F28335的中断系统

F28335的中断系统

如果接触过单片机，应该会知道中断这个词汇。在任何一款事件驱动型的CPU里面都应该会有中断系统，因为中断就是为响应某种事件而存在的。中断的灵活应用不仅能够实现想要实现的功能，而且合理的中断安排可以提高事件执行的效率，因此中断在DSP应用中的地位是非常重要的。本章就详细介绍F28335的中断系统，共同探讨CPU中断、PIE中断、外设中断的三级中断体系，并介绍如何正确编写外设的中断程序，以保证中断的正确执行。

什么是中断

中断(Interrupt)是硬件和软件驱动事件，它使得CPU暂停当前的主程序，并转而去执行一个中断服务子程序。为了更好更形象的理解中断，下面以办公时接电话为例来阐述一下中断的概念，可以通过这个例子来体会一下CPU执行中断时候的一些原理。

什么是中断

假如一个工程师正在办公桌前专心致志的写程序，突然电话铃声响了(很显然，电话是不可错过的，相比手中写程序的活，这个电话肯定是更加重要和紧急的，电话事件相当于产生了一个中断请求，因为某种需求不得不请求这个工程师打断手中正在做的事情)。工程师听到铃声便拿起电话进行交谈(工程师响应了电话的请求，相当于CPU响应了一个中断，停下了正在执行的主程序，并转向执行中断服务子程序)。电话很快就讲完了，工程师挂上了电话，又接着从刚才停下来的地方开始写程序(中断服务子程序执行完成之后，CPU又回到了刚才停下来的地方开始执行主程序)。整个过程如图10-1所示。

什么是中断

图10-1 中断的生活实例

什么是中断

当然，CPU执行中断的时候肯定要比接电话的例子复杂的多，但是通过这个简单的生活实例，希望能够比较感性的理解什么是中断，以及中断产生时CPU是如何去执行一些步骤的。F28335的中断系统从上至下分成了三级，即CPU级中断、PIE级中断和外设中断。下面先从上至下分别详细介绍各级中断，然后再从下至上并结合实例分析CPU三级中断的工作过程。

F28335的CPU中断·CPU中断的概述

F28335的中断主要由两种方式触发。一种是通过在软件中写指令，例如INTR、OR IFR或者TRAP指令。另一种是硬件方式触发，例如来自于片内外设，或者外围设备的中断信号，表示某个事件已经发生。无论是软件中断，还是硬件中断，都可以归结为可屏蔽中断和不可屏蔽中断。

所谓可屏蔽中断就是这些中断可以用软件加以屏蔽或者解除屏蔽。F28335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该中断。

F28335的CPU中断·CPU中断的概述

不可屏蔽中断就是这些中断是不可以被屏蔽的，一旦中断申请信号发出，CPU必须无条件的立即去响应该中断并执行相应的中断服务子程序。F28335的不可屏蔽中断主要包括软件中断(INTR指令和TRAP指令等)、硬件中断 、非法指令陷阱以及硬件复位中断。由于平时遇到最多的还是可屏蔽中断，所以这里不可屏蔽中断除了硬件中断 以外就不多做介绍了。通过XNMI输入选择寄存器GPIOXNMISEL可以进行不可屏蔽中断$\overline{\text{NMI}}$的中断源设置，当相应引脚为低电平时，CPU就可以检测到一个有效的中断请求，从而会响应$\overline{\text{NMI}}$中断。

F28335的CPU中断·CPU中断的概述

F28335的CPU按照图10-2所示的四个步骤来处理中断。首先由外设或者其他方式向CPU提出中断请求，然后如果这个中断是可屏蔽中断，CPU便会去检查这个中断的使能情况，再决定是否响应该中断，如果这个中断是不可屏蔽中断，则CPU便会立即响应该中断。接着，CPU会完整的执行完当前指令，为了记住当前主程序的状态，CPU必须要做一些准备工作，例如将ST0、T、AH、AL、PC等寄存器的内容保存到堆栈中，以便自动保存主程序的大部分内容。在准备工作做完之后，CPU就取回中断向量，开始执行中断服务子程序。当然，处理完相应的中断事件之后，CPU就回到原来的主程序暂停的地方，恢复各个寄存器的内容，继续执行主程序。

