电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2022130102013

姓 名 成棋伟

（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： 成棋伟 学号： 2022130102013**

**指导教师： 兰刚**

**实验地点： 西304 实验时间：2025.05.06**

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。

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。

1. 实验原理：

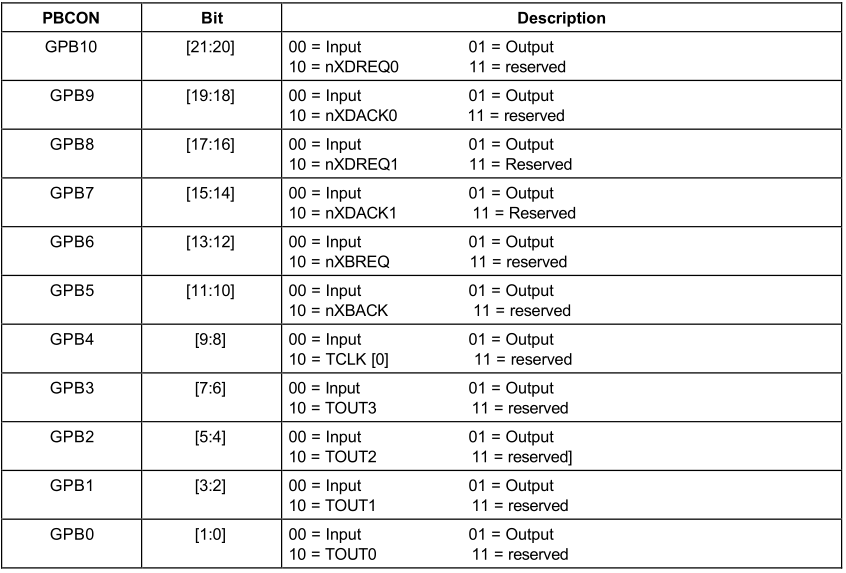
**1、端口B相关寄存器说明**

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：

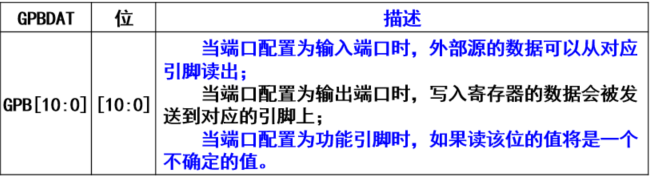


1. GPBCON各位的描述如下：

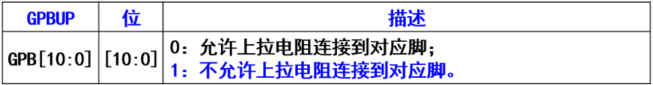


1. 端口B数据寄存器GPBDAT中各位的描述如下：

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：



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；

Keil MDK-ARM uVision5开发工具。

1. 实验步骤：
2. 打开Keil MDK-ARM uVision5开发工具；
3. 新建一个工程文件；
4. 在新建的工程文件中，添加新的源程序文件
5. 编写代码
6. 选择“Build target”菜单对编写好的工程文件进行编译链接。
7. 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：
8. 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。
9. 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。
10. 设置好相关引脚信号后，就可以进行回到“Debug”界面，调试运行。
11. 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。
12. 然后选择“Peripherals”→“I/O Ports”→“PortB(GPB)”，设置拨码开关K1（GPB1）输入为低电平。
13. 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。
14. 最后得到完整的仿真输出波形。
15. 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）
16. **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。

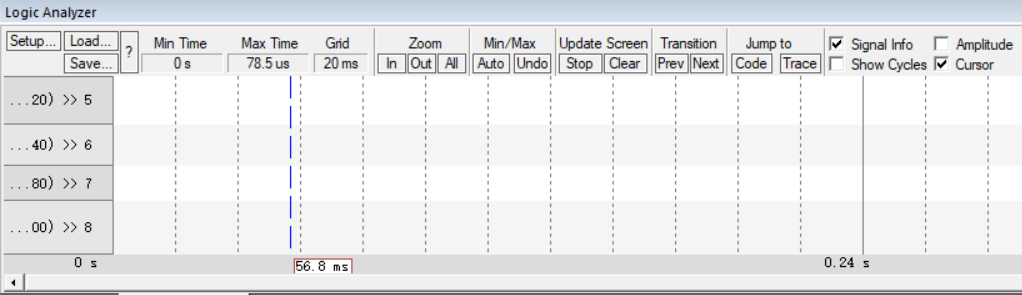


图1 设置完成输入设置后“Logic Analyzer”窗口显示

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.

