第10章 F28035的中断系统

本章内容

本章重点

1、理解中断系统概念；

2、掌握F28035的CPU中断；

3、掌握中断向量和中断优先级；

4、了解CPU中断寄存器；

5、掌握F28035三级中断系统。

10.1 中断系统的介绍

中断系统：中断是硬件和软件驱动事件，它能够让CPU停止当前的程序，转而去执行一个中断服务子程序。在执行完中断子程序后，会返回到原来中断的位置，然后继续按顺序执行后面的程序。中断程序执行示意图如图1所示。

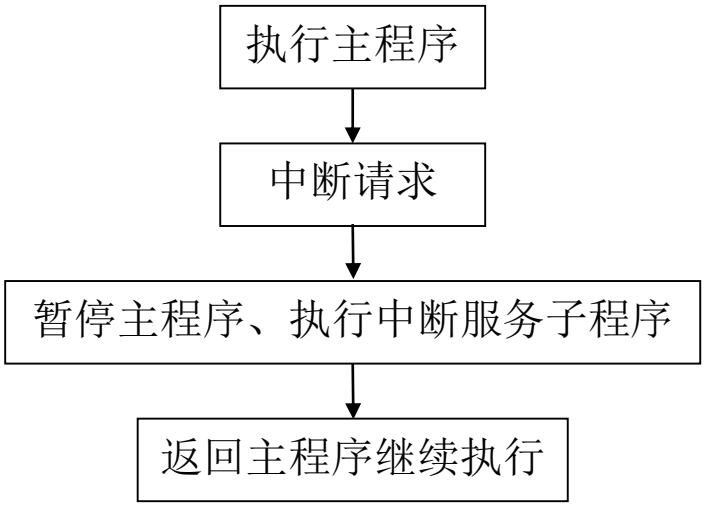
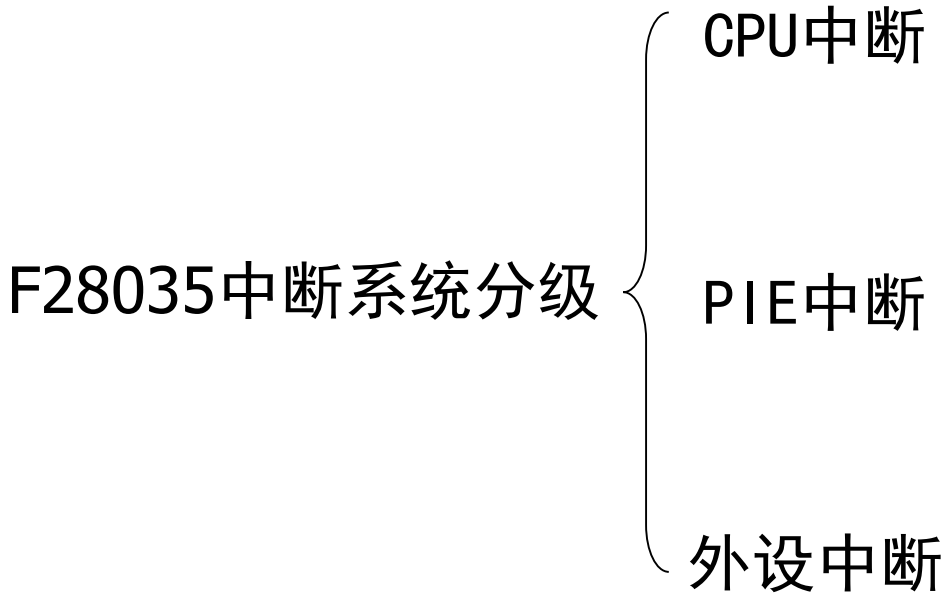


