

**按键触发外部中断控制LED**

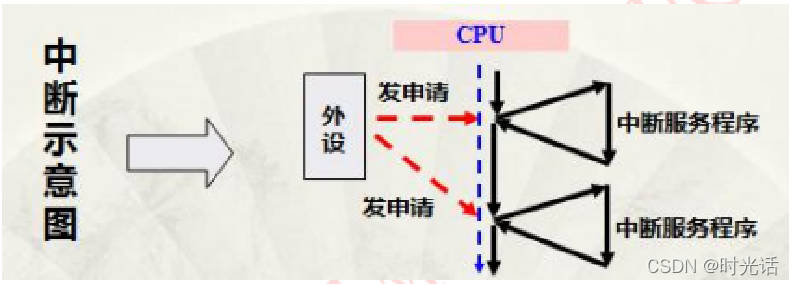
|  |  |
| --- | --- |
| **姓 名** | **冯浩然** |
| **学 号** | **202100800378** |
| **专 业** | **电子科学与技术** |
| **班 级** | **2021级1班** |
| **学 院** | **机电与信息工程学院** |
| **指导教师** | **王小利** |
| **课程名称** | **DSP原理与应用** |
| **课 程 号** | **28033160** |
| **学年学期** | **2023–2024学年第2学期** |
| **实验日期** | **2024年6月13日** |

# 一 中断概念

## 1 中断介绍

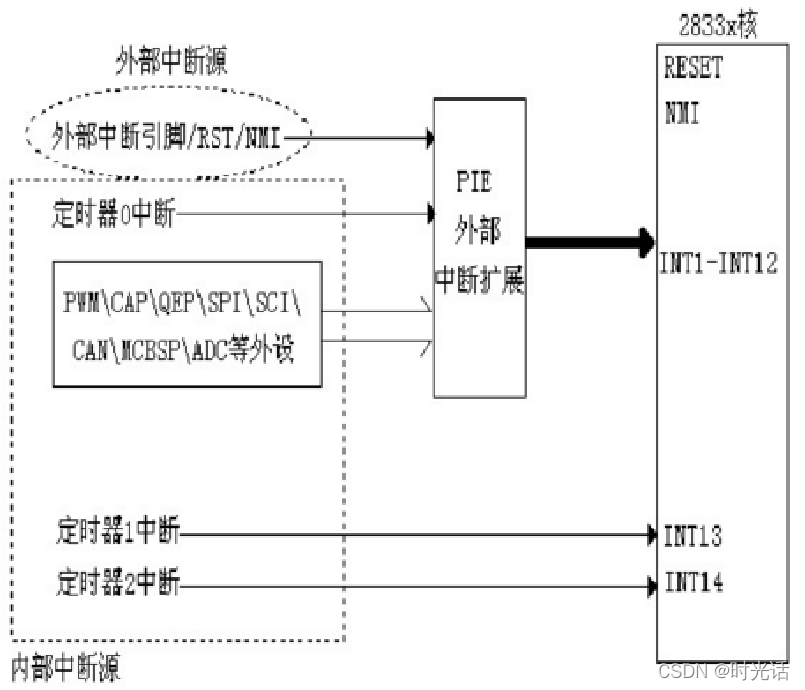
### 1.1中断概念

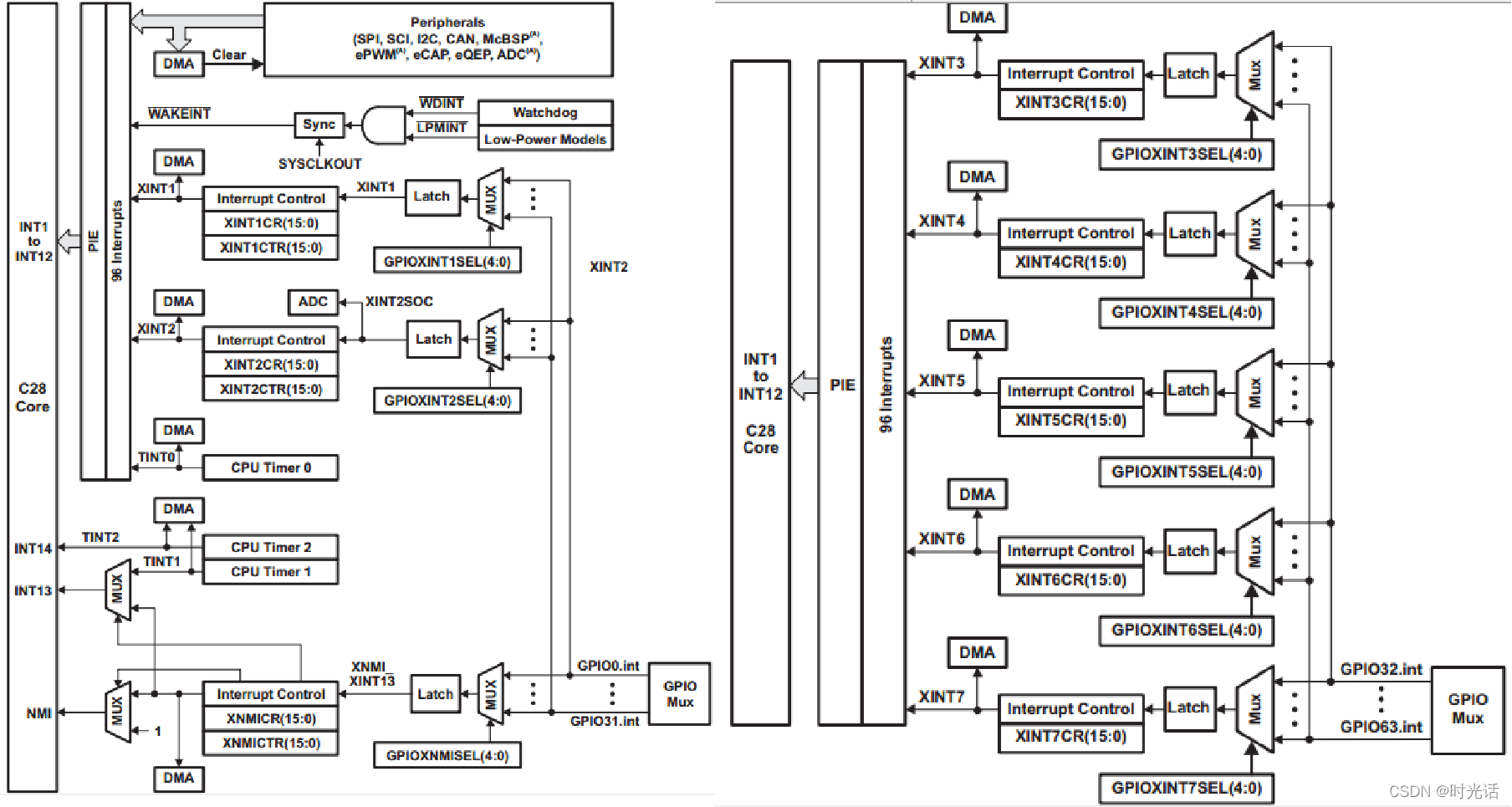
中断其实就是当 CPU 执行程序时，由于发生了某种随机的事件(外部或内部)，引起 CPU 暂时中断正在运行的程序，转去执行一段特殊的服务程序(中断服务子程序或中断处理程序)，以处理该事件，该事件处理完后又返回被中断的程序继续执行，这一过程就称为中断，引发中断的称为中断源。