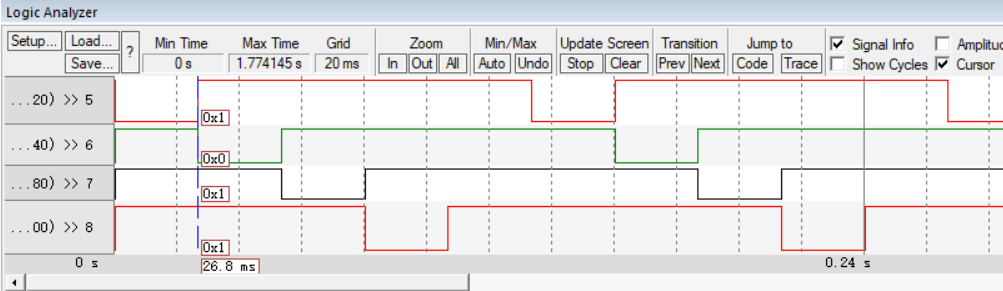


图2 拨码开关K1（GPB1）输入为高电平时的输出波形

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。

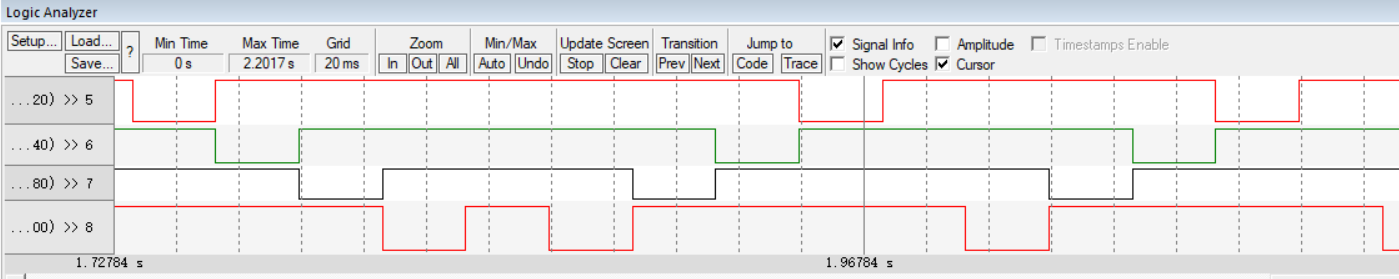


图3 完整的仿真输出波形

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。且低电平的持续时间约为19ms，即使得每个灯被点亮的持续时间约为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查询手册直接配置寄存器来操作外设，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。

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实现大致延时功能，建议加入delay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： 成棋伟 学号：2022130102013**

**指导教师： 兰刚**

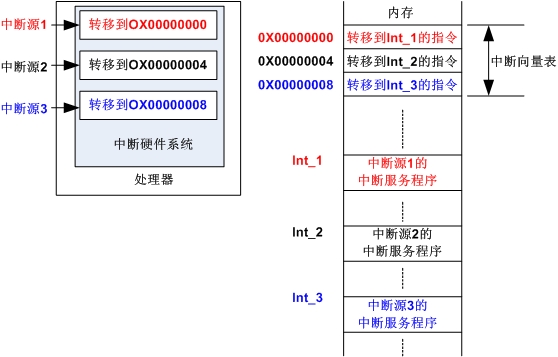
**实验地点： 西303 实验时间： 2025.05.13**

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**

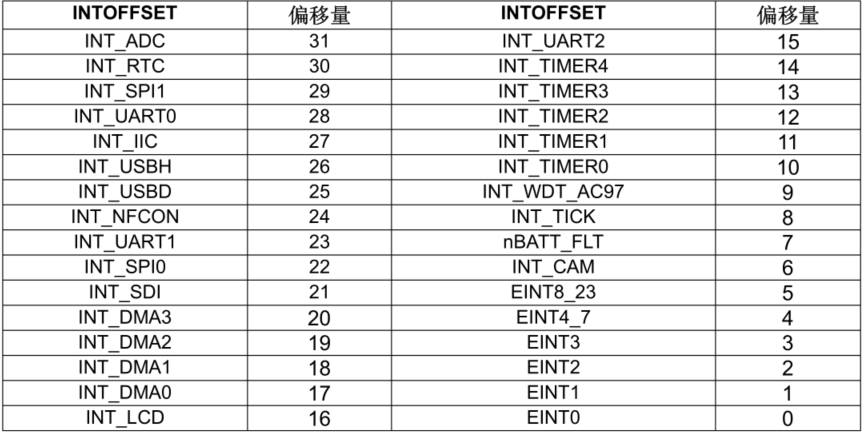
S3C2440X中断控制器有60个中断源，对外提供24个外中断输入引脚，内部所有设备都有中断请求信号，例如：DMA控制器、UART、IIC等。中断控制器中包括5个控制寄存器：源挂起寄存器、中断模式寄存器、屏蔽寄存器、优先级寄存器、中断挂起寄存器。

