电子科技大学信息与软件工程学院

**实 验 报 告**

学 号2020090910016

姓 名 瞿俊杰

（实验）课程名称《ARM处理器体系结构及

应用》课程实验

理论教师 兰 刚

实验教师 兰 刚

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：瞿俊杰 学号：2020090910016 指导教师：兰刚**

**实验地点：信软楼西305 实验时间： 2022.5.18**

1. 实验名称：数字滤波
2. 实验学时：4
3. 实验目的：
4. 通过工程中常用的数字滤波编程实现，验培养学生分析问题、解决问题能力。
5. 通过该编程，进一步巩固和强化学生ARM汇编编程的能。
6. 实验原理：

**（1）中值滤波**

数字图像在其形成、传输记录的过程中往往会受到很多噪声的的污染，比如：椒盐噪声、高斯噪声等，为了抑制和消除这些随即产生的噪声而改善图像的质量，就需要去、对图像进行去滤波噪处理。

中值滤波是图像平滑的一种方法 它是一种非线性平滑滤波技术，在一定条件下可以克服线性滤波带来的图像细节的模糊问题，特别是针对被椒盐噪声污染的图像。中值滤波对脉冲噪声有良好的滤除作用，特别是在滤除噪声的同时，能够保护信号的边缘，使之不被模糊。这些优良特性是线性滤波方法所不具有的。此外，中值滤波的算法比较简单，也易于用硬件实现。所以，中值滤波方法一经提出后，便在数字信号处理领得到重要的应用。

中值滤波方法：对一个数字信号序列xj(-∞<j<∞)进行滤波处理时，首先要定义一个长度为奇数的L长窗口，L=2N+1，N为正整数。设在某一个时刻，窗口内的信号样本为x(i-N)，…，x(i)，…，x(i+N)，其中x(i)为位于窗口中心的信号样本值。对这L个信号样本值按从小到大的顺序排列后，其中值，在i处的样值，便定义为中值滤波的输出值。

**（2）均值滤波**

均值滤波也用于图像处理，均值滤波也称为线性滤波，其采用的主要方法为邻域平均法。线性滤波的基本原理是用均值代替原图像中的各个像素值，即对待处理的当前像素点（x，y），选择一个模板，该模板由其近邻的若干像素组成，求模板中所有像素的均值，再把该均值赋予当前像素点（x，y），作为处理后图像在该点上的灰度g（x，y），即g（x，y）=∑f（x，y）/m m为该模板中包含当前像素在内的像素总个数。

本事实验要求，将 N 个无符号数进行排序，去掉最大值和最小值，剩余的数求平均值，平均值即为本次滤波所得的结果。

**（3）中值滤波程序流程图**

中值滤波的流程如图1所示：

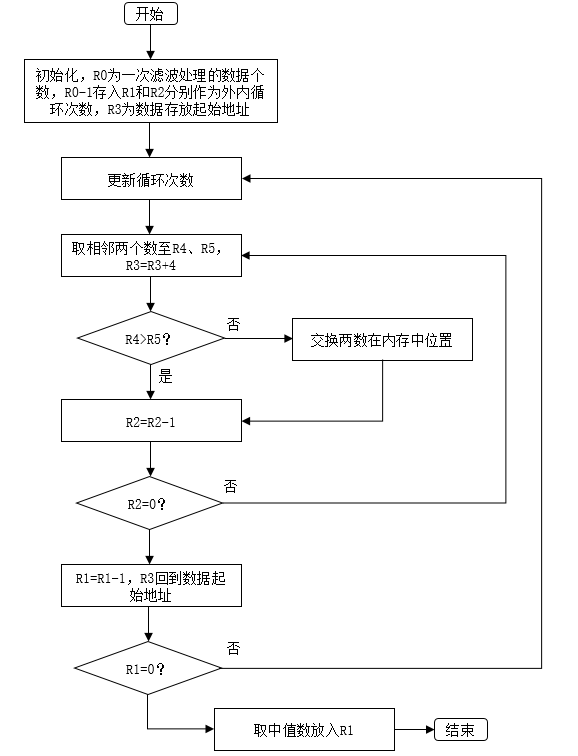


图1 中值滤波流程图

**（4）均值滤波程序流程图**

均值滤波流程图如图2所示：

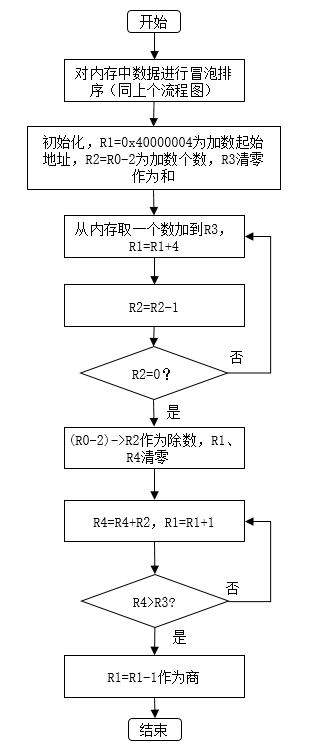


图2 均值滤波流程图

1. 实验内容：
2. 中值滤波及编程实验，要求如下：

* 关于 N（N为奇数）个数的值在程序中能任意、方便设置，并且放在 R0 中；
* 原始数据放在内存 0X40000000 开始的地址空间；
* 中值滤波的结果放在寄存器 R1 中；

1. 均值滤波编程实验，要求如下：

* 关于 N（N为偶数）个数的值在程序中能任意、方便设置，如 4、6、10、18 等，该值放在 R0 中；
* 原始数据放在内存中 0X40000000 开始的地址空间；
* 均值滤波的结果放在寄存器 R1 中；

1. 实验器材（设备、元器件）：
2. PC机一台；
3. Keil MDK-ARM uVision4开发工具。
4. 实验步骤：
5. 打开Keil MDK-ARM uVision4开发工具；
6. 新建一个工程文件；
7. 在新建的工程文件中，添加新的源程序文件
8. 编写代码
9. 选择“Build target”菜单对编写好的工程文件进行编译链接。
10. 点击““Start/Stop Debug Section””按键，对程序进行跟踪调试，在调试界面，单步执行，对CPU各寄存器的值的变化、以及相关内存的变化进行分析比较，判断程序的执行是否符合预期要求。
11. 实验结果与分析（含重要数据结果分析或核心代码流程分析）
12. **中值滤波**
13. **实验代码**

代码1中值滤波程序代码

|  |
| --- |
| AREA TEST, CODE, READONLY  ENTRY  num EQU 9 ;奇数  LDR R0, =num  SUB R1,R0,#1 ;外循环次数  MOV R2,R1 ;内循环次数  MOV R3,#0X40000000 ;循环起始地址  LOOP3  MOV R2,R1 ;更新循环次数  LOOP1  LDR R4,[R3] ;取数并比较  ADD R3,R3,#4  LDR R5,[R3]  CMP R4,R5  BHS LOOP2 ;若前小于后则交换内存中位置  STR R5,[R3,#-4]  STR R4,[R3]  LOOP2  SUBS R2,R2,#1 ;内循环次数减一  BNE LOOP1 ;一次外循环结束则不跳转  MOV R3,#0X40000000 ;从头开始下一次循环  SUBS R1,R1,#1 ;外循环次数减一  BNE LOOP3 ;外循环全部结束则不跳转    MOV R2,R0 ;取中值处理  SUB R2,R2,#1  MOV R2,R2,LSR #1  MOV R4,#4  MUL R5,R4,R2  ADD R3,R3,R5  LDR R1,[R3]  END |

1. **运行过程及结果界面截图**

下图3是程序运行刚开始的界面。R0设置为9，总共有9个单元。手动修改从0x40000000开始的存储单元，使其初始化为：0x11、0x22、0x33、0x44、0x55、0x66、0x77、0x88、0x99。