### 1.2 F28335中断概述

      F28335内部有16个中断线，其中包括2个不可屏蔽中断（RESET和NMI）与14个可屏蔽中断。





 （GPIO0-GPIO31对应XINT1和XINT2， GPIO32-GPIO63对应XINT3--XINT7）

### 1.3 F28335中断机制

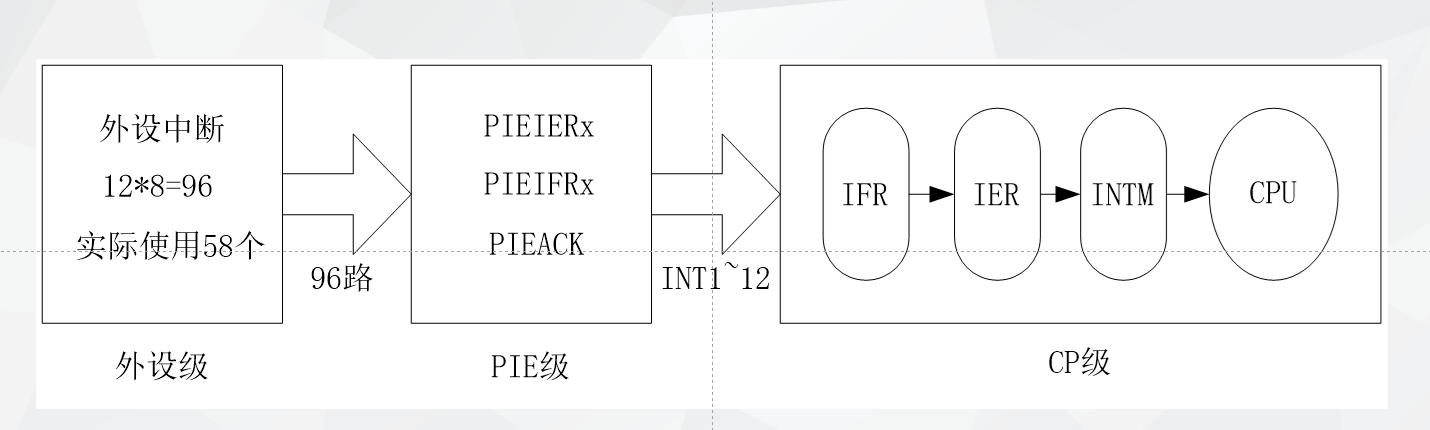
F28335的中断采用的是3级中断机制：外设级中断、PIE级中断、CPU级中断，其中CPU级中断最为内核，即CPU只能响应从CPU中断线上过来的中断请求，而F28335中断源很多，CPU没有那么多中断线，因此便有了中间层的PIE级中断，外设要能够成功产生中断响应，需满足：

1） 经过外设级中断允许；

2） 经过PIE允许；

3） 经过CPU允许，并作出响应；

其工作流程如下图所示：



**1） 外设级中断：**当DSP硬件系统的某个外设因为软件或硬件产生一个中断，此时该中断事件对应的中断的标志位置1；如果此时，该中断事件对应的中断的使能位置1，则该外设就会向PIE控制器发出一个中断请求。相反如果中断事件发生，但如果该中断对应的使能位未置1，则该外设便不会向PIE控制器发出中断请求。

需要注意的是：当中断事件发生，中断标志位置1，但对应的中断使能位未置1，此时中断标志位会一直保持置位状态，直到用程序将其清除为止。当然，中断标志位保持置位状态时，一旦该中断使能位置1，则外设会立即向PIE控制器发送中断请求。

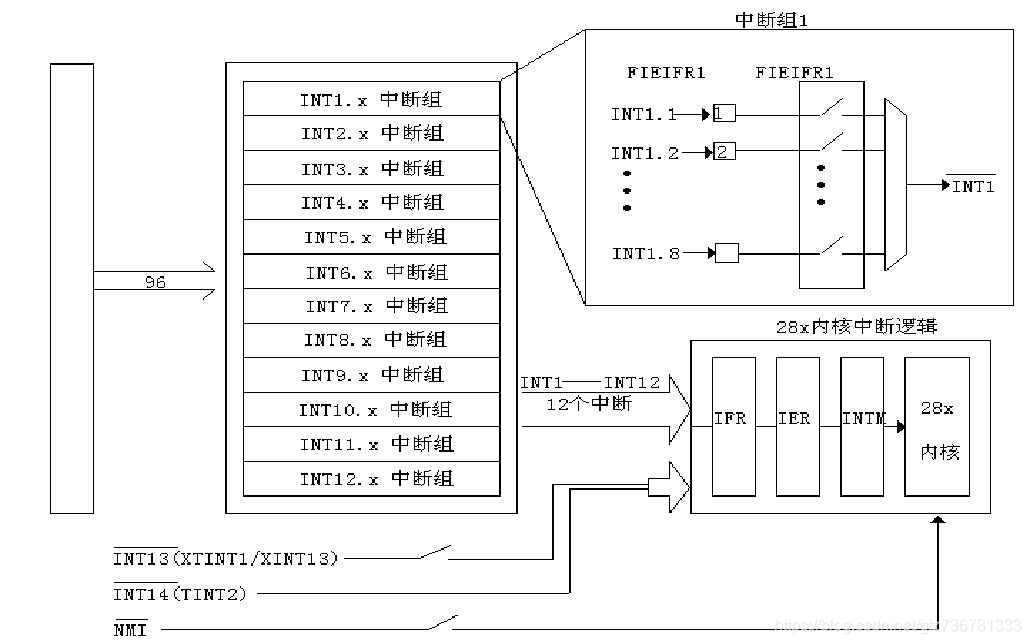
当中断事件发生以后，其对应的中断标志位会无条件的置1，中断标志位的状态会通过中断使能位这个开关发送给PIE控制器。

