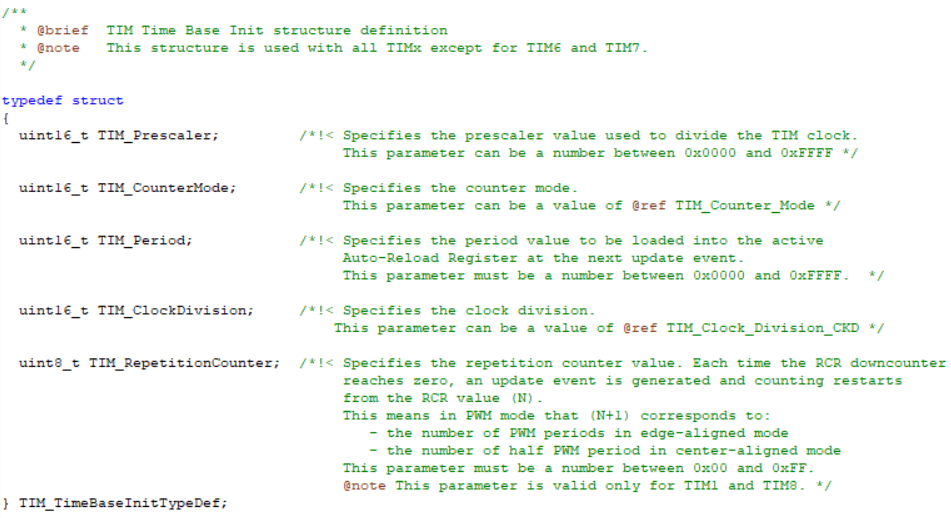
if ( GPIO\_ReadInputDataBit (GPIOA,GPIO\_Pin\_0)) //缺省IO端口被配置为输入

TIM2\_Delay250MS(); //如果PA0输入1，则添加额外的延时

所描述一直，增长了高电平时间。

### 3.2.7 配置定时器

（43）阅读TIM\_TimeBaseInitDef定义文件，了解结构体各成员含义。



TIM\_Prescaler：预分频，用于对TIM的时钟源分频，在0~65535取值。

TIM\_CounterMode：指示计数模式。

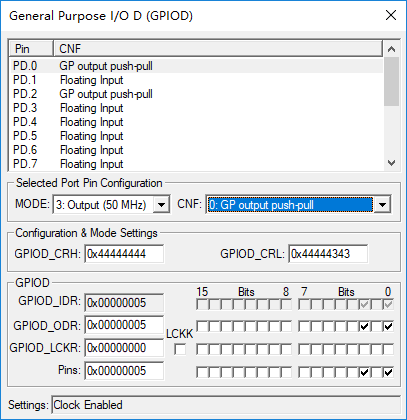
TIM\_Period：指示ARR中的数值。意味着计数到多少产生一个更新事件，在0~65535取值。

TIM\_ClockDivision：指示时钟分频。与TIM\_Prescaler不同，ClockDivision是对外部输入分频， 它不会影响TIM本身的时钟频率。

TIM\_RepetitionCounter：指示周期计数寄存器的值，该值只用于TIM1/8。每次RCR下计数归 零，则产生一个更新时间，并重载RCR。这意味着PWM模式中周期数量的变化。在0~255 取值。

## 3.3 思考题

（44）解释下图中“CRL、CRH、IDR、ODR、LCKR”几个寄存器的作用。



答：

CRL/ CRH：端口配置低/高寄存器。每四位为一组，控制1个引脚的输入/输出模式。

IRD：端口输入数据寄存器。低16位有效，其值为对应端口的状态（输入状态下），只能读 出。

ORD：端口输出数据寄存器。低16位有效，写的时候，其值影响对应端口的输出；读的时 候，其值表示对应端口上一次的输出。

LCKR：端口配置锁定寄存器。仅低17位有效。位16被正确设置时，低16位用来锁定端口 位的配置，当对相应的端口位执行了LOCK序列后，下次系统复位前不可再更改端口位 的配置（锁CRL/CRH）。