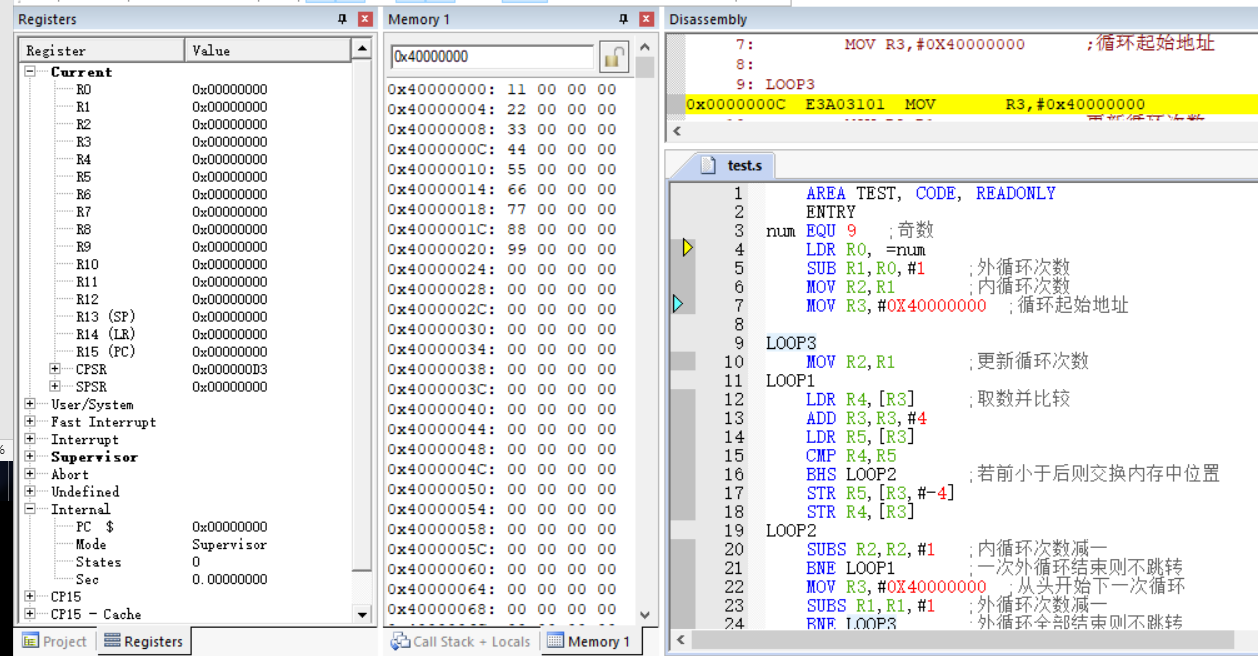


图3 中值滤波程序刚开始运行时的界面截图

下图4是程序运行结束时的界面，可以看出从0x40000000开始的9个存储单元被排序了，运行结果存放于R1中，结果为0x55。

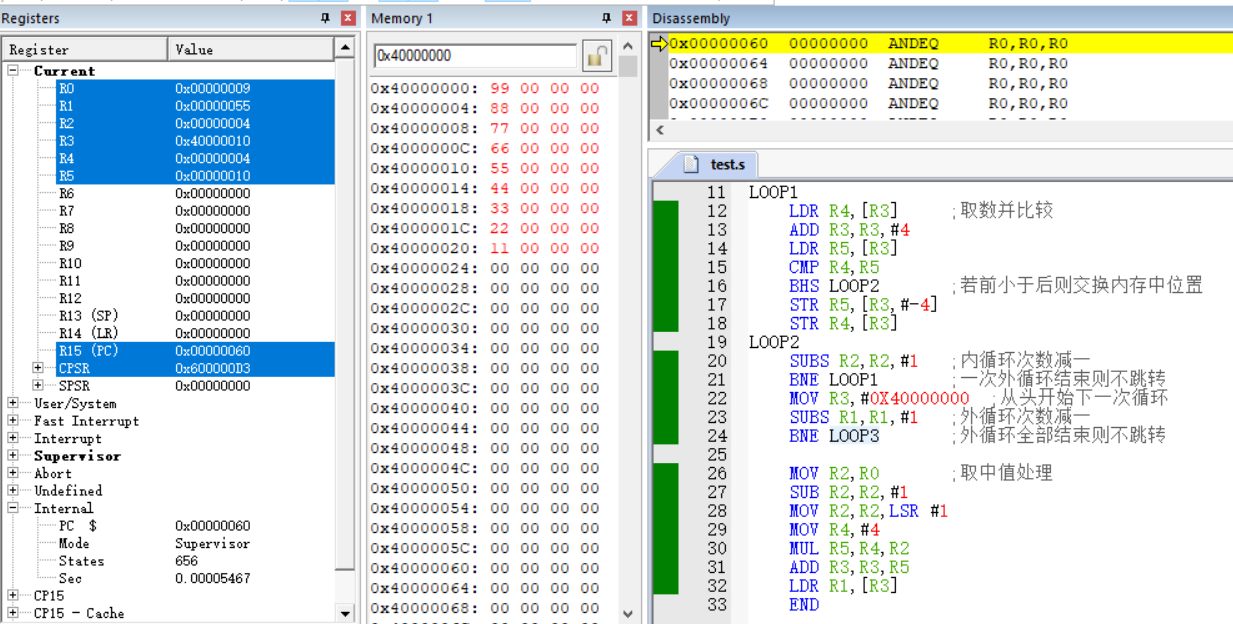


图4 中值滤波程序运行结束时的界面截图

1. **实验结论**

0x11、0x22、0x33、0x44、0x55、0x66、0x77、0x88、0x99这九个数，其中值为0x55，程序运行结果符合预期要求。

**2、均值滤波**

1. **实验代码**

代码2 均值滤波程序代码

|  |
| --- |
| PRESERVE8  AREA TEST,CODE,READONLY  ENTRY  num EQU 8;偶数  LDR R0,=num    SUB R1,R0,#1 ;外循环次数  MOV R2,R1 ;内循环次数  MOV R3,#0X40000000 ;循环起始地址  LOOP3  MOV R2,R1 ;更新循环次数  LOOP1  LDR R4,[R3] ;取数并比较  ADD R3,R3,#4  LDR R5,[R3]  CMP R4,R5  BHS LOOP2 ;若前小于后则交换内存中位置  STR R5,[R3,#-4]  STR R4,[R3]  LOOP2  SUBS R2,R2,#1 ;内循环次数减一  BNE LOOP1 ;一次外循环结束则不跳转  MOV R3,#0X40000000 ;从头开始下一次循环  SUBS R1,R1,#1 ;外循环次数减一  BNE LOOP3 ;外循环全部结束则不跳转      MOV R1,#0X40000004  SUB R2,R0,#2 ;循环加次数  AND R3,#0 ;R3存放和  LOOP4  LDR R4,[R1],#4 ;取一个数  ADD R3,R3,R4 ;加到R3  SUBS R2,R2,#1  BNE LOOP4    SUB R2,R0,#2 ;R2作为除数  AND R4,#0  AND R1,#0  LOOP5  ADD R4,R4,R2 ;R4不断加除数直到超过被除数  ADD R1,#1  CMP R4,R3  BLS LOOP5  SUB R1,R1,#1 ;R1为向下取整值，即商  END |

1. **运行过程及结果界面截图**

下图5是程序运行刚开始的界面。R0设置为8，总共有8个单元。手动修改从0x40000000开始的存储单元，使其初始化为：0x01，0x02，0x03，0x03，0x04，0x05，0x00，0x06。

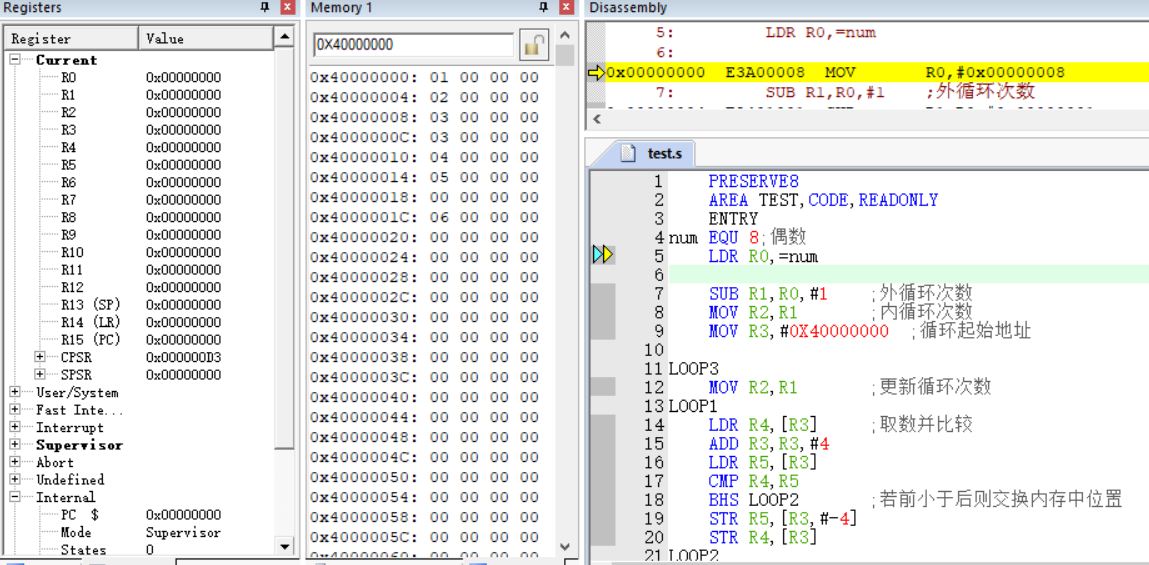


图5 均值滤波程序刚开始运行时的界面截图

下图6是程序运行结束时的界面，可以看出从0x40000000开始的8个存储单元被排序了，运行结果存放于R1中，结果为3。

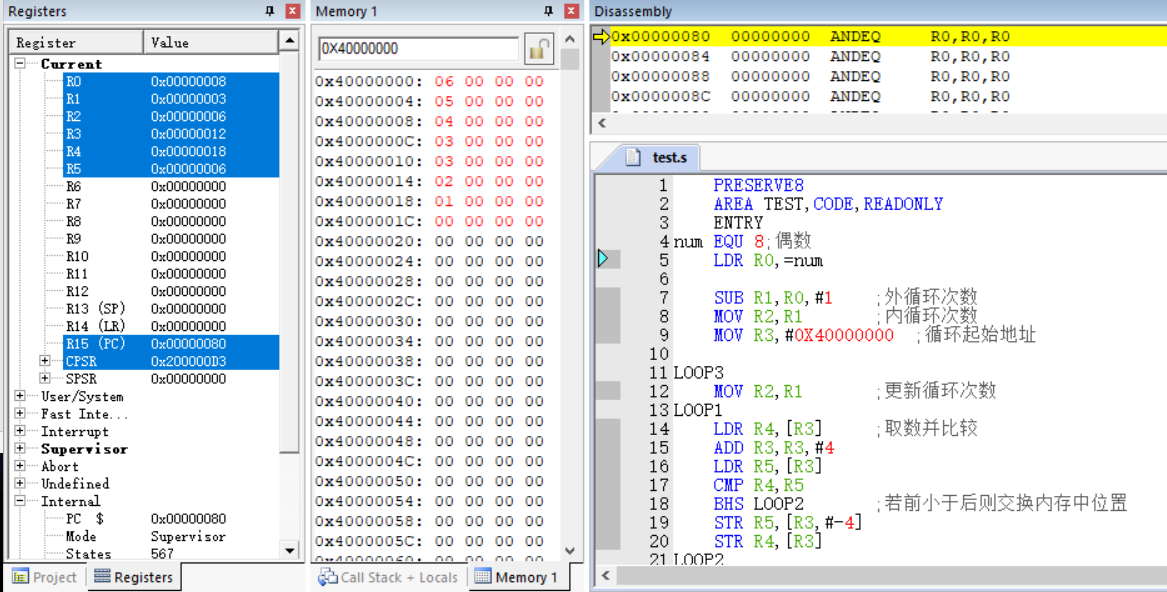


图6 均值滤波程序运行结束时的界面截图

1. **实验结论**

0x01，0x02，0x03，0x03，0x04，0x05，0x00，0x06这8个数，去掉最大值6和最小值0后，其平均值为3，程序运行结果符合预期要求。

1. 总结及心得体会：

本次实验用到的算法很简单，仅是冒泡排序，但是相比于C语言复杂很多，汇编指令能够执行的任务有限，以至于需要小心谨慎排查代码逻辑。另外，我还学习了均值滤波与中值滤波在实际生活中的应用，收获颇丰。