图1 中断系统工作过程

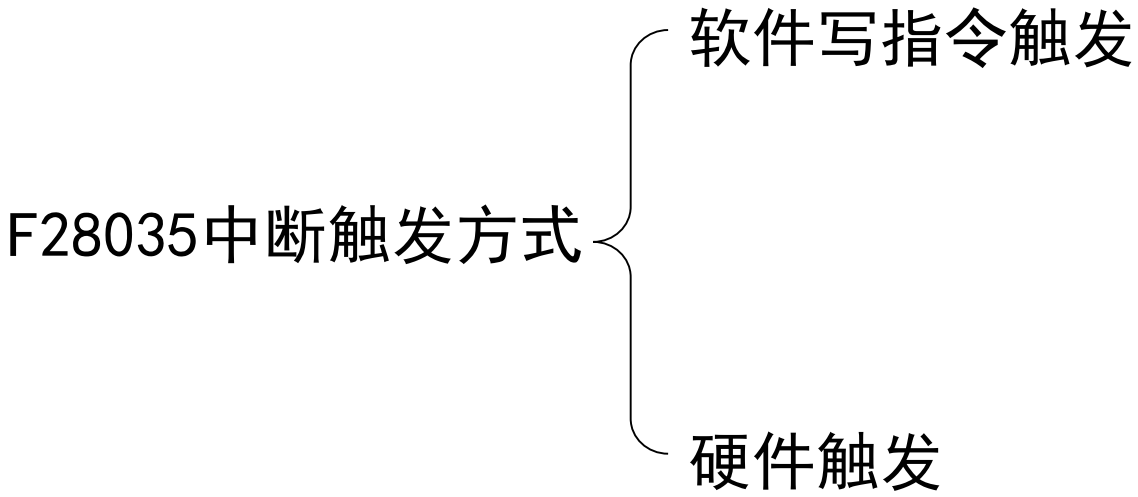
10.1 中断系统的介绍



10.2　F28035的CPU中断

F28035中断概述：DSP中的中断请求信号一般是由软件或者硬件产生的，它能够使CPU暂停正在执行的主程序，转而去执行一个中断服务子程序，通常中断请求信号是由外围设备发出的，表示一个特殊的事件发生了，请求CPU暂停正在执行的主程序，去响应更为紧急的事件。