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

答：将GPIOD的0，2，7号管脚置位。

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

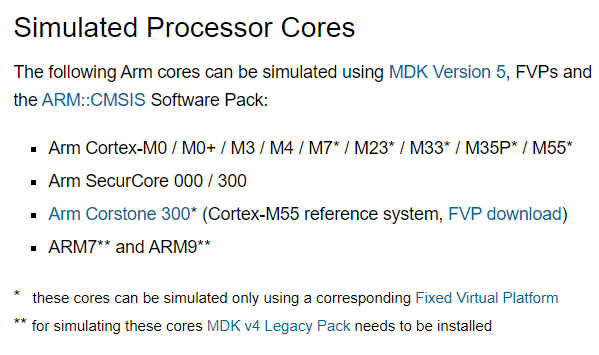
答：将GPIOD的7号管脚复位。

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

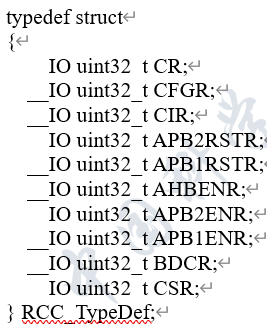
答：读出GPIOD\_Pin\_0的值（0/1），只在该管脚设置输入模式时有效。

（48）\*访问“http://www.keil.com/support/docs/3726.htm”，查询µVision IDE是否支持STM32F407ZG芯片的Simulation。