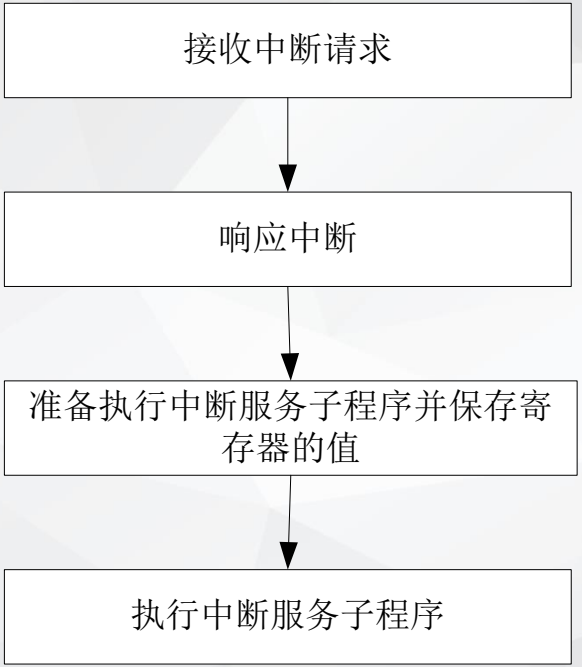


图10-2 CPU处理中断的四个步骤

F28335的CPU中断·CPU中断的概述

上面讲解的是单个中断请求的处理过程，那要是几个中断同时向CPU发出中断请求，CPU该如何处理呢？举个简单的例子，假如有一个医生但是有两个病人需要急诊，一个出了车祸，性命攸关，而另一个只是普通的感冒，这时候医生会先诊治哪个病人呢？很显然，医生肯定会先救治出了车祸的病人，因为从紧急的程度来讲，出了车祸的肯定要比感冒病人来的紧急的多。DSP的CPU就像是这个医生，不同的中断就像是一个个急需救治的病人，每一个CPU中断都具有一种属性，叫优先级，就好比代表了病情的紧急性。当几个中断同时向CPU发出中断请求时，CPU会根据这些中断的优先级来安排处理的顺序，优先级高的先处理，优先级低的后处理。那F28335究竟支持哪些CPU中断呢？这些中断的优先级又是如何安排的呢？

F28335的CPU中断·CPU中断向量和优先级

F28335一共可以支持32个CPU中断，其中每一个中断都是一个32位的中断向量，也就是2个16位的寄存器，里面存储的是相应的中断服务子程序的入口地址，不过这个入口地址是个22位的地址。其中地址的低16位保存该向量的低16位，地址的高16位中的位0-位5则保存它的高6位，其余更高的10位被忽略，如图10-3所示。

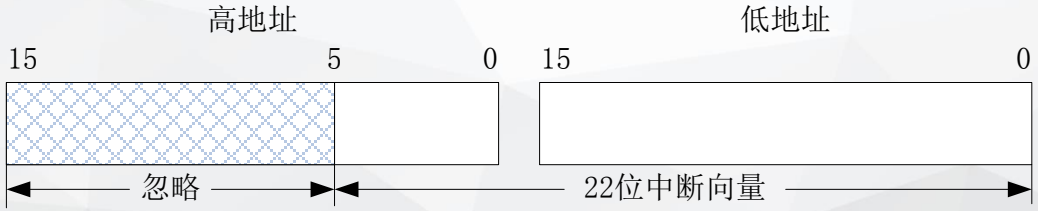


图10-3 22位的CPU中断向量

F28335的CPU中断·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，始终从ROM中0x003FFFC0处提取
INT1	0x0000 0D02	5	可屏蔽中断1，PIE组1
INT2	0x0000 0D04	6	可屏蔽中断2，PIE组2
INT3	0x0000 0D06	7	可屏蔽中断3，PIE组3
INT4	0x0000 0D08	8	可屏蔽中断4，PIE组4
INT5	0x0000 0D0A	9	可屏蔽中断5，PIE组5
INT6	0x0000 0D0C	10	可屏蔽中断6，PIE组6
INT7	0x0000 0D0E	10	可屏蔽中断7，PIE组7
INT8	0x0000 0D10	12	可屏蔽中断8，PIE组8
INT9	0x0000 0D12	13	可屏蔽中断9，PIE组9
INT10	0x0000 0D14	14	可屏蔽中断10，PIE组10
INT11	0x0000 0D16	15	可屏蔽中断11，PIE组11
INT12	0x0000 0D18	16	可屏蔽中断12，PIE组12
INT13	0x0000 0D1A	17	外部中断XINT13或者Timer1中断
INT14	0x0000 0D1C	18	Timer2中断
DLOGINT	0x0000 0D1E	19(最低)	CPU数据记录中断
RTOSINT	0x0000 0D20	4	CPU实时操作系统中断
EMUINT	0x0000 0D22	2	CPU仿真中断
NMI	0x0000 0D24	3	外部不可屏蔽中断
ILLEGAL	0x0000 0D26	——	非法操作
USER1	0x0000 0D28	——	用户自定义软中断
USER2	0x0000 0D2A	——	用户自定义软中断
USER3	0x0000 0D2C	——	用户自定义软中断
USER4	0x0000 0D2E	——	用户自定义软中断
USER5	0x0000 0D30	——	用户自定义软中断
USER6	0x0000 0D32	——	用户自定义软中断
USER7	0x0000 0D34	——	用户自定义软中断
USER8	0x0000 0D36	——	用户自定义软中断
USER9	0x0000 0D38	——	用户自定义软中断
USER10	0x0000 0D3A	——	用户自定义软中断
USER10	0x0000 0D3C	——	用户自定义软中断
USER12	0x0000 0D3E	——	用户自定义软中断