10.2.1 CPU中断的概述



F28035中断的触发方式又可以归结为可屏蔽中断和不可屏蔽中断。

10.2.1 CPU中断的概述

可屏蔽中断：可以通过软件加以屏蔽或者解除屏蔽。F28035片内外设所产生的中断都是可屏蔽中断，每一个中断都可以通过相应寄存器的中断使能位来禁止或者使能中断。

不可屏蔽中断：中断是不可被屏蔽的，一旦有中断请求发出，CPU必须无条件的立即去相应该中断并执行相应的中断服务子程序。

10.2.1 CPU中断的概述

F28035处理中断分为以下四步，如图2所示。

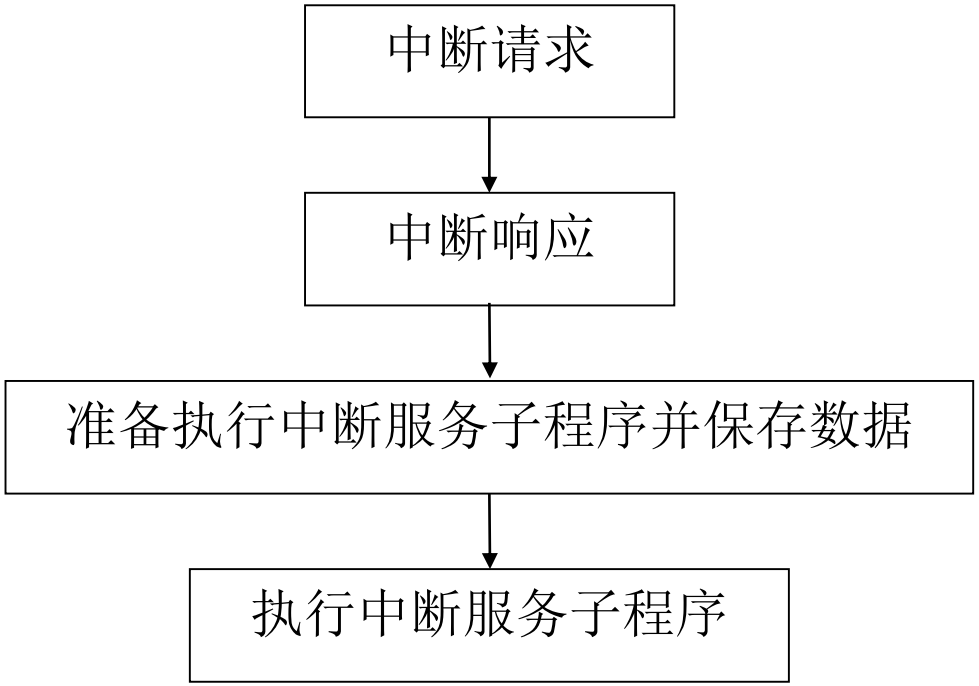


图2 CPU处理中断的4个步骤

10.2.2 CPU中断向量和优先级

在程序执行过程中，当遇到多个中断时，就需要根据中断源的优先级来判断执行顺序，即中断级高的先处理，中断级低的后处理。

F28035一共可以支持32个CPU中断，每一个都是一个32位的中断向量，里面存储的是相应中断服务子程序的入口地址，不过这个入口地址是22位的。其中低16位保存该向量的低16位；高16位保存它的高6位，其余的10位忽略，如图3所示。

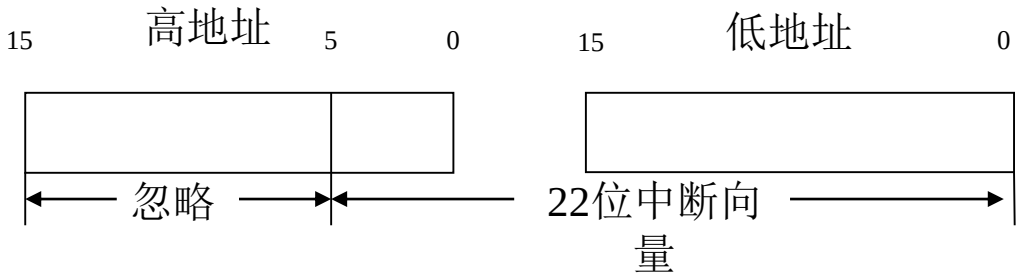


图3 22位的CPU中断向量

10.2.2 CPU中断向量和优先级

详细F28035可以使用的中断向量、各个向量的存储位置以及其各自的优先级见表1所示或者见教材表10-1所示。

10.2.2 CPU中断向量和优先级

表1　CPU中断向量和优先级

中断向量	地址	优先级	说明
Reset	0x0000 0D00	1（最高）	复位中断
INT1	0x0000 0D02	5	可屏蔽中断1
INT2	0x0000 0D04	6	可屏蔽中断2
INT3	0x0000 0D06	7	可屏蔽中断3
INT4	0x0000 0D08	8	可屏蔽中断4
INT5	0x0000 0D0A	9	可屏蔽中断5
INT6	0x0000 0D0C	10	可屏蔽中断6
INT7	0x0000 0D0E	11	可屏蔽中断7
INT8	0x0000 0D10	12	可屏蔽中断8
INT9	0x0000 0D12	13	可屏蔽中断9
INT10	0x0000 0D14	14	可屏蔽中断10
INT11	0x0000 0D16	15	可屏蔽中断11
INT12	0x0000 0D18	16	可屏蔽中断12
INT13	0x0000 0D1A	17	可屏蔽中断13
INT14	0x0000 0D1C	18	可屏蔽中断14
DATALOG	0x0000 0D1E	19（最低）	可屏蔽数据标志中断
RTOSINT	0x0000 0D20	4	可屏蔽实时操作系统中断
EMUINT	0x0000 0D22	2	可屏蔽仿真中断
NMI	0x0000 0D24	3	不可屏蔽硬件中断
ILLEGAL	0x0000 0D26		非法指令捕获
USER1	0x0000 0D28		用户自定义陷阱（TRAP）
USER2	0x0000 0D2A		用户自定义陷阱（TRAP）
USER3	0x0000 0D2C		用户自定义陷阱（TRAP）
USER4	0x0000 0D2E		用户自定义陷阱（TRAP）
USER5	0x0000 0D30		用户自定义陷阱（TRAP）
USER6	0x0000 0D32		用户自定义陷阱（TRAP）
USER7	0x0000 0D34		用户自定义陷阱（TRAP）
USER8	0x0000 0D36		用户自定义陷阱（TRAP）
USER9	0x0000 0D38		用户自定义陷阱（TRAP）
USER10	0x0000 0D3A		用户自定义陷阱（TRAP）
USER11	0x0000 0D3C		用户自定义陷阱（TRAP）
USER12	0x0000 0D3E		用户自定义陷阱（TRAP）