并且，不管什么情况下，外设寄存器中的中断标志位都必须手工清除（这里指的是通过软件修改，区别于硬件置位方式）。

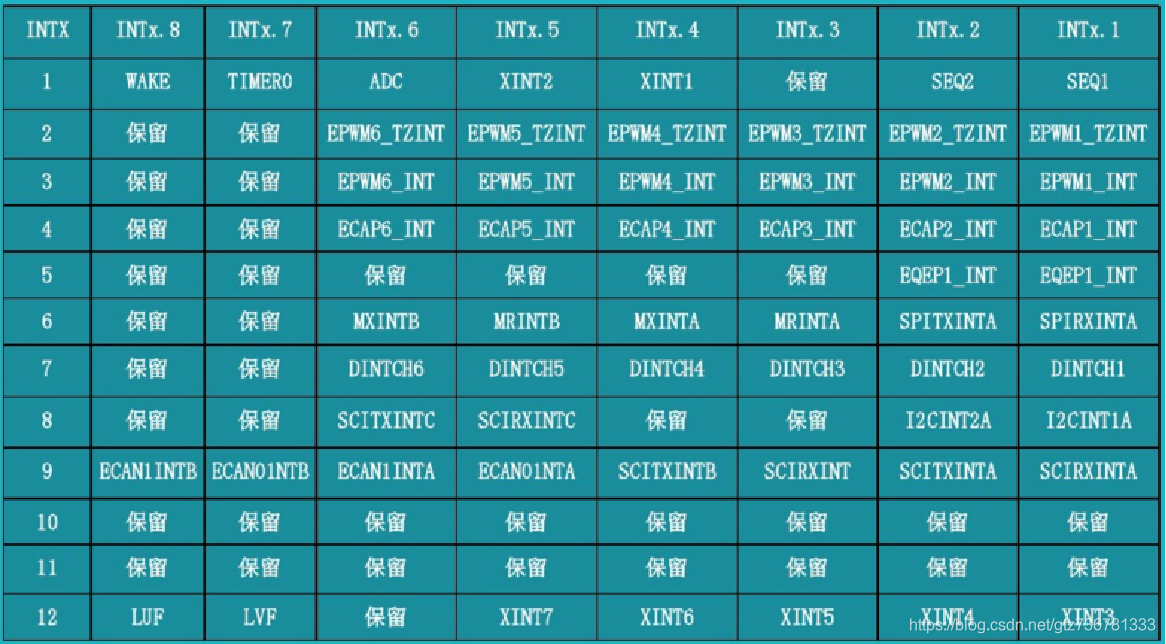
**2） PIE级中断：**F28335处理器内部集成了多种外设，每个外设都会产生一个或者多个外设级中断。由于CPU没有能力处理所有外设级的中断请求，因此F28335的CPU让出了12个中断线交给PIE模块进行复用管理。

当外设产生中断事件，其外设级中断标志位自动置1，若对应的中断使能位也置1，则该外设会向PIE控制器发送中断请求。

PIE将96个外设和外部引脚的中断进行了分区，每8个一组，共十二组，即PIE1-PIE12。每个组的中断被多路汇集进入了一个CPU中断。PIE控制器中每一个组都会有一个中断标志寄存器(PIEIFRx)和一个中断使能寄存器(PIEIERx)（x=1~12）。



从图中可以看出，PIE将外设中断分成了12个组，分别对应着CPU的12个可屏蔽中断线，每一组由8个外设级中断组成，这8个外设中断分别对应相应外设接口的中断引脚，PIE通过一个8选1的多路选择器将这8个外设中断组成一组。具体关系如下：



实际有效外设中断为58个，其余为保留。在PIE模块内每组中断有相应的中断标志位(PIEIFRx)和使能位(PIEIERx.y)。除此之外，每组PIE中断(INT1~INT12)有一个响应标志位(PIEACK)。

一旦PIE控制器有中断产生，相应的中断标志位(PIEIFRx.y)将置1。如果相应的PIE中断使能位也置1，则PIE将检查相应的PIEACKx以确定CPU是否准备响应该中断。如果相应的PIEACKx位清零，PIE向CPU申请中断；如果PIEACKx置1，PIE将等待到相应的PIEACKx清零才向CPU申请中断。PIE通过对PIEACKx的位控制来控制每1组只有1个中断能被响应，一旦响应后，就需要将PIEACKx相应位清零，以让它能够响应该组中后边过来的中断。

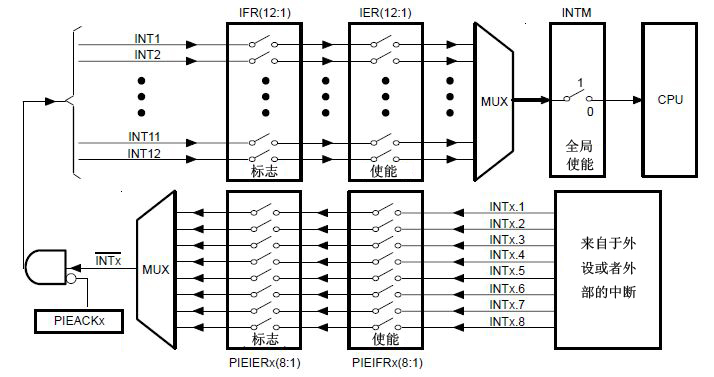
当CPU响应某个中断时，该PIEACK寄存器中中断相应位就会置1，并且一直保持，直到手动清除这个标志位。该位置1，这是如果组内发生其他的外设中断，则暂时不会被PIE控制器响应并发送给CPU(置1是相当于开关打开)，必须等到PIEACK中相应位复位之后，如果该中断请求还存在，那么PIE控制器会立刻把中断请求发送给CPU。所以每个外设中断被响应之后，一定要对PIEACK的相关位进行手动复位，以使得PIE控制器能够响应同组内其他中断。