答：能。<http://www2.keil.com/mdk5/simulation/>



（49）\*阅读STM32F103芯片手册，了解“stm32f10x.h”文件中如下结构体各成员所对应寄存器的功能。



答：

CR：时钟控制寄存器。分别指示：

PLLRDY：PLL时钟是否就绪；

PLLON：PLL是否使能；

CSSON：时钟安全系统是否使能，置位以使用时钟检测器；

HSEBYP：外部高速时钟旁路：在调试模式下由软件修改以选用外部晶体振荡器；

HSERDY：外部高速时钟就绪标志，表示外部4-16MHz振荡器已稳定；

HSEON：外部高速时钟使能；

HSICAL：内部高速时钟校准，系统启动时自动初始化；

HSITRIM：内部高速时钟调整，软件调整内部HIS RC振荡器的频率；

HSIRDY：内部高速时钟就绪标志，用于表示内部8MHz振荡器是否稳定；

HSION：内部高速时钟使能。

CFGR：时钟配置寄存器。各位/位段的含义为：

MCO：微控制器时钟输出，用于选择输出的时钟源；

USBPRE：设置USB预分频；

PLLMUL：设置PLL倍频系数；

PLLXTPRE：设置HSE分频器作为PLL输入；

PLLSRC：设置PLL输入时钟源；

ADCPRE：设置ADC预分频；

PPRE2：设置高速APB预分频；

PPRE1：设置低速APB预分频；

HPRE：设置AHB预分频；

SWS：系统时钟切换状态，指示系统时钟源；

SW：系统时钟切换，设置系统时钟源。

CIR：时钟中断寄存器。各位/位段的含义为：

CSSC：清除时钟安全系统中断标志位；

PLLRDYC：清除PLL就绪中断标志位；

HSERDYC：清除HSE就绪中断标志位；

HSIRDYC：同上；

LSERDYC：同上；

LSIRDYC：同上；

PLLRDYIE：使能或关闭PLL就绪中断；

HSERDYIE：使能或关闭外部4-16MHz振荡器就绪中断；

HSIRDYIE：使能或关闭内部8MHz RC振荡器就绪中断；

LSERDYIE：使能或关闭外部32kHz RC振荡器就绪中断；

LSIRDYIE：使能或关闭内部40kHz RC振荡器就绪中断；

CSSF：时钟安全系统中断标志，指示外部4-16MHz振荡器时钟出现故障。

APB2RSTR：APB2外设复位寄存器，用于复位APB2总线上挂载的外设和功能。

APB1RSTR：APB1外设复位寄存器，用于复位APB1总线上挂载的外设和功能。

AHBENR：AHB外设时钟使能寄存器，设置挂载时钟使能或关闭。

APB2ENR：APB2外设时钟使能寄存器。

APB1ENR：APB1外设时钟使能寄存器。

BDCR：备份域控制寄存器。

CSR：控制/状态寄存器。用于控制备份域软件复位、RTC时钟的管理、外部低速时钟的管理。

# 实验四 基于STM32库的中断

## 4.1 实验目的

1、掌握EXTI中断配置流程

2、理解异常向量表

3、理解异常优先级配置

4、了解ST公司提供的TIM、NVIC相关库函数

5、掌握µVision IDE中外设仿真模块（NVIC）的使用

## 4.2 实验内容

### 4.2.1 建立基于STM32库的Project

（50）创建STM32F103ZE的工程。

（51）拷贝ST库文件

（52）添加库文件

### 4.2.2 配置Project的头文件目录、预编译参数、Simulator

（53）配置“include file”路径执行ST库头文件目录

（54）配置预编译选项“Preprocessor Symbols”为：USE\_STDPERIPH\_DRIVER

（55）设置采用Simulator方式进行调试，配置为STM32F103ZE芯片的simulator，具体配置同上一个实验

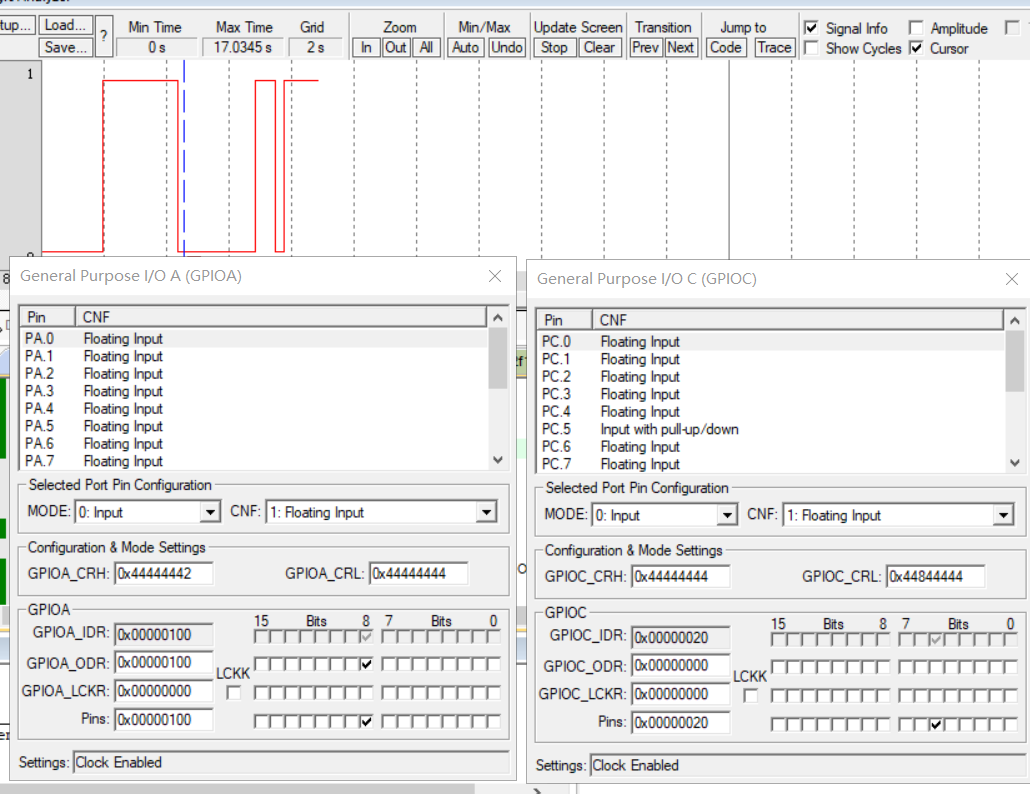
### 4.2.3 使用Logic Analyzer和外设仿真功能验证EXIT及GPIO输出