10.2.3 CPU中断寄存器

CPU所列的中断里INT1-INT14是14个通用中断，DLOGINT数据标志中断和RTOSINT实时操作系统中断是为仿真而设计的两个中断。经常使用的通用中断是INT1-INT14。

IER寄存器各位的情况见图4所示，各位的描述见教材表10-2所示。从图中可以看到CPU中断使能寄存器中的每一位都与一个CPU中断相对应，当某一个位的值为1，与其对应的中断就被使能；当某一个位的值为0，与其对应的中断就被关闭。

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

图4　IER寄存器

10.2.3 CPU中断寄存器

在DSP中，有一个CPU中断的标志寄存器IFR与其相似，这个寄存器中位的情况即反映为该CPU中断是否提出了请求。

IFR寄存器各位的情况见图5所示，各位的描述见教材表10-3所示。

15	14	13	12	11	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT11	INT10	INT9
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

图5 IFR寄存器

10.3 F28035的PIE中断

PIE中断概述：F2803x设备有许多外设并且每个外设在外设级可能产生一个或者多个中断，因此，为了更好的处理这些中断，专门设计了一个外设中断扩展模块,即PIE，它能够对各种中断源做出相应的决策。

PIE模块能够支持96个独立的中断，这些中断分为12个组，每一组8个中断。同时，每个组都被反应到CPU内核的12条中断线上。详细的外设中断在PIE中的分布情况见教材中表10-4所示。

10.3.1 PIE中断概述

PIE中中断优先级的判断：从表10-4中，中断优先级的判断如下，INT1的优先级比INT2高，剩下的依次类推；对于PIE同组内的各个中断，INTx.1比INTx.2高，INTx.2比INTx.3高，剩下的依次类推；对于不同组之间，排在前面组内的任何一个中断优先级要比排在后面组内的任何一个中断的优先级高。

10.3.2 PIE中断寄存器

PIE控制器相关的寄存器如表2所示，或见教材表10-5所列。

表2 PIE控制器的寄存器

名称	地址	大小（×16）	说明
PIECTRL	0x0000-0CE0	1	PIE控制寄存器
PIEACK	0x0000-0CE1	1	PIE应答寄存器
PIEIER1	0x0000-0CE2	1	PIE，INT1组使能寄存器
PIEIFR1	0x0000-0CE3	1	PIE，INT1组标志寄存器
PIEIER2	0x0000-0CE4	1	PIE，INT2组使能寄存器
PIEIFR2	0x0000-0CE5	1	PIE，INT2组标志寄存器
PIEIER3	0x0000-0CE6	1	PIE，INT3组使能寄存器
PIEIFR3	0x0000-0CE7	1	PIE，INT3组标志寄存器
PIEIER4	0x0000-0CE8	1	PIE，INT4组使能寄存器
PIEIFR4	0x0000-0CE9	1	PIE，INT4组标志寄存器
PIEIER5	0x0000-0CEA	1	PIE，INT5组使能寄存器
PIEIFR5	0x0000-0CEB	1	PIE，INT5组标志寄存器
PIEIER6	0x0000-0CEC	1	PIE，INT6组使能寄存器
PIEIFR6	0x0000-0CED	1	PIE，INT6组标志寄存器
PIEIER7	0x0000-0CEE	1	PIE，INT7组使能寄存器
PIEIFR7	0x0000-0CEF	1	PIE，INT7组标志寄存器
PIEIER8	0x0000-0CF0	1	PIE，INT8组使能寄存器
PIEIFR8	0x0000-0CF1	1	PIE，INT8组标志寄存器
PIEIER9	0x0000-0CF2	1	PIE，INT9组使能寄存器
PIEIFR9	0x0000-0CF3	1	PIE，INT9组标志寄存器
PIEIER10	0x0000-0CF4	1	PIE，INT10组使能寄存器
PIEIFR10	0x0000-0CF5	1	PIE，INT10组标志寄存器
PIEIER11	0x0000-0CF6	1	PIE，INT11组使能寄存器
PIEIFR11	0x0000-0CF7	1	PIE，INT11组标志寄存器
PIEIER12	0x0000-0CF8	1	PIE，INT12组使能寄存器
PIEIFR12	0x0000-0CF9	1	PIE，INT12组标志寄存器