当外设中断向PIE提出中断请求之后，PIE中断标志位会自动置1，当这时如果响应的PIEIERx相关的中断使能置位，PIEACK相应位的值为0，则PIE控制器便会将该外设中断请求提交给CPU；否则如果相应的PIEIERx相关的使能寄存器没有被置位，就是没有使能，或者PIEACK的相应位为1，则PIE控制器都暂时不会响应外设的中断请求。

在PIE级需要编程时手动处理的地方有两处：1、PIE中断的使能屏蔽，需要将对应组的使能寄存器PIEIERx的相应位置1或清零；2、PIEACK相关位的清除，以使得CPU能够响应同组内的其他中断。

外设的中断标志位是自动置位，手动清除；PIE的中断标志位是自动置位或者清除，但PIEACK需要手动置位；

**3） CPU级中断：**同外设级与PIE级相同，CPU同样也有中断标志位IFR和中断使能寄存器IER。当某一个外设中断请求通过PIE发送CPU时，CPU中断标志位IFR中对应的中断标志位INTx就会被置位。当标志位置1后，CPU不会立刻执行中断响应，而是检查IER寄存器中相关位的使能情况和CPU寄存器ST1中全局中断屏蔽位INTM的使能情况。



如果IER中的相关位被置位，并且INTM的值为0，则中断就会被响应。

CPU中断标志位的置位和清除都是自动完成的。

当CPU接到中断请求，并发现可以去响应时，就得暂停目前正在执行的程序，转而去响应中断程序，但此时，它必须做一些准备工作，以便在执行完中断程序之后回过头来还能找到原来的位置和原来的状态。CPU会将相应的IFR位进行清除，EALLOW也被清除，INTM被置位，即不能响应别的中断，等于CPU向别的中断发出通知，现在在忙，没有时间处理别的请求，等处理完该中断，再接收别的请求。

CPU 的中断优先级有高到低依次是从 INT1-INT12。每组 PIE 控制的 8 个中断优先级依次是从 INTx.1-INTx.8。

### 1.4 F28335中断向量

CPU响应中断，就是CPU要去执行相应的中断服务程序，其响应过程是CPU将现执行程序的指令地址压入堆栈，跳转到中断服务程序入口地址，中断服务程序的入口地址就是中断向量，这个中断向量用2个16位寄存器存放。入口地址是22位的，地址的低16位保存在该向量的低16位；地址的高16位则保存在它的高6位，更高的10位保留。

### 1.5 中断向量映射方式

在F28335中，中断向量表可以被映射到4个不同的存储区域，在实际应用中，F28335只能使用PIE中断向量表映射区域。中断向量表映射主要由以下型号控制。

①VMAP：该位在状态寄存器1（ST1）的第3位，复位后值为1。可以通过改变ST1值或使用SETC/CLRC VMAP指令改变VMAP的值，正常操作时该位置1。

②MOM1MAP：该位在状态寄存器1（ST1）的第11位，复位后该位置1.可以通过改变ST1的值或使用SETC/CLRC M0M1MAP指令改变M0M1MAP的值，正常操作该位置1。M0M1MAP=0是厂家测试时使用。

③ENPIE：该位在PIECTRL寄存器的第0位，复位的默认值为0（PIE被屏蔽）。器件复位后，可以通过调整PIECTRL寄存器的值进行修改。

根据上述控制位的不同设置，中断向量表有不同的映射方式，如图



## 2.中断操作

### 2.1 复位中断操作过程

PIE模块8个中断分成一组与外部中断一起共用一个CPU中断，总共有12组中断（INT1-INT12）。每组中断有相应的中断标志（PIEIFR）和使能寄存器（PIEIER），这些寄存器控制PIE向CPU申请中断。同时CPU还根据PIEIFR和PIEIER寄存器确定执行哪个中断服务程序。在清除PIEIFR和PIEIER的位时，要遵循以下3个规则。