1. 对本实验过程及方法、手段的改进建议：

本实验我使用不断累加并比较大小来实现除法，当计算量较大时这种方法效率较低且耗时久，可以考虑更换算法进行优化。

报告评分：

指导教师签字：

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：瞿俊杰 学号：2020090910016 指导教师：兰刚**

**实验地点：信软楼西305 实验时间：2022.5.25**

1. 实验名称：流水灯仿真实验
2. 实验学时：4
3. 实验目的：
4. 掌握ARM处理器的输入输出接口。
5. 掌握通过MDK提供的仿真功能，实现系统的仿真运行。
6. 通过该编程实验，进一步巩固和强化学生ARM汇编编程的能，ARM应用程序框架，培养学生实际应用的能力。
7. 实验原理：

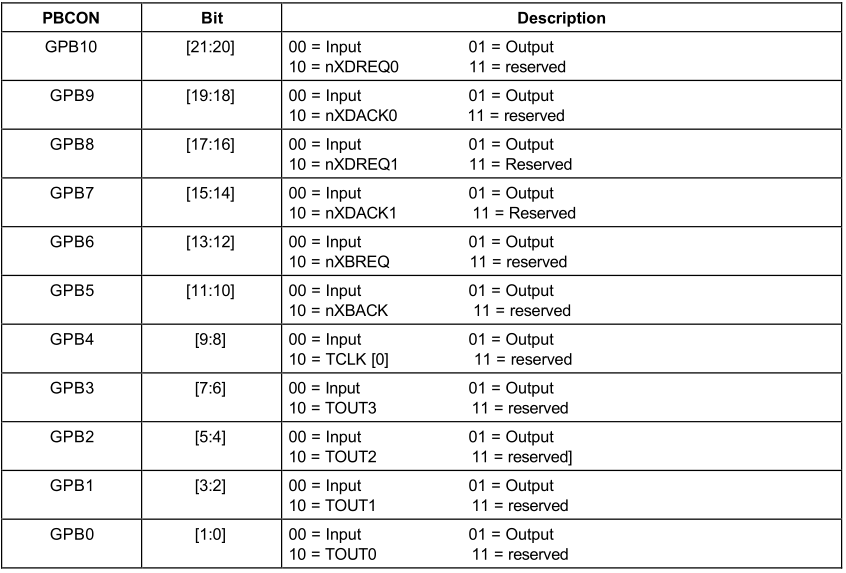
**1、端口B相关寄存器说明**

1. 端口B对应的寄存器

端口B 控制寄存器包括：端口B配置寄存器GPBCON、数据寄存器GPBDAT和上拉寄存器GPBUP，具体如下所示：

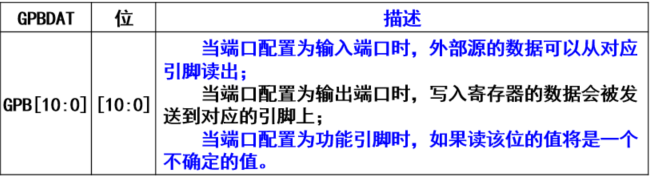


1. GPBCON各位的描述如下：

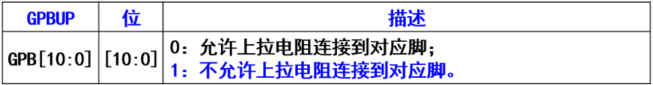


1. 端口B数据寄存器GPBDAT中各位的描述如下：

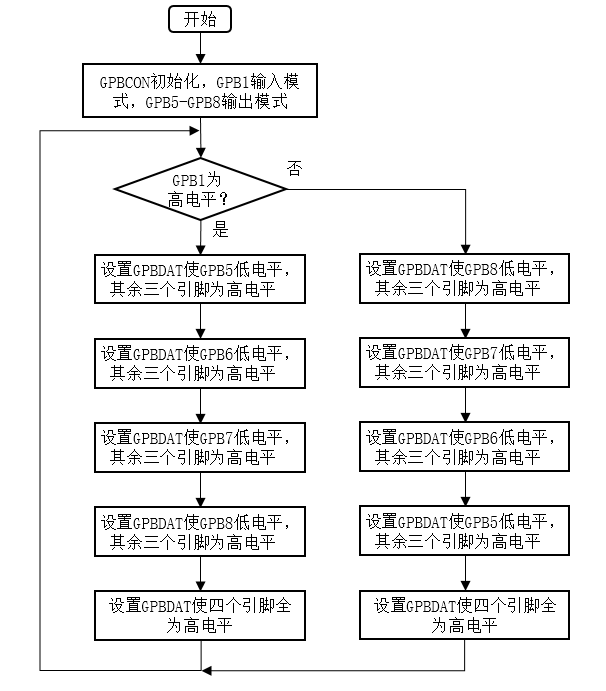
GPBDAT为准备输出或输入的数据，其值为11位[10:0]。



1. 端口B上拉寄存器GPBUP中各位的描述如下：



**2. 实验流程图如下：**



1. 实验内容：

按下面电路图，编写一个流水灯程序，并通过MDK的仿真功能进行验证。



实验要求：

1. 有1个拨码开关K1(接GPB1端口)作为输入；
2. 有4个指示灯作为输出(接GPB5-GPB5端口)；
3. 拨码开关K1输入高电平时，指示灯从上到下（LED1到LED4）循环显示，每次只有一个灯亮；
4. 拨码开关K1输入低电平时，指示灯从下到上（LED4到LED1）循环显示，每次只有一个灯亮；
5. 要求每个灯点亮持续的时间是20ms-30ms之间。
6. 使用C语言编写程序，给完整程序并加注释。
7. 通过MDK的仿真功能验证程序的正确性。
8. 实验器材（设备、元器件）：
9. PC机一台；
10. Keil MDK-ARM uVision5开发工具。
11. 实验步骤：
12. 打开Keil MDK-ARM uVision5开发工具；
13. 新建一个工程文件；
14. 在新建的工程文件中，添加新的源程序文件
15. 编写代码
16. 选择“Build target”菜单对编写好的工程文件进行编译链接。
17. 点击““Start/Stop Debug Section””按键，对程序进行跟踪调试，在调试界面，单步执行，对CPU各寄存器的值的变化、以及相关内存的变化进行分析比较，判断程序的执行是否符合预期要求。在程序调试期间，利用MDK的“Logic Analyzer”对输出波形进行仿真，以验证程序执行是否正确。仿真的步骤如下：
18. 在调试界面中，选择“Analysis Windows”→“Logic Analyzer”，进入“Logic Analyzer”界面后，点击“Setup”按键，设置需要观察的输出端口。
19. 进入“Setup Logic Analyzer”窗口后，就可以对需要观察的输出端口引脚信号进行设置。由于本实验主要涉及到GPB5-8共4个输出端口的输出信号进行观察，因此需要添加这4个信号。
20. 设置好相关引脚信号后，就可以进行回到“Debug”界面，调试运行。
21. 先测试拨码开关K1（GPB1）输入为高电平的时候情况，指示灯从上到下（LED1到LED4）循环显示的输出情况。
22. 然后选择“Peripherals”→“I/O Ports”→“PortB(GPB)”，设置拨码开关K1（GPB1）输入为低电平。
23. 再次回到“Debug”界面，继续调试运行，测试拨码开关K1（GPB1）输入为低电平的时候情况，指示灯从下到上（LED4到LED1）循环显示的输出情况。
24. 最后得到完整的仿真输出波形。
25. 实验结果与分析（含重要数据结果分析或核心代码流程分析）
26. **实验代码**