1. 中断屏蔽寄存器INTMSK

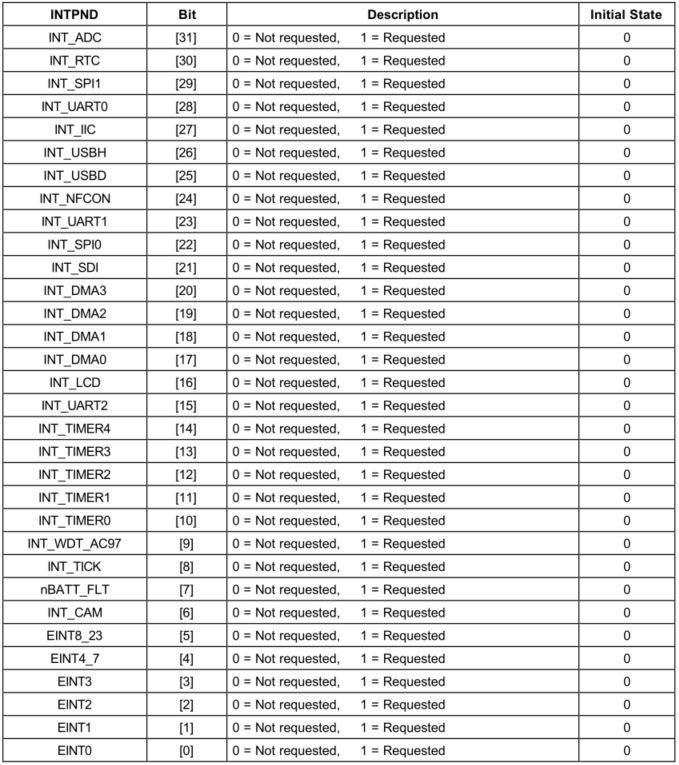
中断屏蔽寄存器INTMSK包括了32位，每一个比特位均与相应的一个中断源相对应。如果某位被设置为1，那么CPU不会执行相应中断源提出的中断请求（即使源挂起寄存器的相应位被设置为1）。如果屏蔽位为0，那么中断请求会被正常执行。

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。



中断挂起寄存器 INTPND（C语言程序中使用rINTPND进行访问）



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);