10.3.2 PIE中断寄存器

PIE控制寄存器（PIECTRL）各位的情况见图6所示，各位描述见教材表10-6所示。

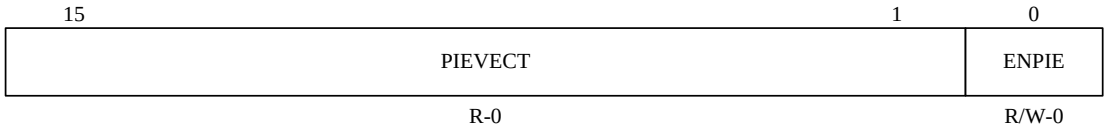


图6　PIE控制寄存器

10.3.2 PIE中断寄存器

PIE中断应答寄存器各位的情况见图7所示，各位描述见教材表10-7所示。

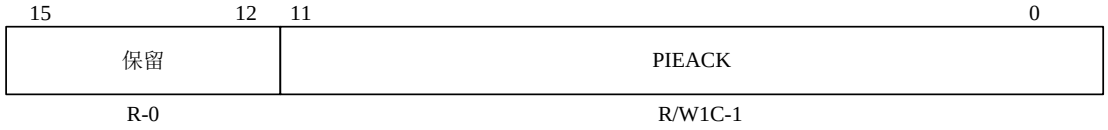


图7　PIE中断应答寄存器

10.3.2 PIE中断寄存器

PIE中断标志寄存器各位的情况见图8所示，各位描述见教材表10-8所示。

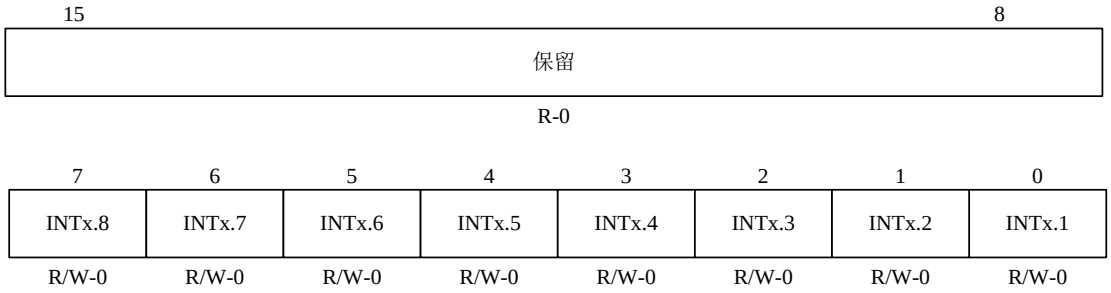


图8　PIE中断标志寄存器

10.3.2 PIE中断寄存器

PIE中断使能寄存器各位的情况见图9所示，各位描述见教材表10-9所示。

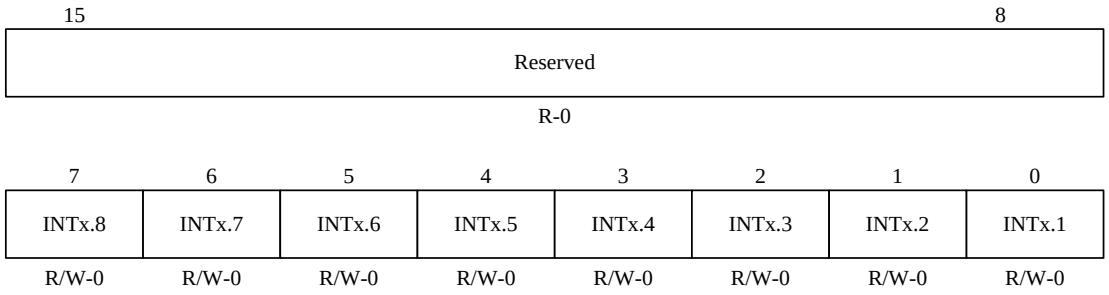


图9　PIE中断使能寄存器

10.3.3 中断向量表