①不要用软件编程清除PIEIFR的位

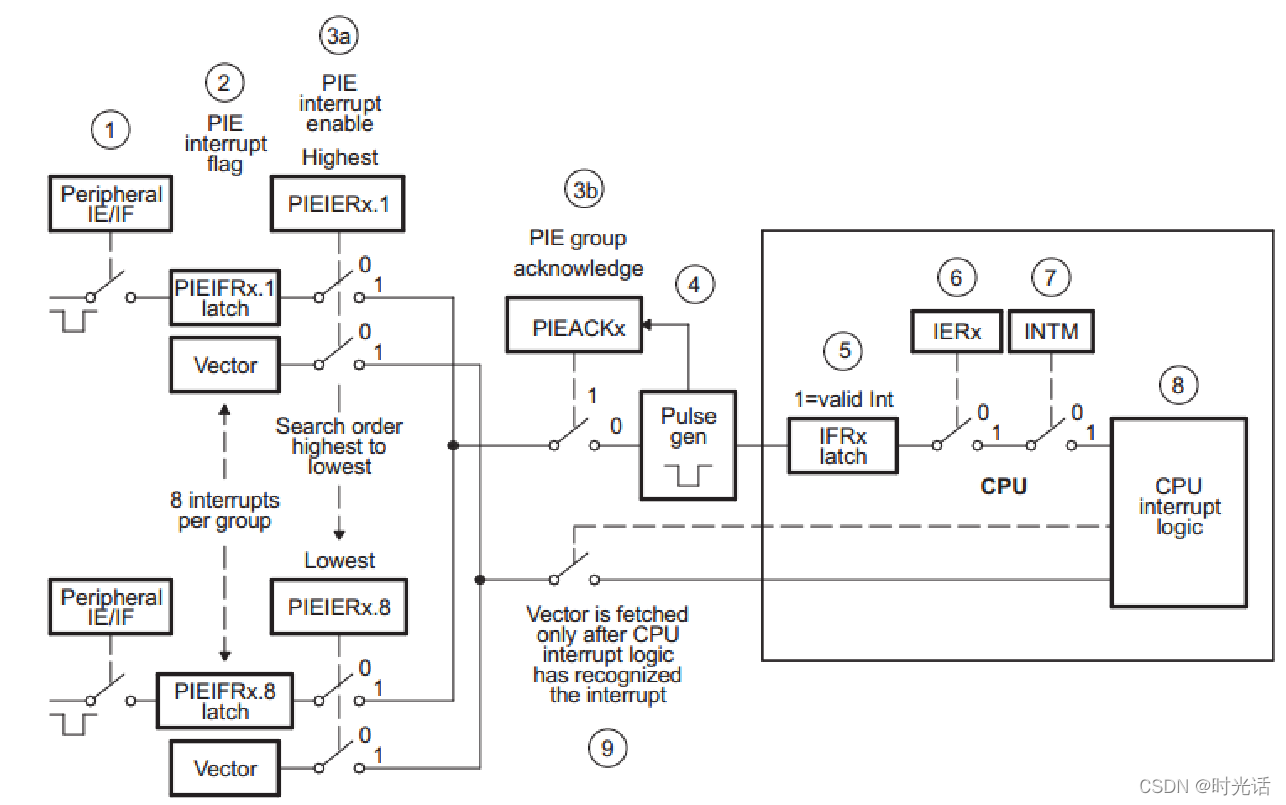
②软件设置中断优先级

③使用PIEIER禁止中断

### 2.2 使能/禁止复用外设中断的处理

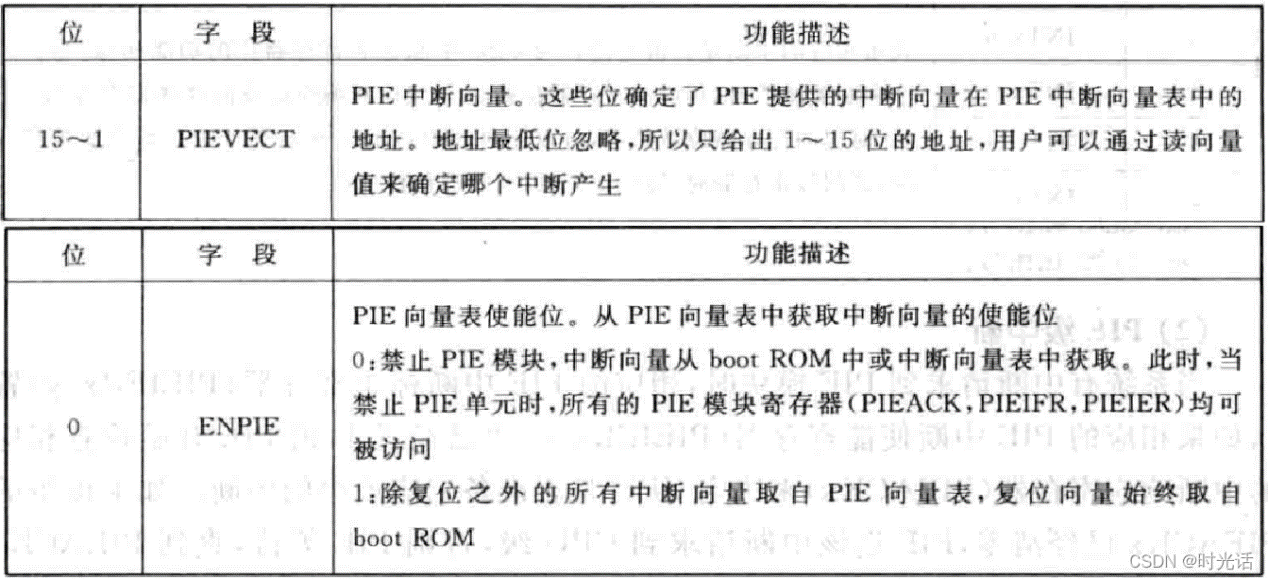
应用外设中断的使能/禁止标志位使能/禁止外设中断，PIEIER和CPU IER寄存器主要是在同一组中断内设置中断优先级。如果要修改PIEIER寄存器的设置，有两种方法。第一种方法是保护相应的PIE标志寄存器标志位，防止中断丢失。第二种方法是清除相应的PIE寄存器的标志位。

2.3 外设复用中断向CPU申请中断的流程



## 3.中断相关寄存器

### 3.1 PIE控制寄存器（PIECTRL）



## 4.中断配置

### （1）使能外设对应的PIE中断：

由于外设中断较多，它们是由PIE统一管理，所以要根据你所使用的外设中断选择对应的组，比如外部中断1，它是由PIE组1的第4线连接（查阅中断向量表），这个在前面中断介绍时讲解过。因此可由PIE控制寄存器中相应中断使能位来控制。

PieCtrlRegs.PIEIER1.bit.INTx4 = 1; // 使能PIE组1的INT4

### （2）使能外设中断：

这个具体是由外设相关中断使能位来控制，比如外部中断1，这个可由外部中断1的控制寄存器中相应中断使能位来控制。

XIntruptRegs.XINT1CR.bit.ENABLE= 1; // 使能XINT1

### （3）指定中断向量表中断服务函数地址：

这个通过对PIE中断向量表寄存器的相应位进行设置，中断服务函数名可自定义，但是要符合C语言标识符命名规则，在中断函数名前需加上地址符“&”。在对PIE中断向量表寄存器设置时要先声明EALLOW，修改完成后还要声明EDIS。比如外部中断1，其设置如下：

EALLOW; // 修改被保护的寄存器，修改前应添加EALLOW语句

PieVectTable.XINT1 = &EXTI1\_IRQn;