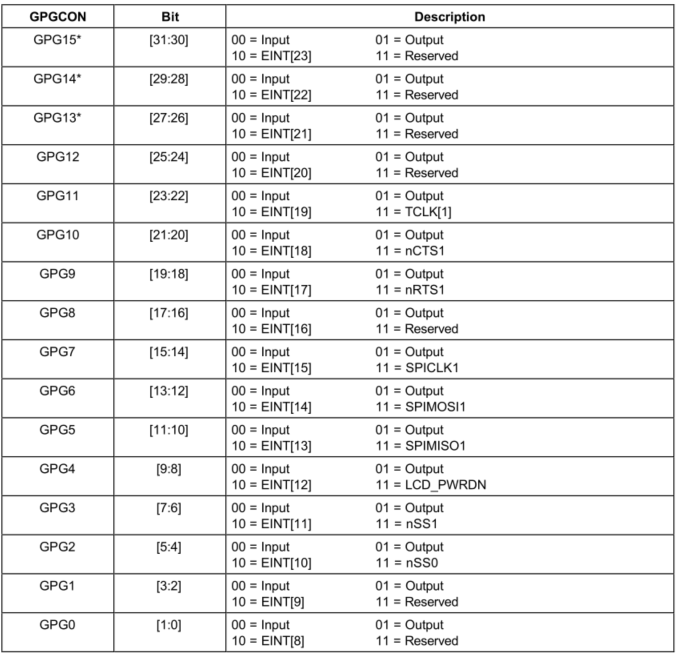
rEXTINT1 |= (3<<0); //设置EINT[8]下降沿触发

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：

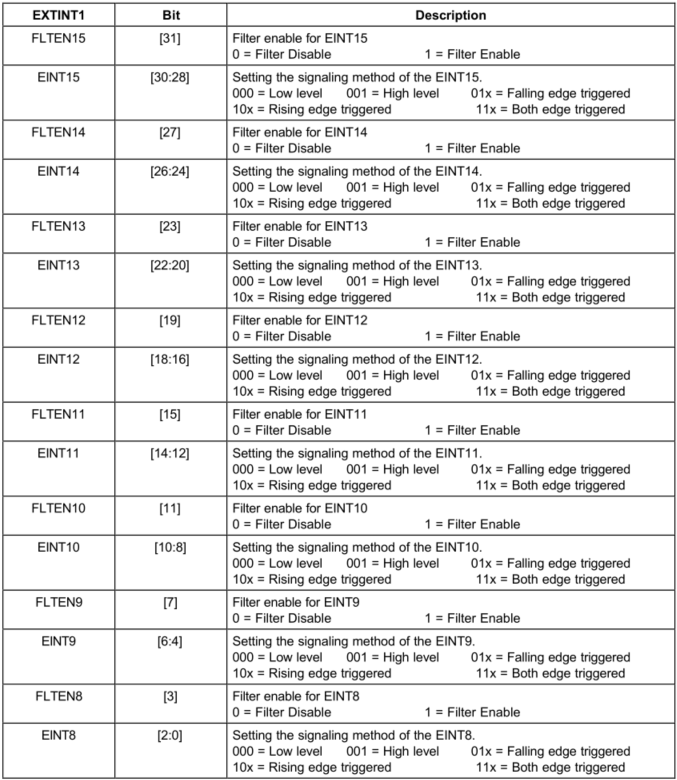
rEINTMASK &= ~(1<<8)