PIE中断向量表的构成：PIE中断向量表是由各个中断服务子程序的地址存储在一片连续的RAM空间内所构成的。详细的中断向量表见教材表10-10所示。

注：如果DSP芯片复位，在没有初始化PIE前，使用的是BROM向量。因此，在DSP复位和程序引导完成之后，必须对PIE向量表进行初始化，然后由应用程序使能PIE向量表。这样CPU在响应中断时，才能取出中断服务子程序的地址。

10.4 F28035的三级中断系统

F2803x的中断为三级中断机制，分为外设级、PIE级和CPU级。当一个中断发生后，若其中任意一级不允许，该中断都不会执行。中断操作顺序如图4所示。

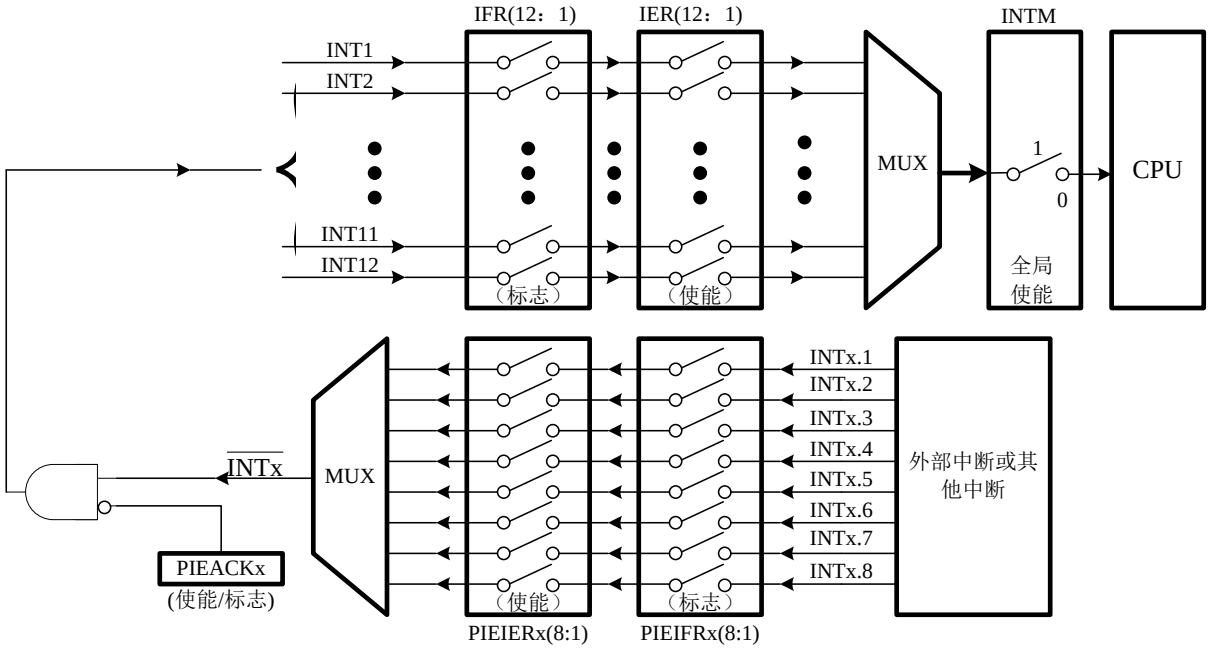


图4 中断操作顺序图

10.4 F28035的三级中断系统

- 外设级

当外设产生了一个中断，那么与该中断对应的相关中断标志位将被置为1，此时若该中断的中断使能被置位，则外设会向PIE发送一个中断请求；若该中断使能没有被置位，则不会发送中断请求。但与其对应的中断标志位不会清除，直到用软件将其清除。