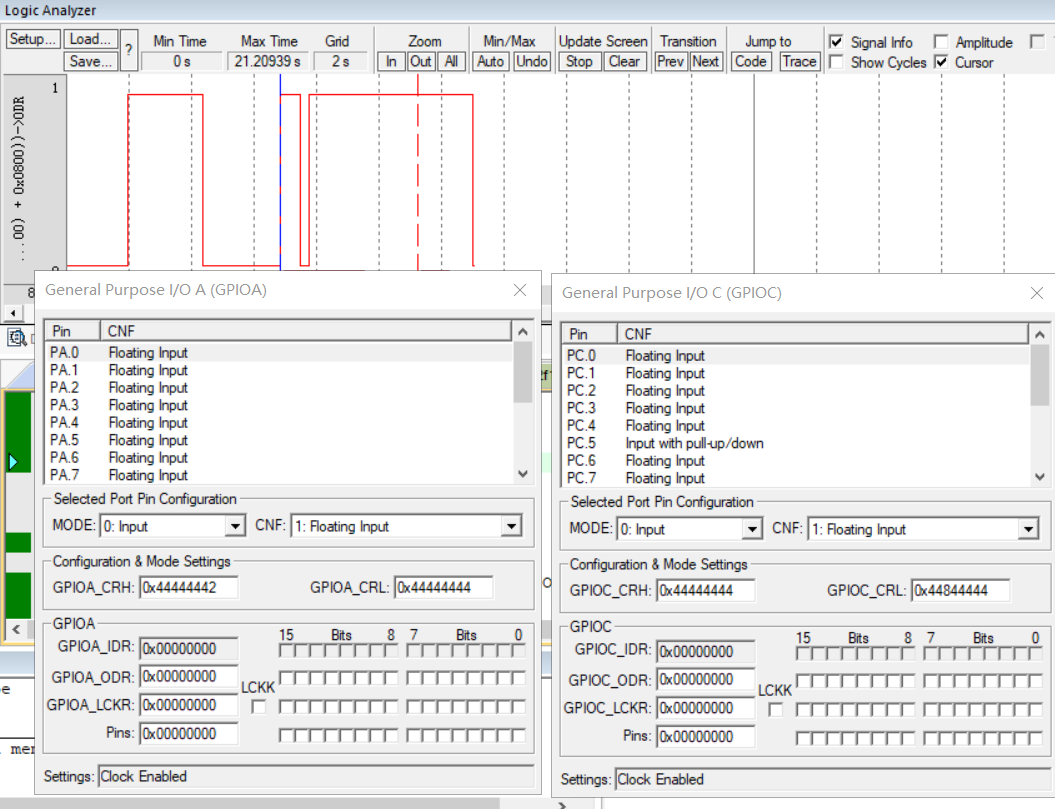
（56）用逻辑分析模块观察GPIOA

（57）用外设仿真模块观察GPIOA

（58）用外设仿真模块观察GPIOC

（59）在GPIOC窗口输入PC5作为中断信号，随PC5信号的变化，观察PA8的变化。

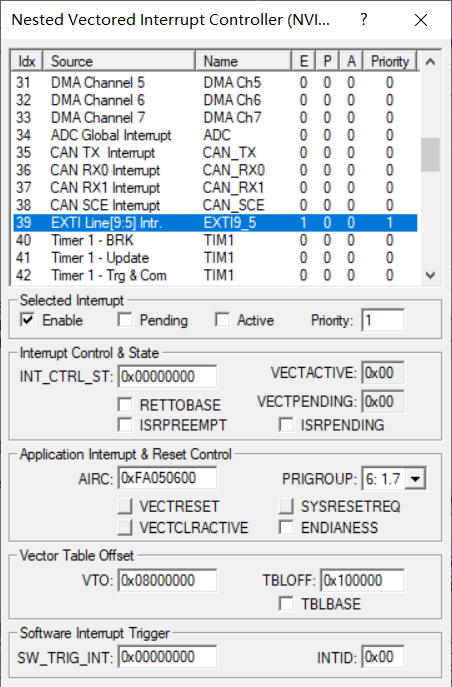




由于设置的是下降沿触发，因此PC5引脚输入由1变为零时，触发中断，实现PA8端口输出反转。

### 4.2.4 观察NVIC寄存器组

（60）观察外部中断#5有关的寄存器信息。



## 4.3 思考题

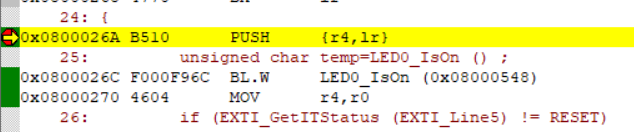
（61）为什么通过GPIOC的窗口（PeripheralsGeneral purpose I/OGPIOC）模拟PC5（中断信号）输入的时候，改变两次PC5后PA8才会发生变化？

答：因为设置的下降沿触发。

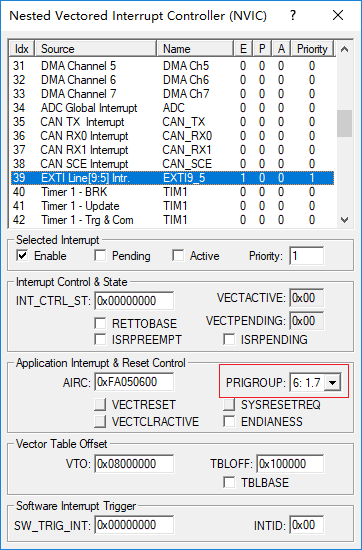


（62）请通过调试获得EXTI9\_5\_IRQHandler()的入口地址（应该是0x800026A），这个地址保存在在异常向量表什么位置？

答：入口地址为：0x0800026A； 保存在异常向量表的39号位。



（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：优先级分组为6，表示有1位抢占优先级，每组7位子优先级。在优先级配置寄存器AIRCR设置为8位宽度时，将有2个抢占优先级和每组128个子优先级。AIRCR和PRIGROUP的关系是两者共同影响优先级寄存器，AIRCR影响优先级寄存器的长度，PRIGROUP影响优先级寄存器每位的含义。