代码1 main.c实验代码

|  |
| --- |
| #include "s3c2440.h"  #define U32 unsigned int  U32 GPB = 0x00; //记录GPBDAT的值  int main(){  int i=80000;    GPBCON &=~0X3FC03; //GPB0、GPB5-GPB8位清零  GPBCON |= 0X15400; //GPB1输入，GPB5-8输出  while(1){  if(GPBDAT & 0X2){ //如果GPB1高电平  GPB = GPBDAT;  GPB |= 0X1E0; //灯全灭  GPB &=~ 0X20; //LED1亮  GPBDAT =GPB;  while(i--);  i=80000;  GPB |= 0X1E0; //灯全灭  GPB &=~ 0X40; //LED2亮  GPBDAT =GPB;  while(i--);  i=80000;  GPB |= 0X1E0; //灯全灭  GPB &=~ 0X80; //LED3亮  GPBDAT =GPB;  while(i--);  i=80000;  GPB |= 0X1E0; //灯全灭  GPB &=~ 0X100; //LED4亮  GPBDAT =GPB;  while(i--);  i=80000;  GPB |= 0X1E0; //灯全灭  GPBDAT =GPB;  while(i--);  i=80000;  }  else{ //如果GPB1低电平  GPB = GPBDAT;  GPB |= 0X1E0; //灯全灭  GPB &=~ 0X100; //LED4亮  GPBDAT =GPB;  while(i--);  i=80000;  GPB |= 0X1E0; //灯全灭  GPB &=~ 0X80; //LED3亮  GPBDAT =GPB;  while(i--);  i=80000;  GPB |= 0X1E0; //灯全灭  GPB &=~ 0X40; //LED2亮  GPBDAT =GPB;  while(i--);  i=80000;  GPB |= 0X1E0; //灯全灭  GPB &=~ 0X20; //LED1亮  GPBDAT =GPB;  while(i--);  i=80000;  GPB |= 0X1E0; //灯全灭  GPBDAT =GPB;  while(i--);  i=80000;  }  }  } |

1. **运行过程及结果界面截图**

设置完成输入设置后，“Logic Analyzer”窗口显示如下，里面多了4个引脚的显示。

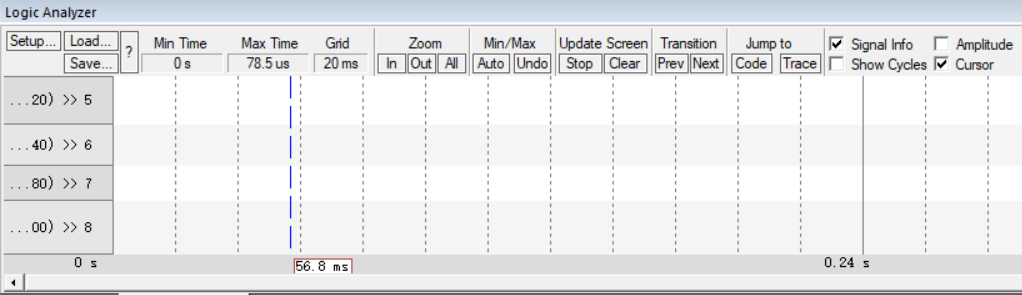


图1 设置完成输入设置后“Logic Analyzer”窗口显示

图2是拨码开关K1（GPB1）输入为高电平的时候情况，指示灯从上到下（LED1到LED4）循环显示。从图中可以看出，每个灯点亮的时间约为26.8ms

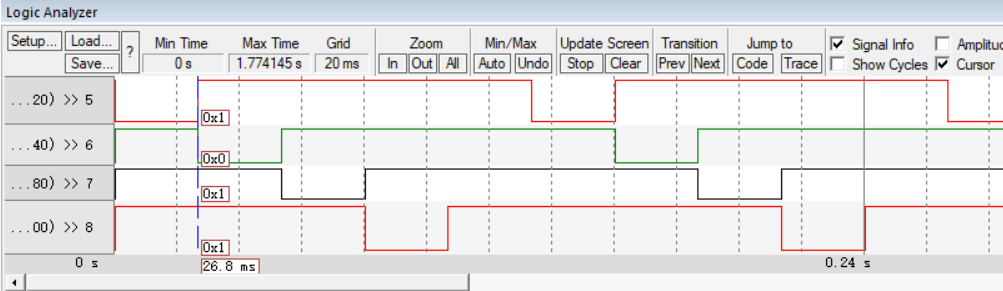


图2 拨码开关K1（GPB1）输入为高电平时的输出波形

完整的仿真输出波形如图3所示，图中后半部分就是GPB1为低电平时，指示灯从下到上（LED4到LED1）循环显示。

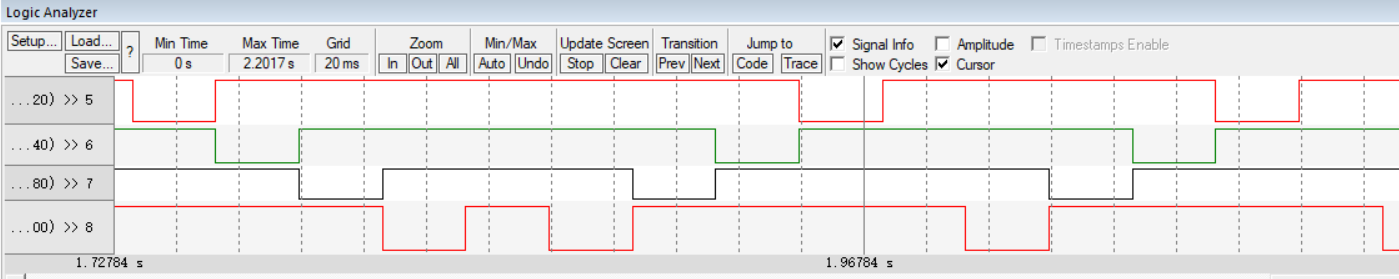


图3 完整的仿真输出波形

1. **实验结论**

通过仿真测试，可以看出，GPB5-8这4个端口，在GPB1为高电平时，轮流输出低电平，使得指示灯从上到下（LED1到LED4）循环显示；在GPB1为低电平时，轮流输出低电平，使得指示灯从下到上（LED4到LED1）循环显示，每个灯点亮的时间约为26.8ms，符合实验要求。

1. 总结及心得体会：

通过本次实验我学会了通过修改寄存器来操作外设，并实现了简单的流水灯例程，大型工程的最底层都是对寄存器的操作，一层层封装成了我们常见的函数，令人感慨。

1. 对本实验过程及方法、手段的改进建议：

没有库函数的情况下，点亮灯的时间无法确定，只能用while实现粗延时功能，建议加入delay库函数以便于控制灯的点亮时间。

报告评分：

指导教师签字：

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：瞿俊杰 学号：2020090910016 指导教师：兰刚**

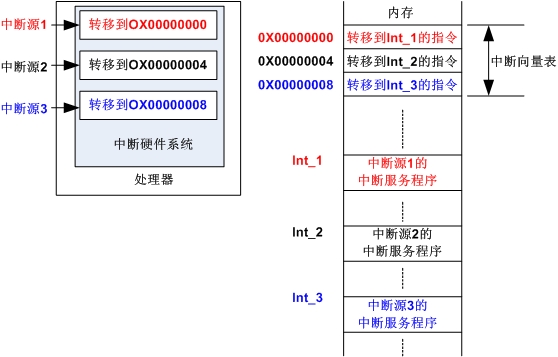
**实验地点：信软楼西305 实验时间：2022.6.1**

1. 实验名称：基于中断的按键处理程序实验
2. 实验学时：4
3. 实验目的：
4. 掌握ARM处理器的中断处理过程。
5. 掌握ARM处理器中断服务程序的编写方法。
6. 通过该编程实验，进一步巩固和强化学生ARM汇编编程的能，ARM应用程序框架，培养学生实际应用的能力。
7. 实验原理：
8. **中断及中断向量**

为了与普通子程序的首地址进行区分，中断服务程序的首地址（入口地址）通常被叫做中断向量，或中断矢量。

在处理器收到中断请求之后，它们都需要获得中断服务程序首地址——中断向量。所有的中断向量都按一定规律存放在一个固定的存储区域，这个集中存放了中断向量或与中断向量相关信息的存储区域就叫做中断向量表。

S3C2440的中断向量安排如下：



1. **ARM的中断（异常）向量表**

ARM中断（异常）的各个向量在向量表中的分配如下：



1. **S3C2440X中断控制器**

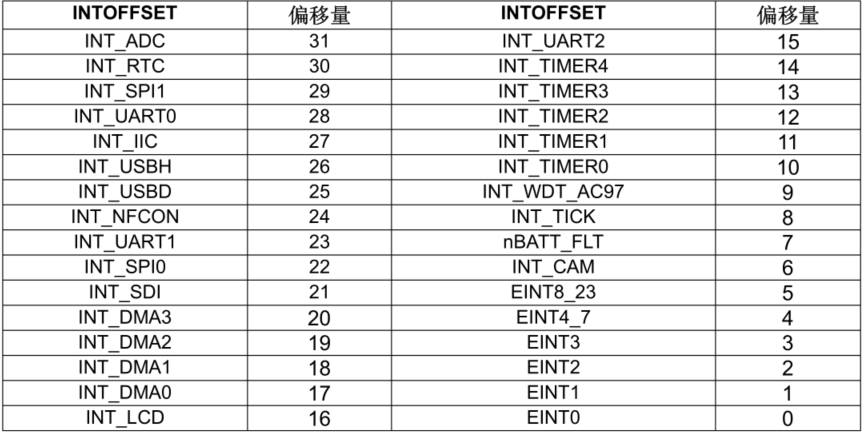
S3C2440X中断控制器有60个中断源，对外提供24个外中断输入引脚，内部所有设备都有中断请求信号，例如：DMA控制器、UART、IIC等。中断控制器中包括5个控制寄存器：源挂起寄存器、中断模式寄存器、屏蔽寄存器、优先级寄存器、中断挂起寄存器。