EDIS; // EDIS的意思是不允许修改被保护的寄存器

### （4）使能CPU中断及全局中断：

这个通过对IER和EINT寄存器相应位设置进行使能或者失能。比如外部中断1，其代码如下：

IER |= M\_INT1; // 使能CPU中断1（INT1）

EINT; // 开全局中断

### （5）编写中断服务函数

配置好中断后如果有触发，即会进入中断服务函数，中断服务函数名在前面已定义好，所以要保证一致，否则将不会进入中断服务函数内执行。在DSP28335软件开发中，要在中断服务函数名前加上关键字interrupt。例如外部中断1的中断服务函数：

interrupt void EXTI1\_IRQn(void)

{

...功能程序

}

# 二、中断操作

## 2.1 复位中断操作过程

PIE模块8个中断分成一组与外部中断一起共用一个CPU中断，总共有12组中断(INT1-INT12)。每组中断有相应的中断标志(PIEIFR)和使能寄存器(PIEIER)，这些寄存器控制PIE向CPU申请中断。同时CPU还根据PIEIFR和PIEIER寄存器确定执行哪个中断服务程序。在清除PIEIFR和PIEIER的位须遵循：

①不要用软件编程清除PIEIFR的位：清除PIEIFR寄存器的位时，有可能会使产生的中断丢失。要清除PIEIFR位时，还未被执行的中断必须被执行，如果用户希望在执行正常的服务程序之前就要清除PIEIFR时，需遵循一下步骤：

1） 设置EALLOW位为1，允许修改PIE向量表

2） 修改PIE向量表，使外设服务程序指针向量指向一个临时的ISR，这个临时的ISR只执行一个中断返回（IRET）操作。

3） 使能中断，使中断执行临时中断服务程序。

4） 在执行中断服务程序之后，PIEIFR位将被清除。

5） 修改PIE向量表，重新映射外设服务程序到正确的中断服务程序。

6） 清除EALLOW位。

CPU中断标志寄存器IFR在CPU内部，这样操作将不会影响任何向CPU申请的中断。

②软件设置中断优先级。使用 CPU IER 寄存器控制全部中断的优先级，PIEIER 寄存器控制每组中断的优先级，只有与被服务的中断在同一组时，修改 PIEIFR 寄存器的值才有意义，当 PIEACK 位保持来自 CPU 中断时，修改操作才被最终执 行。当来自无关本组的中断被执行时，禁止本组的 PIEIER 位没有意义。

③用 PIEIER 禁止中断。如果 PIEIFR 寄存器用来使能一个中断，同样可以 禁止该中断。

## 2.2 使能/禁止复用外设中断的处理

应用外设中断的使能/禁止标志位使能/禁止外设中断，PIEIER 和 CPUIER 寄存器主要是在同一组中断内设置中断优先级。如果要修改 PIEIER 寄存器的设置，有两种方法。

第一种方法是保护相应的 PIE 标志寄存器标志位，防止中断丢失。

第二种方法是清除相应的 PIE 寄存器的标志位。

方法1：

利用 PIEIERx 寄存器禁止中断保护相应的 PIEIERx 相关的标志位， 需要采取以下步骤：

1） 屏蔽全局中断（INTM=1）；

2） 清除 PIEIERx.y 位，屏蔽给定的外设中断。这样可以屏蔽同一组中断的一个或多个外设中断。

3） 等待 5 个周期，这个延时是为了保证在 CPUIFR 寄存器中产生的任何中断都能向CPU 发出申请；

4） 清除 CPUIFRx 内相应外设中断组的标志位，在 CPUIFR寄存器上采用这样的操作是比较安全的；

5） 清除相应外设中断组的 PIEACKx 寄存器位；

6） 使能全局中断（INTM=0）；

方法 2：使用 PIEIFRx 寄存器屏蔽中断并清除相应的 PIEIFRx 的标志位。为了完成外设中断的软件复位和清除 PIEIFRx 和 CPUIFR 内相应的标志位，需要采取以下的处理步骤：

1） 屏蔽全局中断（INTM=1）；

2） 设置 EALLOW 位为 1；

3） 修改 PIE 向量表，使外设服务程序指针向量指向一个临时的 ISR，这个临时的ISR只执行一个中断返回（IRET）操作，这种方法能够清楚单个中断标志位 PIEIFRx.y，而且保证不会丢失同一组内其他外设产生的中断；

4） 屏蔽外设寄存器中的外设中断；

5） 使能全局中断（INTM=0）；

6） 等待所有挂起外设中断由空的中断服务程序处理；

7） 屏蔽全局中断（INTM=1）；

8） 调整 PIE 向量表，将外设中断向量映射到原来的中断服务程序；

9） 清除 EALLOW 位；

10）屏蔽给定的 PIEIER 位；

11）清除给定外设中断组的标志位 IFR（对 CPU IFR 寄存器操作比较安全）；

12）清除 PIE 组的 PIEACK 位；

13）使能全局中断；

## 2.3用中断向CPU申请中断的流程

外设复用中断向 CPU 申请中断的流程，如下图所示：

1） 任何一个 PIE 中断组的外设或外部中断产生中断。如果外设模块内的中断被使能，中断请求将被送到 PIE 模块。

2） PIE 模块将识别出 PIE 中断组 x 内的 y 中断（INTx.y）申请，然后相应的 PIE 中断标志位被锁存：PIEIFRx.y=1。

3） PIE 的中断如要送到 CPU 需满足下面两个条件： ①相应的使能位必须被设置（PIEIFRx.y=1）。 ②相应的 PIEACKx 位必须被清除。

4） 如果满足步骤 3 中的两个条件，中断请求将被送到 CPU 并且相应的相应寄存器位被设置 1（PIEACKx=1）。PIEACKx 位将保持不变，除非为了使本组中的其他中断向 CPU 发出申请而清除该位。