相关寄存器如下：

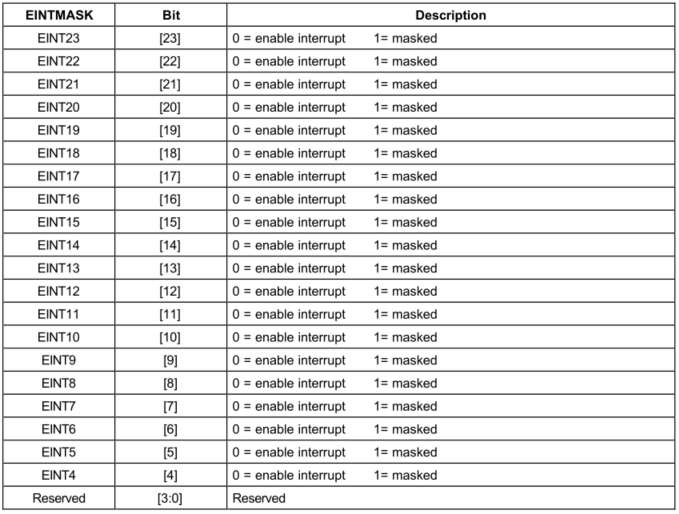
1. 端口G的控制寄存器各位含义如下：



1. 外部中断控制EINT1的部分位含义如下：



外部中断屏蔽寄存器（EINTMASK）



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。



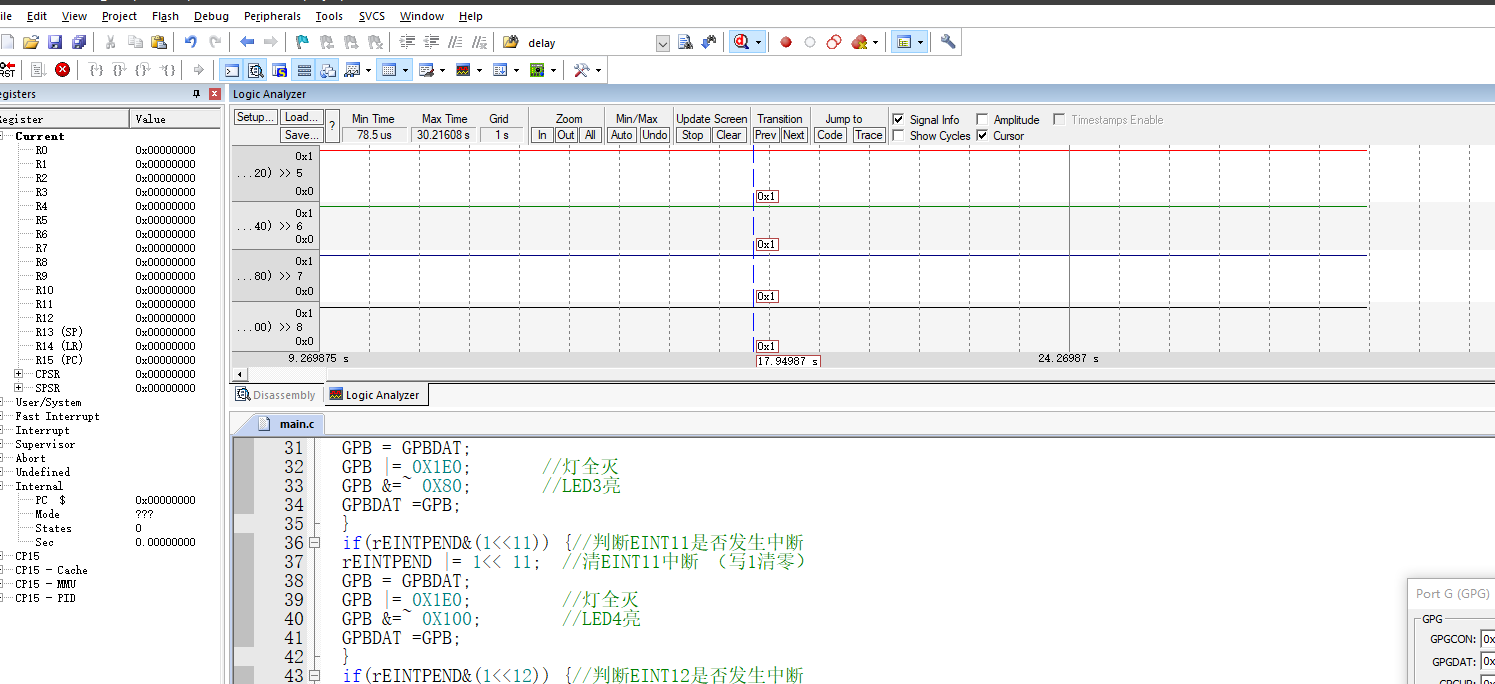
实验要求：

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：
16. 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。
17. 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。
18. 设置好相关引脚信号后，就可以进行回到“Debug”界面，调试运行。
19. 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。
20. 最后得到完整的仿真输出波形。
21. 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）
22. **实验代码**