1. 中断屏蔽寄存器INTMSK

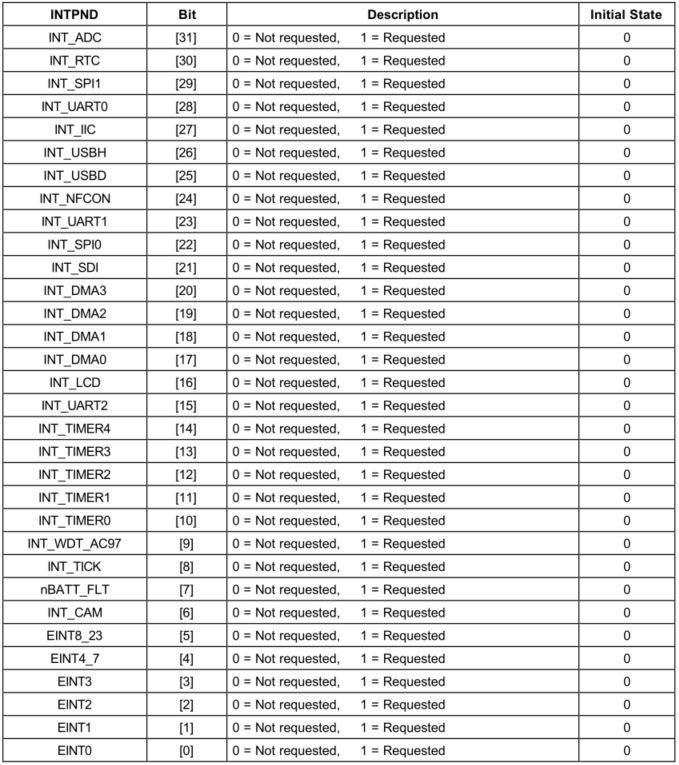
中断屏蔽寄存器INTMSK包括了32位，每一个比特位均与相应的一个中断源相对应。如果某位被设置为1，那么CPU不会执行相应中断源提出的中断请求（即使源挂起寄存器的相应位被设置为1）。如果屏蔽位为0，那么中断请求会被正常执行。

1. INTOFFSET寄存器

INTOFFSET寄存器的值说明了INTPND寄存器中哪一个IRQ模式的中断请求有效。这个比特位可以通过清除SRCPND和 INTPND寄存器来自动清除。



1. 中断挂起寄存器 INTPND（C语言程序中使用rINTPND进行访问）



1. 外部中断挂起寄存器EINTPEND（C语言程序中使用rEINTPEND进行访问）



1. **S3C2440X外中断相关控制器**

在具体执行中断之前，要初始化好要用的中断。2440的外部中断引脚EINT与通用IO引脚F和G复用（本实验用的EINT8-23使用的是PORTG对应的引脚），要想使用中断功能，就要把相应的引脚配置成中断模式，如我们想把端口G0设置成外部中断（EINT8），而其他引脚功能不变，则使用以下语句

rGPGCON = rGPGCON & (~(0x03)) | (0x02)。

配置完引脚后，还需要配置具体的中断功能。我们要打开某一中断的屏蔽，这样才能响应该中断，相对应的寄存器为INTMSK；

还要设置外部中断的触发方式，如低电平、高电平、上升沿、下降沿等，相对应的寄存器为EXTINTn。比如设置EINT8使用下降沿触发：

rEXTINT1 &= ~(7<<0);

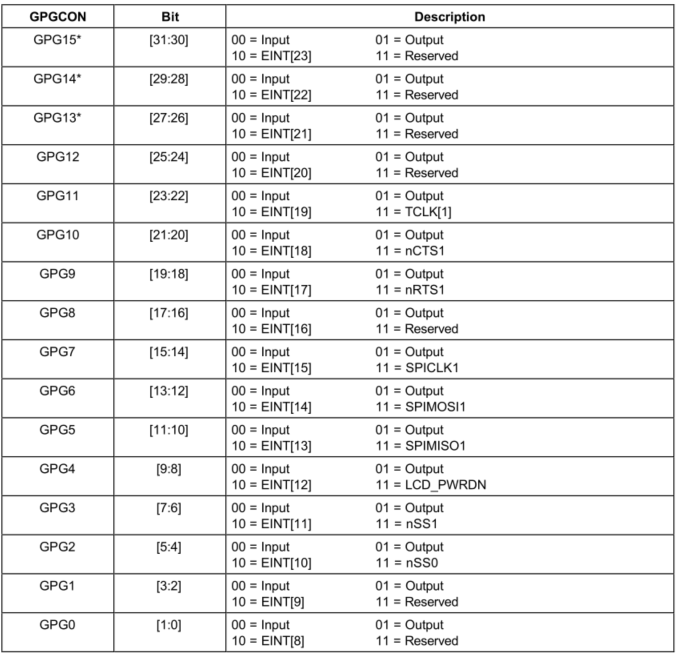
rEXTINT1 |= (3<<0); //设置EINT[8]下降沿触发

另外由于EINT4到EINT7共用一个中断向量，EINT8到EINT23也共用一个中断向量，而INTMSK只负责总的中断向量的屏蔽，要具体打开某一具体的中断屏蔽，还需要设置EINTMASK。比如使能EINT8：

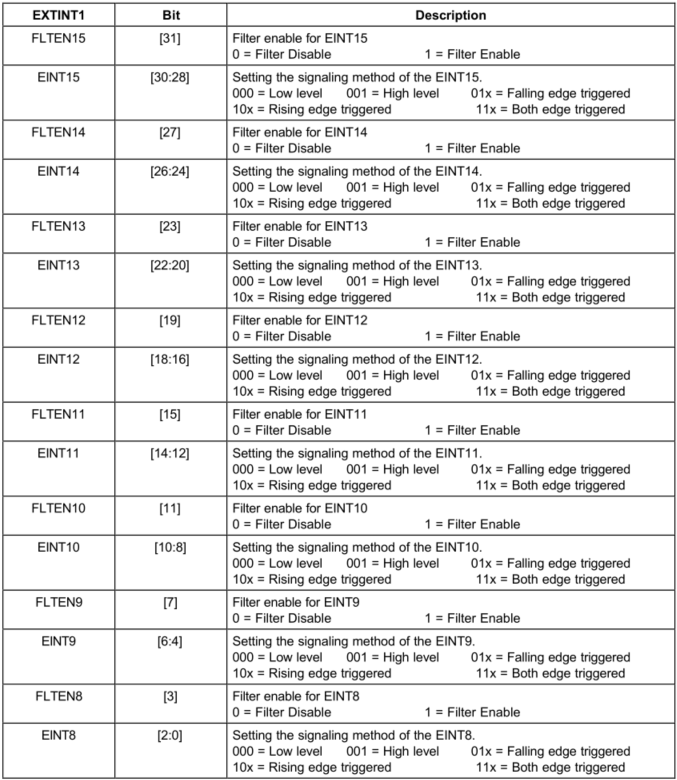
rEINTMASK &= ~(1<<8)