5） CPU中断标志位被置位（CPU IFRx=1），表明产生一个 CPU 级的挂起中断。

6） 如果 CPU 中断被使能（CPU IERx=1 或者 DBGIERx=1），并且全局中断使能（INTM=0），CPU 将处理中断 INTx。

7） CPU 识别到中断并自动保存相关的中断信息，清除使能寄存器（IER）位， 设置 INTM，清除 EALLOW。CPU 完成这些任务准备执行中断服务程序。

8） CPU 从 PIE 中获取响应的中断向量。

9） 对于复用中断，PIE 模块用 PIEIERx 和 PIEIFRx 寄存器中的值确定响应中断的向量地址。有以下两种情况： ①在步骤 4 中若有更高优先级的中断产生，并使能了 PIEIERx 寄存器，且 PIEIERx 的相应位处于挂起状态，则首先响应优先级高的中断。 ②如果在本组内没有挂起的中断被使能，PIE 将响应组内优先级最高的中断， 调转地址使用 INTx.1。这种操作相当于处理器的 TRAP 或 INT 指令。 CPU 进入中断服务程序后，将清除 PIEIFRx.y 位。需要说明的是，PIEIFRx 寄存器用来确定中断向量，在清除 PIEIFRx 寄存器时必须注意。

## 2.4 可屏蔽中断处理

按照是否可以被屏蔽，可将中断分为两大类：不可屏蔽中断（又叫非屏蔽中 断，NMI）和可屏蔽中断。不可屏蔽中断源一旦提出中断请求，CPU 必须无条件 响应；而对可屏蔽中断源的请求，CPU 可以响应，也可以不响应。对于可屏蔽中断，除了受本身的屏蔽位控制外，还都要受一个总的控制，即 CPI 标志寄存器中 的中断允许标志位 IF 的控制。IF 位为 1，可以得到 CPU 响应，否则得不到响应。 IF 位可以由用户控制。

可屏蔽中断的响应过程实质上就是中断的产生、使能到处理过程，其结构如 下图所示。包括两个部分设置：CPU 级中断设置（INT1-INT4）和 PIE 级中断设 置（INTx.1-INTx.8）

### 1） CPU级中断

①CPU 级中断标志设置。除了系统初始化，一般不建议改变标志寄存器的状 态，否则可能清除某些有用的中断信息或者产生意外中断。但有时也可能希望通 过软件方式使某些中断标志位置位或者清零，在这种情况下可以通过以下 2 条指 令完成。

IFR|=0X0008;//INT4 位置位

IFR|=0XFFF7;//INT4 位清零

如果在清除中断标志寄存器中的某些状态位时刚好有中断产生，则此时中断 有更高的优先级，相应的标志位仍然为 1.在系统复位和 CPU 相应中断后，中断 标志位将自动清零。

②CPU 级中断使能。CPU 级中断使能寄存器的 16 位分别控制每个中断的使能 状态。当相应的位置 1 时，使能中断，写 0 禁止中断。在 C 语言中可采用下面代 码实现中断的控制，系统复位禁止所有中断。

IER|=0X0008;//使能中断 INT4

IER|=0XFFF7;//禁止中断 INT4

③全局中断使能。状态寄存器 ST1 的位 0（INTM）为全局中断使能控制位， 当该位等于 0 时全局中断使能，当该位等于 1 时禁止所有中断。CPU 要实现中断 处理必须在产生中断的前提下，相应的 IER 寄存器位使能并且需要全局使能位使能。可以采用以下代码实现全局中断使能控制。

asm(" clrc INTM");

asm(" setc INTM");

### 2） PIE级中断

当系统有中断请求到 PIE 模块时，相应的 PIE 中断标志寄存器（PIEIFRx.y） 置 1，如果相应的 PIE 中断使能寄存器（PIEIFRx.y）也已经置 1，则 PIE 开始检 查相应的中断响应寄存器（PIEACKx）来决定 CPU 是准备响应该组的中断。如果 该组的 PIEACKx 已经清零，PIE 送该中断请求到 CPU 级，否则 PIE 等待，直到 PIEACKx 已经清零，才送中断请求到 CPU 级。

## 2.5 不可屏蔽中断处理

不可屏蔽中断设置简单得多，以 XNMI\_XINT13 引脚外部中断设置为例，只要配置好相应的引脚配置寄存器就可以，因为 NMI 优先级最高，CPU 必须响应，无需配置 CPU。

# 三 代码实现

使用外部中断1功能通过按键K1控制D2亮灭，K4按键控制D3亮灭，D1指示灯闪烁提示系统运行状态。

1. void EXTI1\_Init(void)
2. {
3. EALLOW;
4. SysCtrlRegs.PCLKCR3.bit.GPIOINENCLK = 1;    *// GPIO input clock*
5. EDIS;
7. EALLOW;
8. *//KEY端口配置*
9. GpioCtrlRegs.GPAMUX1.bit.GPIO12=0;
10. GpioCtrlRegs.GPADIR.bit.GPIO12=0;
11. GpioCtrlRegs.GPAPUD.bit.GPIO12=0;
12. GpioCtrlRegs.GPAQSEL1.bit.GPIO12 = 0;        *// 外部中断1（XINT1）与系统时钟SYSCLKOUT同步*
14. GpioCtrlRegs.GPBMUX2.bit.GPIO48=0;
15. GpioCtrlRegs.GPBDIR.bit.GPIO48=1;
16. GpioCtrlRegs.GPBPUD.bit.GPIO48=0;
17. GpioDataRegs.GPBCLEAR.bit.GPIO48=1;
18. EDIS;
20. EALLOW;
21. GpioIntRegs.GPIOXINT1SEL.bit.GPIOSEL = 12;   *// XINT1是GPIO12*
22. EDIS;
24. EALLOW; *// 修改被保护的寄存器，修改前应添加EALLOW语句*
25. PieVectTable.XINT1 = &EXTI1\_IRQn;
26. EDIS;   *// EDIS的意思是不允许修改被保护的寄存器*
28. PieCtrlRegs.PIEIER1.bit.INTx4 = 1;          *// 使能PIE组1的INT4*
30. XIntruptRegs.XINT1CR.bit.POLARITY = 0;      *// 下降沿触发中断*
31. XIntruptRegs.XINT1CR.bit.ENABLE= 1;        *// 使能XINT1*
33. IER |= M\_INT1;                              *// 使能CPU中断1（INT1）*
34. EINT;                                       *// 开全局中断*
35. ERTM;
36. }
38. interrupt void EXTI1\_IRQn(void)
39. {
40. Uint32 i;
41. for(i=0;i<10000;i++);    *//键盘消抖动*
42. while(!KEY\_H1);
43. LED2\_TOGGLE;
44. PieCtrlRegs.PIEACK.bit.ACK1=1;
45. }