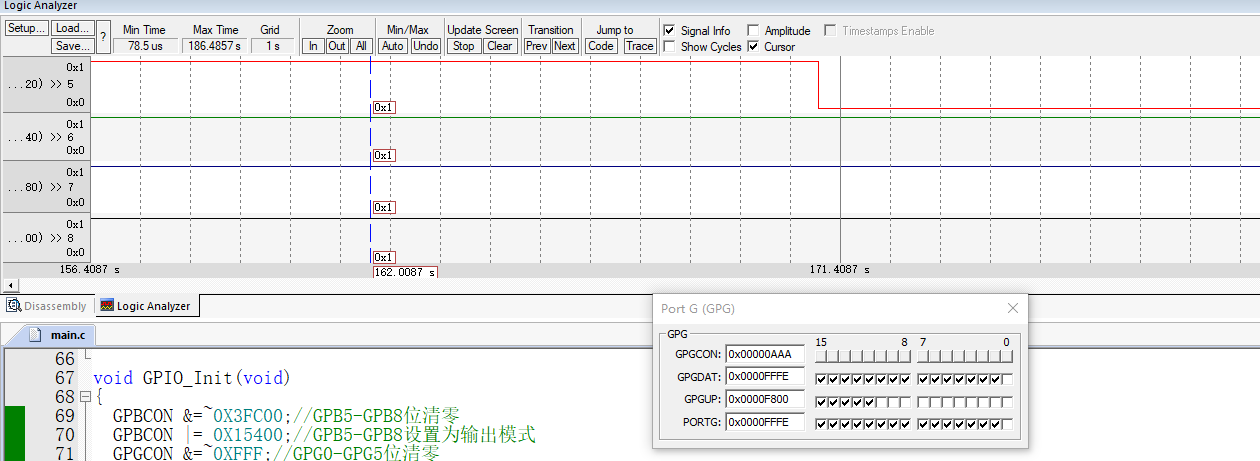
代码1 main.c实验代码

|  |
| --- |
| #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);  } |

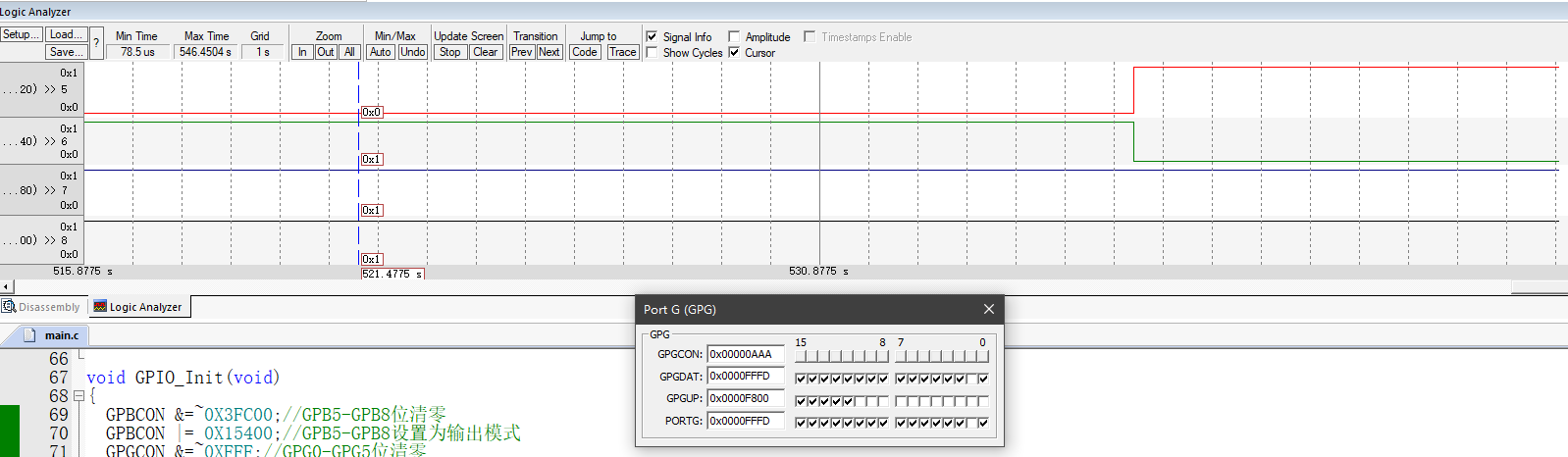
1. **运行过程及结果界面截图**
2. 图1是程序运行，还没有按键按下是，系统输出的波形。从图中可以看出，所有LED灯对应的输出端口都是高电平，没有电灯被点亮。



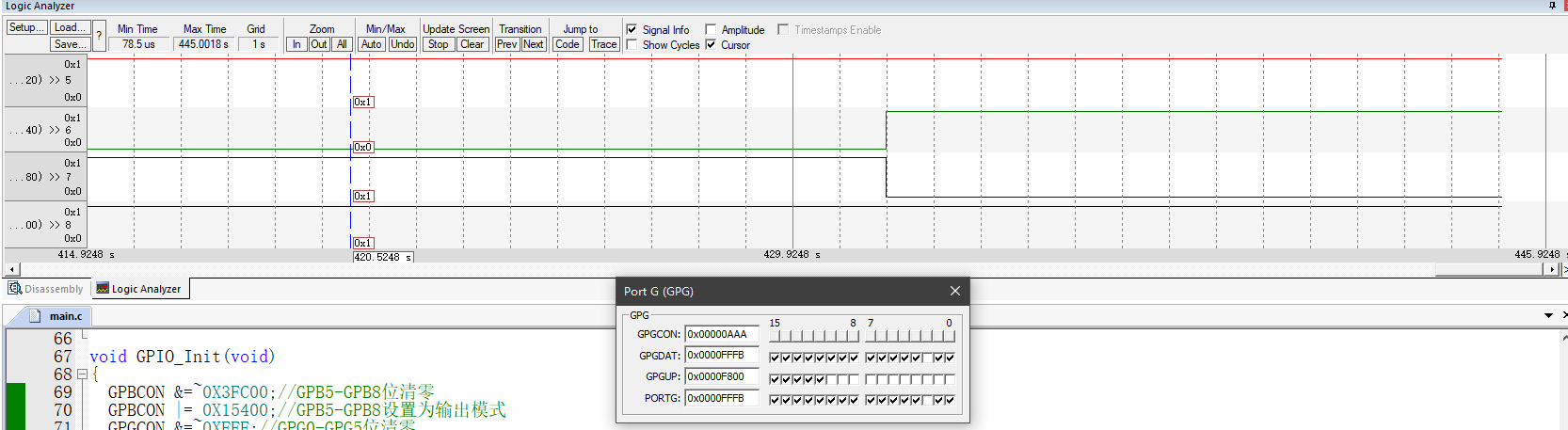
1. 图1 系统运行后没有按键按下时的输出波形
2. 图2是按键K1按下，PORTG.0输入变为低电平的时候情况：LED1对应的引脚变为低电平，LED1被点亮。