相关寄存器如下：

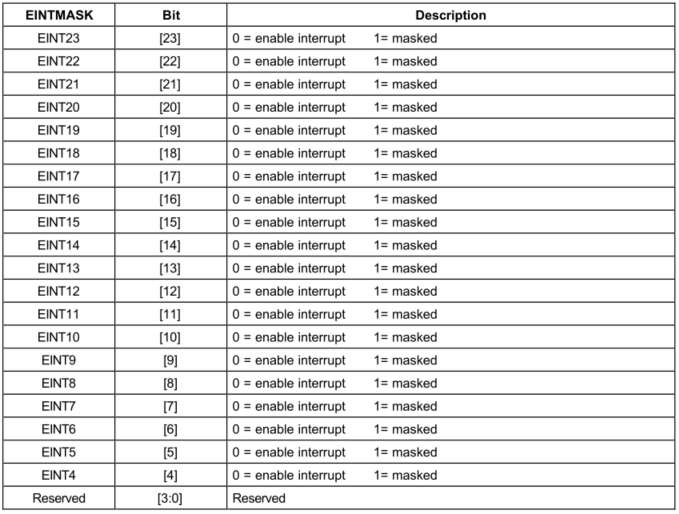
1. 端口G的控制寄存器各位含义如下：



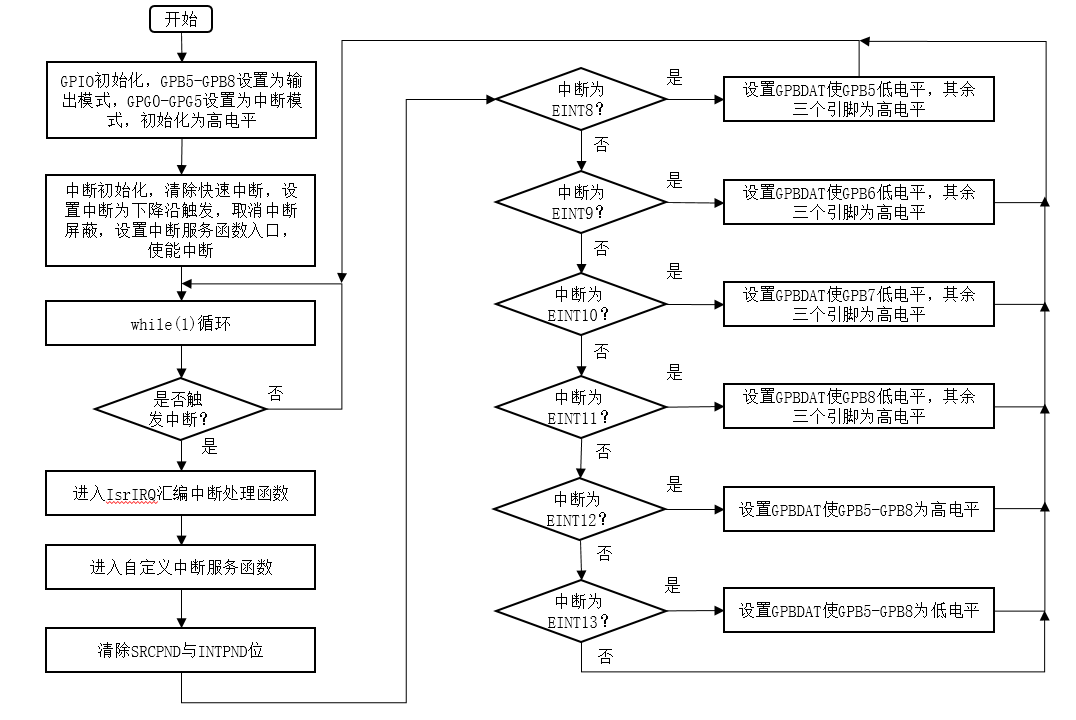
1. 外部中断控制EINT1的部分位含义如下：



1. 外部中断屏蔽寄存器（EINTMASK）



1. **实验流程图如下：**



1. 实验内容：

按下面电路图，编写一个基于中断的按键处理程序，并通过MDK的仿真功能进行验证。



实验要求：

1. 系统有6个按键作为输入，这6个按键分别连接到EINT6-EINT13这6个外中断输入端，系统以中断的方式处理这些按键输入；
2. 有4个指示灯作为输出(接GPB5-GPB5端口)，端口输出低电平时，对应的LED等被点亮；
3. K1按键按下后，指示灯LED1灯亮；K2按键按下后，指示灯LED2灯亮；K3按键按下后，指示灯LED3灯亮；K4按键按下后，指示灯LED4灯亮；K5按键按下后，所有的指示灯都被灯亮；K6按键按下后，熄灭所有的指示灯；
4. 主程序C语言编写程序，给出完整程序并添加注释。
5. 通过MDK的仿真功能验证程序的正确性。
6. 实验器材（设备、元器件）：
7. PC机一台；
8. Keil MDK-ARM uVision5开发工具。
9. 实验步骤：
10. 打开Keil MDK-ARM uVision5开发工具；
11. 新建一个工程文件；
12. 在新建的工程文件中，添加新的源程序文件
13. 编写代码
14. 选择“Build target”菜单对编写好的工程文件进行编译链接。
15. 点击““Start/Stop Debug Section””按键，对程序进行跟踪调试，在调试界面，单步执行，对CPU各寄存器的值的变化、以及相关内存的变化进行分析比较，判断程序的执行是否符合预期要求。在程序调试期间，利用MDK的“Logic Analyzer”对输出波形进行仿真，以验证程序执行是否正确。仿真的步骤如下：

[1] 在调试界面中，选择“Analysis Windows”→“Logic Analyzer”，进入“Logic Analyzer”界面后，点击“Setup”按键，设置需要观察的输出端口。

[2] 进入“Setup Logic Analyzer”窗口后，就可以对需要观察的输出端口引脚信号进行设置。由于本实验主要涉及到GPB5-8共4个输出端口的输出信号进行观察，因此需要添加这4个信号。

[3] 设置好相关引脚信号后，就可以进行回到“Debug”界面，调试运行。

[4] 一次测试按键K1-K6（EINT8-EINT13）输入变为低电平的时候情况，查看相关中断服务程序是否能产生正确的输出波形，点亮或熄灭对应的LED灯。

1. 最后得到完整的仿真输出波形。
2. 实验结果与分析（含重要数据结果分析或核心代码流程分析）
3. **实验代码**

代码1 main.c实验代码

|  |
| --- |
| #include <stdlib.h>  #include <string.h>  #include <S3C2440.H>  #include "2440addr.h"  #define U32 unsigned int  U32 GPB = 0x00; //记录GPBDAT的值  static void \_\_irq Key\_ISR(void) //定义中断服务程序  {  if(rINTPND==BIT\_EINT8\_23) { //INTPND同时只能有一位为1  ClearPending(BIT\_EINT8\_23); //清SRCPND、INTPND （写1清零）  }  if(rEINTPEND&(1<<8)) { //判断EINT8是否发生中断  rEINTPEND |= 1<<8; //清EINT8中断 （写1清零）  GPB = GPBDAT;  GPB |= 0X1E0; //灯全灭  GPB &=~ 0X20; //LED1亮  GPBDAT =GPB;  }  if(rEINTPEND&(1<<9)) {//判断EINT9是否发生中断  rEINTPEND |= 1<< 9; //清EINT9中断 （写1清零）  GPB = GPBDAT;  GPB |= 0X1E0; //灯全灭  GPB &=~ 0X40; //LED2亮  GPBDAT =GPB;  }  if(rEINTPEND&(1<<10)) {//判断EINT10是否发生中断  rEINTPEND |= 1<< 10; //清EINT10中断 （写1清零）  GPB = GPBDAT;  GPB |= 0X1E0; //灯全灭  GPB &=~ 0X80; //LED3亮  GPBDAT =GPB;  }  if(rEINTPEND&(1<<11)) {//判断EINT11是否发生中断  rEINTPEND |= 1<< 11; //清EINT11中断 （写1清零）  GPB = GPBDAT;  GPB |= 0X1E0; //灯全灭  GPB &=~ 0X100; //LED4亮  GPBDAT =GPB;  }  if(rEINTPEND&(1<<12)) {//判断EINT12是否发生中断  rEINTPEND |= 1<< 12; //清EINT12中断 （写1清零）  GPB = GPBDAT;  GPB &=~ 0X1E0; //灯全亮  GPBDAT =GPB;  }  if(rEINTPEND&(1<<13)) {//判断EINT13是否发生中断  rEINTPEND |= 1<< 13; //清EINT13中断 （写1清零）  GPB = GPBDAT;  GPB |= 0X1E0; //灯全灭  GPBDAT =GPB;  }  }  void IRQ\_init(void)  {  rINTMOD = 0;  rEXTINT1 &=~ ((7<<0)|(7<<4)|(7<<8)|(7<<12)|(7<<16)|(7<<20));//清除EXIINT对应EINT8-13的触发方式  rEXTINT1 |= ((3<<0)|(3<<4)|(3<<8)|(3<<12)|(3<<16)|(3<<20));//均设置为下降沿触发  rEINTMASK &=~((1<<8)|(1<<9)|(1<<10)|(1<<11)|(1<<12)|(1<<13)); //使中断不被屏蔽  pISR\_EINT8\_23 = (U32)Key\_ISR; //中断服务程序  EnableIrq(BIT\_EINT8\_23); //使能中断  }  void GPIO\_Init(void)  {  GPBCON &=~0X3FC00;//GPB5-GPB8位清零  GPBCON |= 0X15400;//GPB5-GPB8设置为输出模式  GPGCON &=~0XFFF;//GPG0-GPG5位清零  GPGCON |= 0XAAA;//设置为EINT8-13模式  GPB = GPBDAT; //复制GPBDAT  GPB |= 0X1E0 ;//灯全部熄灭  GPBDAT = GPB; //写回GPBDAT  GPGDAT |= 0X3F;//GPG0-GPG5初始化为高电平  }  int main()  {  GPIO\_Init();  IRQ\_init();  while(1);  } |

1. **运行过程及结果界面截图**
2. 图1是程序运行，还没有按键按下时，系统输出的波形。从图中可以看出，所有LED灯对应的输出端口都是高电平，没有电灯被点亮。