- PIE级

和外设的处理情况相似，当PIEIFR和PIEIER被置位，同时对应中断响应寄存器为0，则产生的中断会被送到CPU，此时中断响应寄存器为1，若不清除，则无法响应该组其他中断。这里的清除中断标志的方法是写入1，而不是写入0。

10.4　F28035的三级中断系统

- CPU级

和外设和PIE级相似，当IFR和IER被置位，同时全局中断被打开，则CPU会响应产生的中断。反之，则不会。

10.5 LED灯不同频率闪烁

程序模块编写如下：

（1）初始化GPIO

```
void Gpio_LedInit()
{
      EALLOW;
      GpioCtrlRegs.GPBMUX1.bit.GPIO40=0;//将GPIO40配置为通用I/O口
      GpioCtrlRegs.GPBDIR.bit.GPIO40=1;//将GPIO40方向变为输出
      GpioCtrlRegs.GPAMUX2.bit.GPIO26=0;//GPIO40配置为通用的IO口
      GpioCtrlRegs.GPADIR.bit.GPIO26=1;//配置为输出
      EDIS;
}
```

10.5 LED灯不同频率闪烁

（2）初始化定时器

```
void CpuTimer_IntHandlerConfig()
{
        EALLOW;
        PieVectTable.TINT0 = &Timer0_IsrHandler;
        PieVectTable.TINT1=&Timer1_IsrHandler;
        EDIS;
}
```

10.5 LED灯不同频率闪烁

```
void CpuTimer_CpuIntEn()
{
      PieCtrlRegs.PIECTRL.bit.ENPIE = 1;
      IER |= M_INT1;  //使能定时器T0中断
      IER|=M_INT13;  //INT13连接到定时器1
      PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
      EINT;
      ERTM;        //使能全局中断
}
```

10.5 LED灯不同频率闪烁

```
void CPU_TimerInit()
{
        CpuTimer_IntHandlerConfig();
        InitCpuTimers();
       ConfigCpuTimer(&CpuTimer0, 60, 1000000); //定时器0实现1S定时，
周期为2s
        ConfigCpuTimer(&CpuTimer1,60,2000000); //定时器1实现2s定时，
周期为4s
        CpuTimer0Regs.TCR.bit.TSS = 0; // 启动定时器0(TSS = 0)
        CpuTimer1Regs.TCR.bit.TSS=0;     //启动定时器1
        CpuTimer_CpuIntEn();//开中断
}
```

10.5 LED灯不同频率闪烁

（3）配置定时器0中断内容

```
interrupt void Timer0_IsrHandler()
{
      GpioDataRegs.GPATOGGLE.bit.GPIO26=1; //电平翻转
      PieCtrlRegs.PIEACK.bit.ACK1=1;  //清除中断标志
}
```

（4）配置定时器1中断的内容

```
interrupt void Timer1_IsrHandler()
{
      GpioDataRegs.GPBTOGGLE.bit.GPIO40=1;//电平翻转
      PieCtrlRegs.PIEACK.bit.ACK1=1;   //清除中断标志
}
```

10.5 LED灯不同频率闪烁

（5）主函数

```
#include "DSP28x_Project.h"
void main(void)
{
        InitSysCtrl();
        DINT;                          //先禁止CPU中断
        InitPieCtrl();//给PIE寄存器赋初值
        IER = 0x0000; //禁止CPU的中断并清楚所有相关的中断标志
        IFR = 0x0000;
        InitPieVectTable();//初始化中断向量表
        Gpio_LedInit();
        CPU_TimerInit();//定时器初始化
        while(1);
}
```