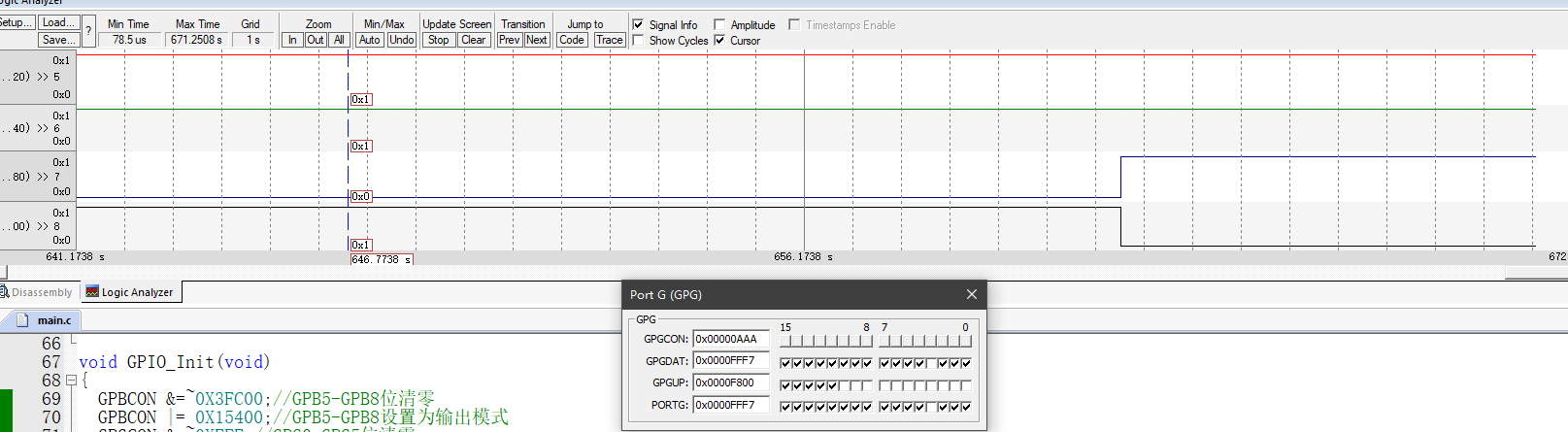
1. 图2 按键K1按下时的输出波形
2. 图3是按键K2按下，PORTG.1输入变为低电平的时候情况：LED2对应的引脚变为低电平，LED2被点亮。



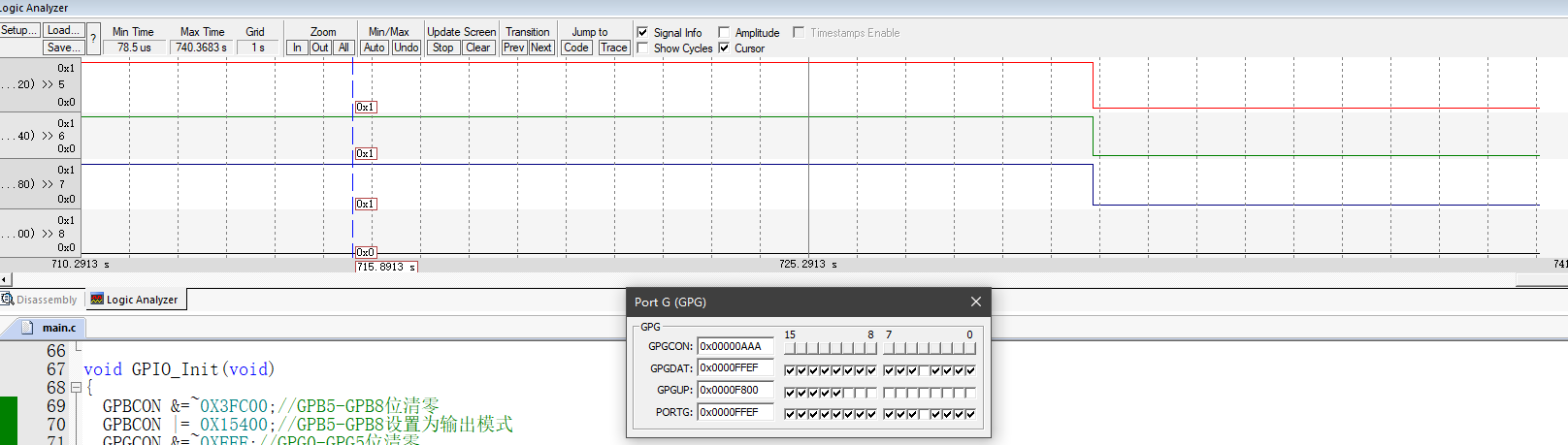
1. 图3 按键K2按下时的输出波形
2. 图4是按键K3按下，PORTG.2输入变为低电平的时候情况：LED3对应的引脚变为低电平，LED3被点亮。
3. 图4 按键K3按下时的输出波形
4. 图5是按键K4按下，PORTG.3输入变为低电平的时候情况：LED4对应的引脚变为低电平，LED4被点亮。