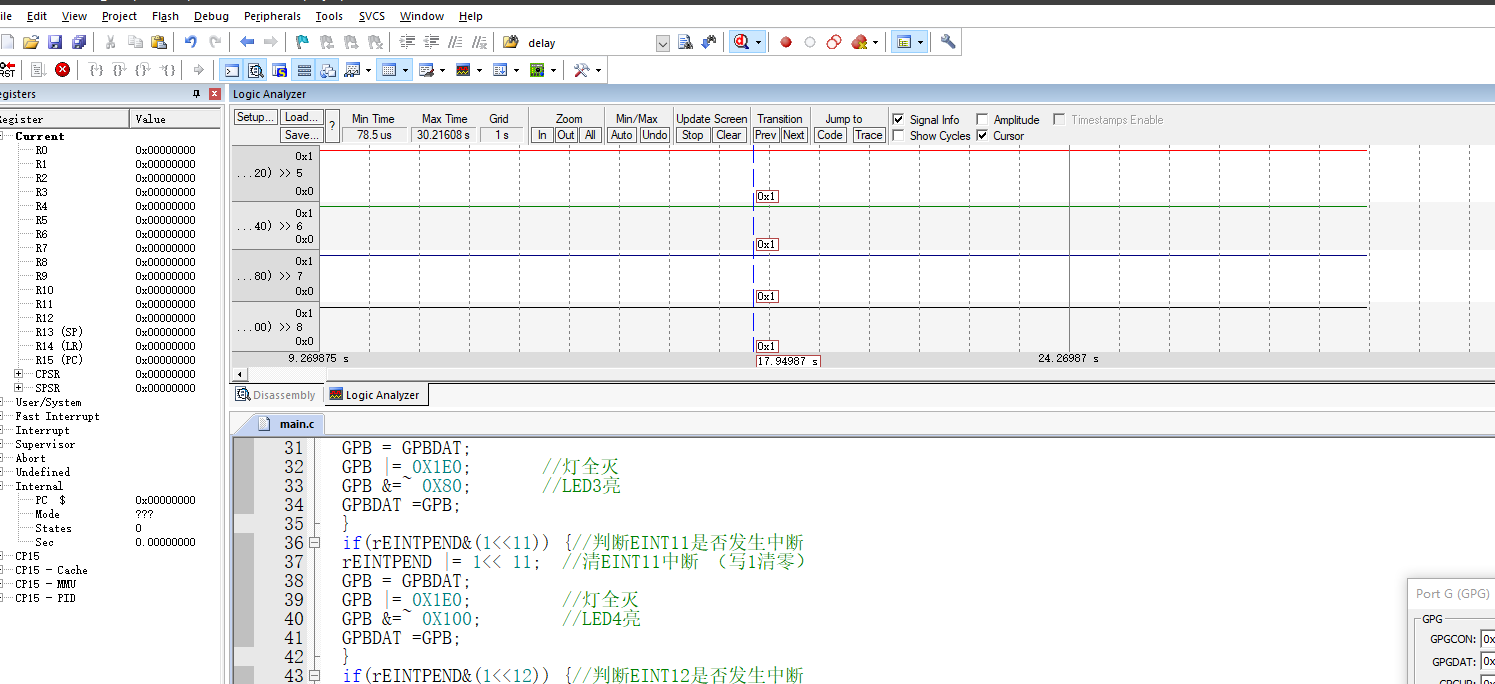


图1 系统运行后没有按键按下时的输出波形

1. 图2是按键K1按下，PORTG.0输入变为低电平的时候情况：LED1对应的引脚变为低电平，LED1被点亮。

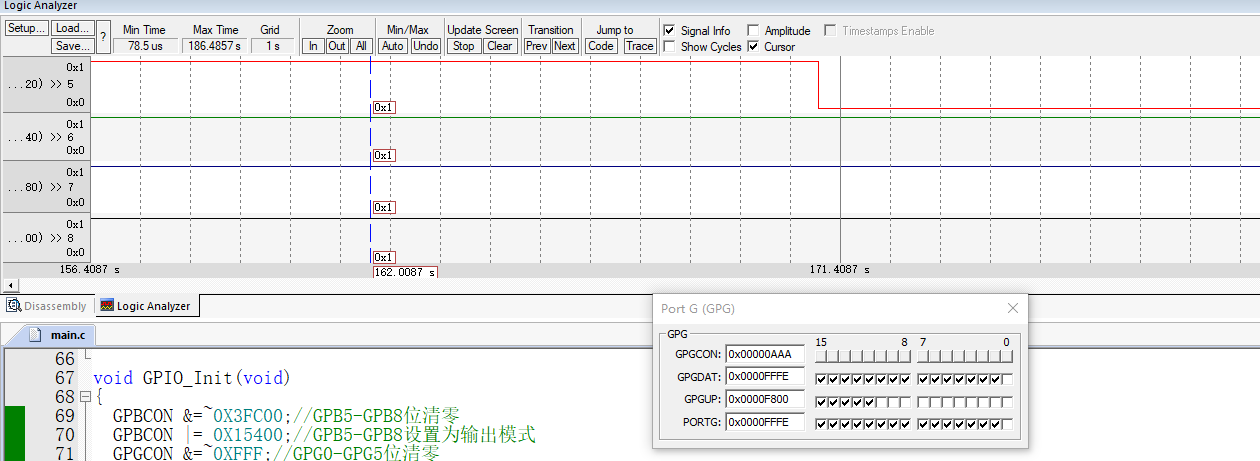


图2 按键K1按下时的输出波形

1. 图3是按键K2按下，PORTG.1输入变为低电平的时候情况：LED2对应的引脚变为低电平，LED2被点亮。

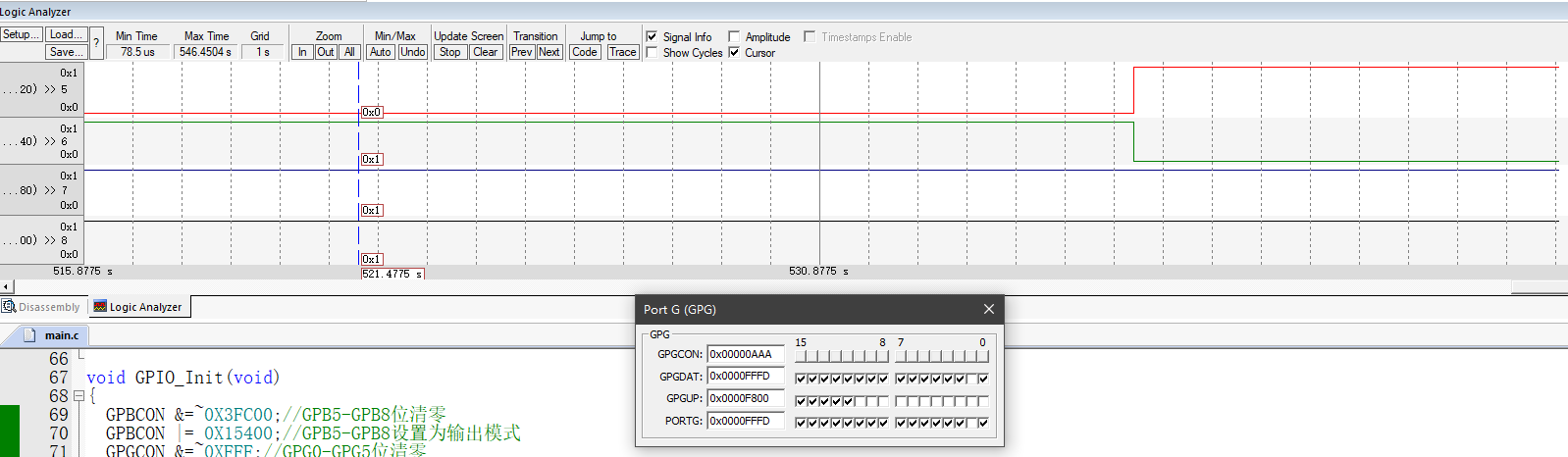


图3 按键K2按下时的输出波形

1. 图4是按键K3按下，PORTG.2输入变为低电平的时候情况：LED3对应的引脚变为低电平，LED3被点亮。

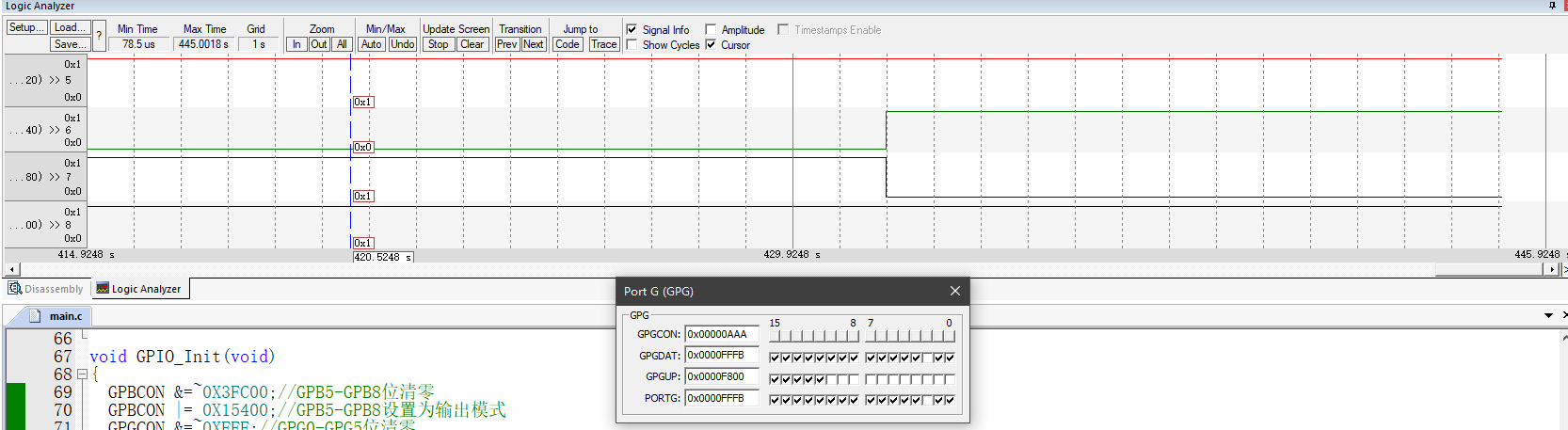


图4 按键K3按下时的输出波形

1. 图5是按键K4按下，PORTG.3输入变为低电平的时候情况：LED4对应的引脚变为低电平，LED4被点亮。

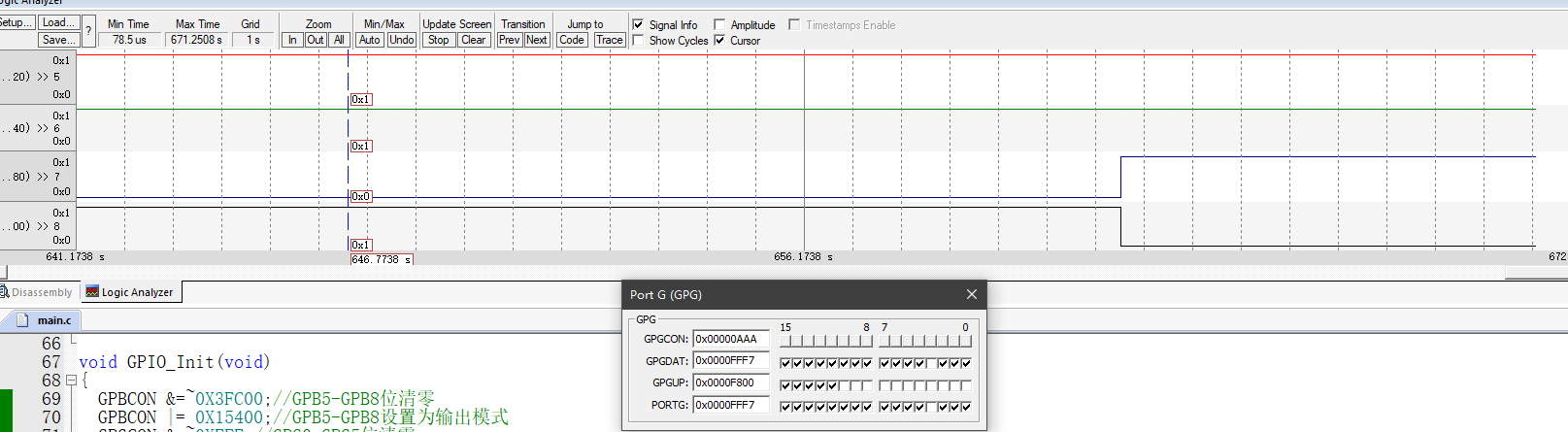


图5 按键K4按下时的输出波形

1. 图6是按键K5按下，PORTG.4输入变为低电平的时候情况：所有LED等对应的4个引脚变为低电平，所有LED被点亮。

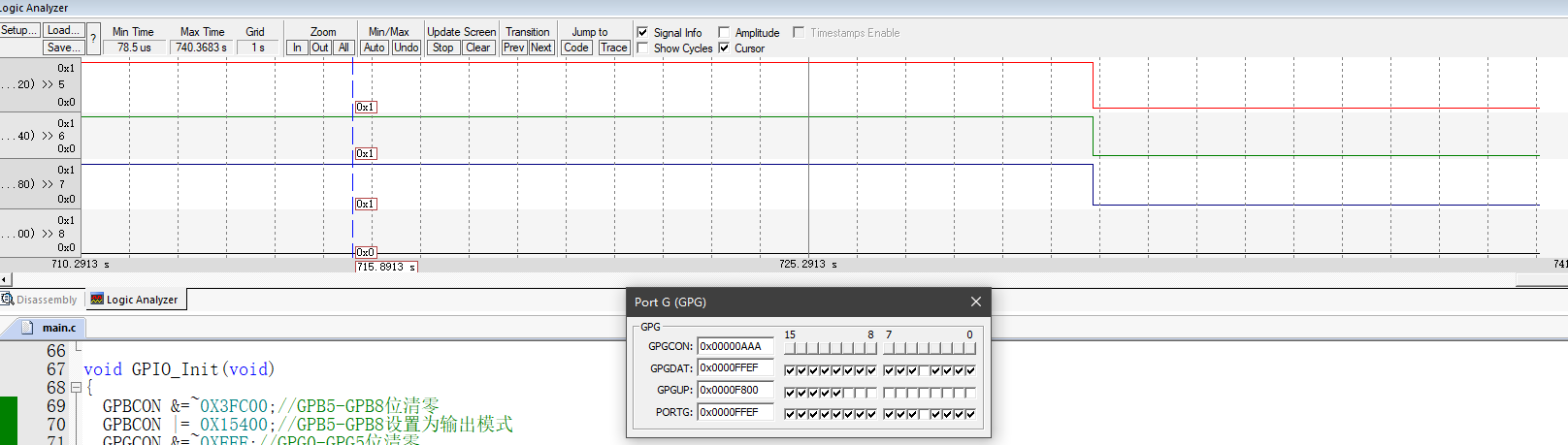


图6 按键K5按下时的输出波形