48. void EXTI2\_Init(void)
49. {
50. EALLOW;
51. SysCtrlRegs.PCLKCR3.bit.GPIOINENCLK = 1;    *// GPIO input clock*
52. EDIS;
54. EALLOW;
55. *//KEY端口配置*
56. GpioCtrlRegs.GPAMUX1.bit.GPIO13=0;
57. GpioCtrlRegs.GPADIR.bit.GPIO13=0;
58. GpioCtrlRegs.GPAPUD.bit.GPIO13=0;
59. GpioCtrlRegs.GPAQSEL1.bit.GPIO13 = 2;        *// 外部中断2（XINT2）输入限定6个采样窗口*
60. GpioCtrlRegs.GPACTRL.bit.QUALPRD1 = 0xFF;   *// 每个采样窗口的周期为510\*SYSCLKOUT*
62. GpioCtrlRegs.GPBMUX2.bit.GPIO48=0;
63. GpioCtrlRegs.GPBDIR.bit.GPIO48=1;
64. GpioCtrlRegs.GPBPUD.bit.GPIO48=0;
65. GpioDataRegs.GPBCLEAR.bit.GPIO48=1;
66. EDIS;
68. EALLOW;
69. GpioIntRegs.GPIOXINT2SEL.bit.GPIOSEL = 13;   *// XINT2是GPIO13*
70. EDIS;
72. EALLOW; *// 修改被保护的寄存器，修改前应添加EALLOW语句*
73. PieVectTable.XINT2 = &EXTI2\_IRQn;
74. EDIS;   *// EDIS的意思是不允许修改被保护的寄存器*
76. PieCtrlRegs.PIEIER1.bit.INTx5 = 1;          *// 使能PIE组1的INT5*
78. XIntruptRegs.XINT2CR.bit.POLARITY = 0;      *// 下降沿触发中断*
79. XIntruptRegs.XINT2CR.bit.ENABLE = 1;        *// 使能XINT2*
81. IER |= M\_INT1;                              *// 使能CPU中断1（INT1）*
82. EINT;                                       *// 开全局中断*
83. ERTM;
84. }
86. interrupt void EXTI2\_IRQn(void)
87. {
88. Uint32 i;
89. for(i=0;i<10000;i++);    *//键盘消抖动*
90. while(!KEY\_H2);
91. LED3\_TOGGLE;
92. PieCtrlRegs.PIEACK.bit.ACK1=1;
93. }

主函数

1. //---------------------------------------------------------------------------
2. / / InitPieCtrl:
3. //---------------------------------------------------------------------------
4. //这个函数初始化PIE控制寄存器到一个已知的状态。
5. //
6. void InitPieCtrl(void)
7. {
8. // Disable Interrupts at the CPU level:
9. DINT;
11. // Disable the PIE
12. PieCtrlRegs.PIECTRL.bit.ENPIE = 0;
14. // Clear all PIEIER registers:
15. PieCtrlRegs.PIEIER1.all = 0;
16. PieCtrlRegs.PIEIER2.all = 0;
17. PieCtrlRegs.PIEIER3.all = 0;
18. PieCtrlRegs.PIEIER4.all = 0;
19. PieCtrlRegs.PIEIER5.all = 0;
20. PieCtrlRegs.PIEIER6.all = 0;
21. PieCtrlRegs.PIEIER7.all = 0;
22. PieCtrlRegs.PIEIER8.all = 0;
23. PieCtrlRegs.PIEIER9.all = 0;
24. PieCtrlRegs.PIEIER10.all = 0;
25. PieCtrlRegs.PIEIER11.all = 0;
26. PieCtrlRegs.PIEIER12.all = 0;
28. // Clear all PIEIFR registers:
29. PieCtrlRegs.PIEIFR1.all = 0;
30. PieCtrlRegs.PIEIFR2.all = 0;
31. PieCtrlRegs.PIEIFR3.all = 0;
32. PieCtrlRegs.PIEIFR4.all = 0;
33. PieCtrlRegs.PIEIFR5.all = 0;
34. PieCtrlRegs.PIEIFR6.all = 0;
35. PieCtrlRegs.PIEIFR7.all = 0;
36. PieCtrlRegs.PIEIFR8.all = 0;
37. PieCtrlRegs.PIEIFR9.all = 0;
38. PieCtrlRegs.PIEIFR10.all = 0;
39. PieCtrlRegs.PIEIFR11.all = 0;
40. PieCtrlRegs.PIEIFR12.all = 0;

43. }
45. //---------------------------------------------------------------------------
46. / / InitPieVectTable:
47. //---------------------------------------------------------------------------
48. //这个函数初始化一个已知状态的PIE向量表。
49. //该函数必须在启动后执行。
50. //
51. void InitPieVectTable(void)
52. {
53. int16 i;
54. Uint32 \*Source = (void \*) &PieVectTableInit;
55. Uint32 \*Dest = (void \*) &PieVectTable;
57. EALLOW;
58. for(i=0; i < 128; i++)
59. \*Dest++ = \*Source++;
60. EDIS;
62. // Enable the PIE Vector Table
63. PieCtrlRegs.PIECTRL.bit.ENPIE = 1;
65. }
66. /**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****\*\*\**\*
67. \* 函 数 名         : main
68. \* 函数功能     : 主函数
69. \* 输    入         : 无
70. \* 输    出         : 无
71. **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****\*\*\**\*/
72. void main()
73. {
74. int i=0;
76. InitSysCtrl();
78. InitPieCtrl();
79. IER = 0x0000;
80. IFR = 0x0000;
81. InitPieVectTable();
83. LED\_Init();
84. EXTI1\_Init();
85. EXTI2\_Init();
87. while(1)
88. {
89. i++;
90. if(i%2000==0)
91. {
92. LED1\_TOGGLE;
93. }
94. DELAY\_US(100);
95. }
96. }