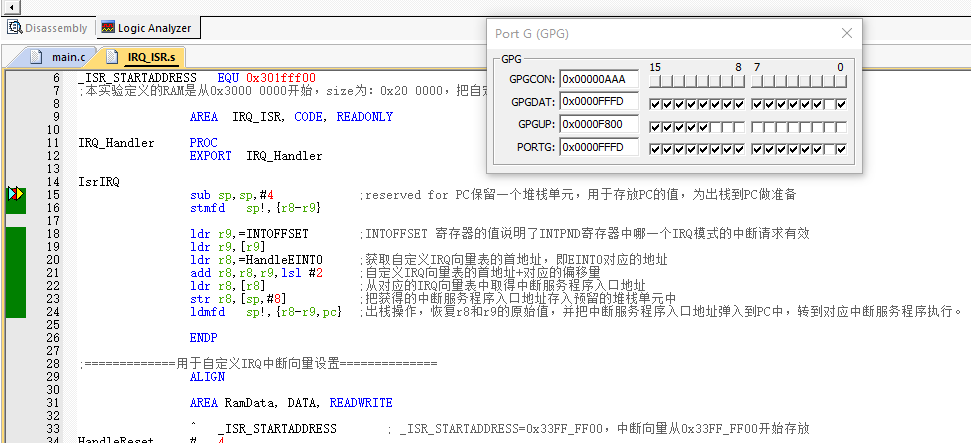
1. 图5 按键K4按下时的输出波形
2. 图6是按键K5按下，PORTG.4输入变为低电平的时候情况：所有LED等对应的4个引脚变为低电平，所有LED被点亮。



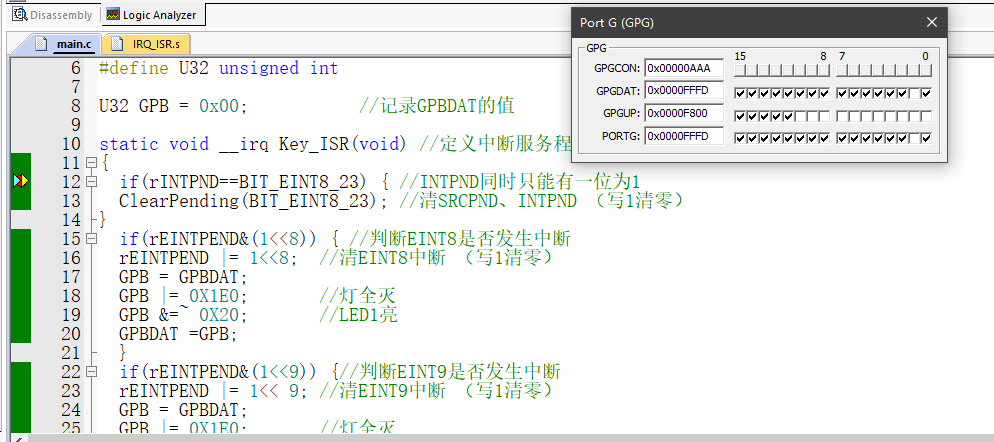
1. 图6 按键K5按下时的输出波形
2. 图7是按键K6按下，PORTG.5输入变为低电平的时候情况：所有LED对应的引脚变为高平，所有LED等熄灭。



1. 图7 按键K6按下时的输出波形
2. 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。



1. 图8 按键按下后，系统运行到IsrIRQ中断服务程序的断点
2. 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。



1. 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
2. **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。

**思考题**

EINT8-23这几个外部中断，每个都对应一个外部引脚。如果连接到这些外部引脚的多个外部中断源（EINT8-23）同时提出中断请求，那么CPU响应那个外部中断呢？

当多个EINT8-23中断同时触发时，CPU会响应INT\_EINT8\_23主中断，但具体处理哪个中断由软件在ISR中通过轮询标志位决定。硬件不自动分配优先级，最终执行顺序取决于软件逻辑（如先检查低编号引脚的中断）。

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：