1. 图7是按键K6按下，PORTG.5输入变为低电平的时候情况：所有LED对应的引脚变为高平，所有LED等熄灭。

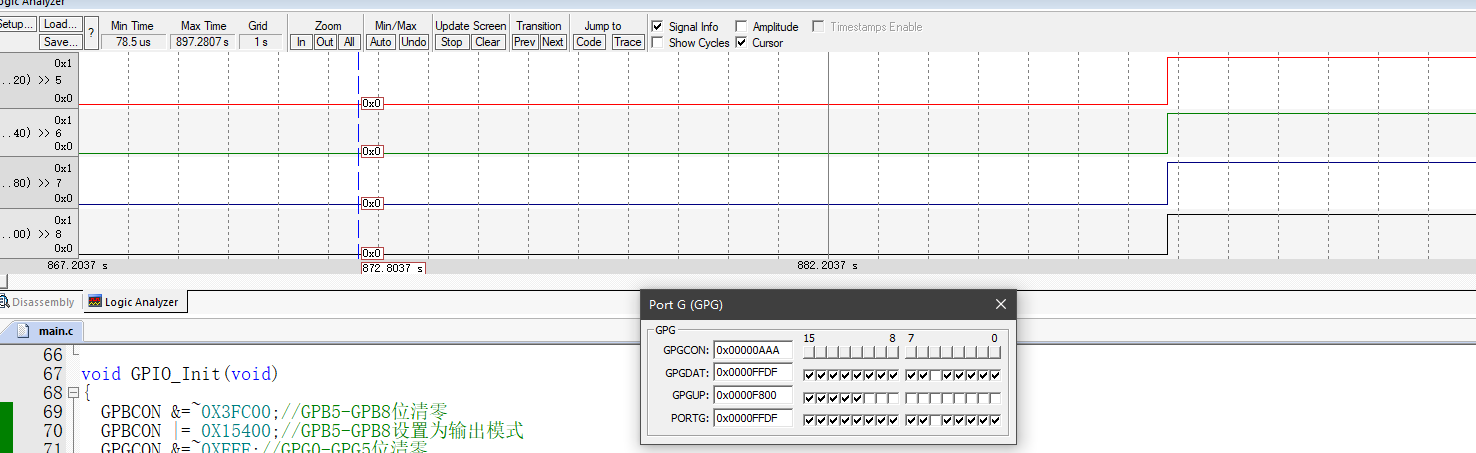


图7 按键K6按下时的输出波形

1. 图8是按键按下后，程序产生中断，先进入IRQ中断服务程序IsrIRQ执行的截图。

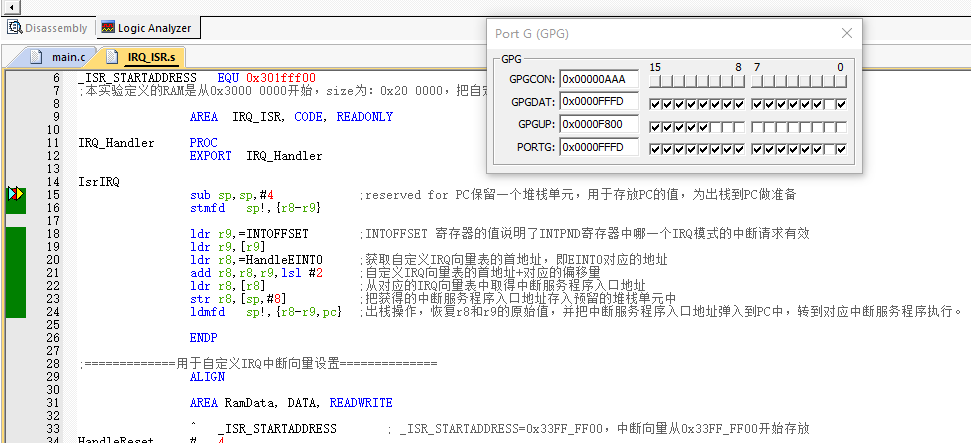


图8 按键按下后，系统运行到IsrIRQ中断服务程序的断点

1. 图9是按键按下后，程序产生中断，进入到C语言按键中断服务程序Key\_ISR时的截图。

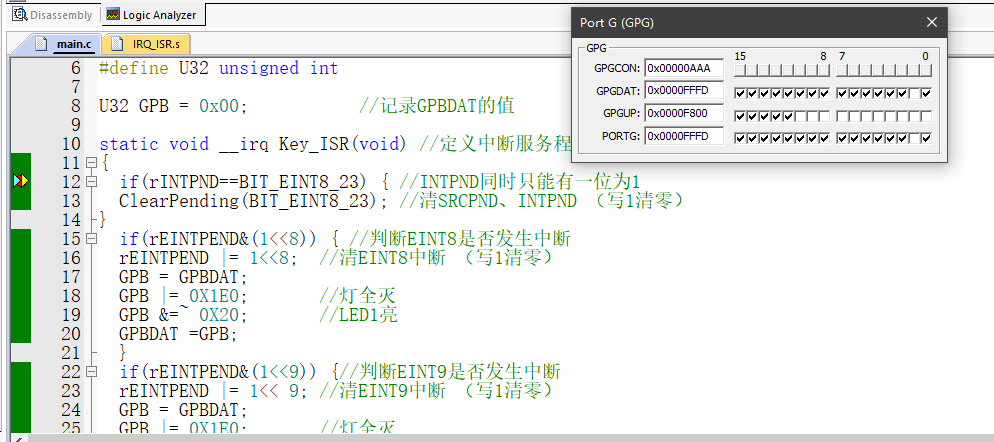


图9 按键按下后，系统运行到C语言按键中断服务程序Key\_ISR的断点

1. **实验结论**

通过仿真可以观察到程序在按键K1-K6按下时都准确执行了相应中断服务程序，实验结果符合预期。

1. 总结及心得体会：

通过本次实验我学会了使用函数封装对寄存器的操作从而使main函数内运行的代码没有实际上那么冗杂，同时也学习到了ARM中断的实际应用，实践完成了按键中断服务函数的编写，收获颇丰。

1. 对本实验过程及方法、手段的改进建议：

本实验中我对于寄存器的操作大多为根据手册内容修改相应位数据，实际上这样的代码在没有注释的情况下很难读，不便于后续修改，可以考虑多使用头文件中的宏定义来代替寄存器中对应位，增加代码的可读性。

报告评分：

指导教师签字：