表10-1 CPU中断向量和优先级

F28335的CPU中断·CPU中断的寄存器

表10-1所列的CPU中断中，$\overline{\text{INT1}}$ ~ $\overline{\text{INT14}}$是14个通用中断，DLOGINT数据记录中断和RTOSINT实时操作系统中断是为仿真目的而设计的两个中断。通常在实际使用时，用到最多的还是通用中断$\overline{\text{INT1}}$~ $\overline{\text{INT14}}$。这16个中断都属于可屏蔽中断，根据可屏蔽中断的字面含义，不难理解这些是能够通过软件设置来使能或者禁止的中断，那在DSP中是怎么实现的呢？很简单，通过CPU中断使能寄存器IER就可以来实现。

F28335的CPU中断·CPU中断的寄存器

下图为IER寄存器的位情况。

15	14	13	12	10	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT10	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

CPU 中断使能寄存器 IER

注：R=可读，W=可写，–0=复位后的值。

F28335的CPU中断·CPU中断的寄存器

位	名称	说明
15	RTOSINT	实时操作系统中断使能位。该位使CPU RTOS中断使能或禁止。 0 RTOSINT中断禁止 1 IRTOSINT中断使能
14	DLOGINT	数据记录中断使能位。该位使CPU数据记录中断使能或禁止。 0 CPU数据记录中断禁止 1 CPU数据记录中断使能
13	INT14	中断14使能位。该位使CPU中断级INT14使能或禁止。 0 INT14禁止 1 INT14使能
12	INT13	中断13使能位。该位使CPU中断级INT13使能或禁止。 0 INT13禁止 1 INT13使能
10	INT12	中断12使能位。该位使CPU中断级INT12使能或禁止。 0 INT12禁止 1 INT12使能
10	INT10	中断10使能位。该位使CPU中断级INT10使能或禁止。 0 INT10禁止 1 INT10使能
9	INT10	中断10使能位。该位使CPU中断级INT10使能或禁止。 0 INT10禁止 1 INT10使能
8	INT9	中断9使能位。该位使CPU中断级INT9使能或禁止。 0 INT9禁止 1 INT9使能
7	INT8	中断8使能位。该位使CPU中断级INT8使能或禁止。 0 INT8禁止 1 INT8使能
6	INT7	中断7使能位。该位使CPU中断级INT7使能或禁止。 0 INT7禁止 1 INT7使能
5	INT6	中断6使能位。该位使CPU中断级INT6使能或禁止。 0 INT6禁止 1 INT6使能
4	INT5	中断5使能位。该位使CPU中断级INT5使能或禁止。 0 INT5禁止 1 INT5使能
3	INT4	中断4使能位。该位使CPU中断级INT4使能或禁止。 0 INT4禁止 1 INT4使能
2	INT3	中断3使能位。该位使CPU中断级INT3使能或禁止。 0 INT3禁止 1 INT3使能
1	INT2	中断2使能位。该位使CPU中断级INT2使能或禁止。 0 INT2禁止 1 INT2使能
0	INT1	中断1使能位。该位使CPU中断级INT1使能或禁止。 0 INT1禁止 1 INT1使能

CPU中断说明

F28335的CPU中断·CPU中断的寄存器

从图10-4可以看到，CPU中断使能寄存器中的每一个位都和一个CPU中断相对应，这个位的值就像是开关的状态，1为打开，0为关闭。当某一个位的值为1时，相对应的中断就被使能；当某一个位的值为0时，相对应的中断就被禁止，也就是如果这个时候有该中断的请求信号的话，这个请求信号CPU不会理，也就是被屏蔽。

除了可屏蔽中断的使能和禁止以外，还有一个问题，就是CPU是如何知道某个中断提出了中断请求信号的呢？举个小例子，在学校里上课的时候，学生如果要回答问题，得先举手，然后老师明白这个学生想要回答问题，再允许其发言。举手这个动作就是一个想要回答问题的标志，类似的，DSP中也有一个CPU中断的标志寄存器IFR，寄存器中的每一个位都和一个CPU中断相对应，这个位的状态就表示了该中断是否向CPU提出了请求。

F28335的CPU中断·CPU中断的寄存器

CPU 中断标志寄存器 IFR 的位情况如下图所示。

15	14	13	12	10	10	9	8
RTOSINT	DLOGINT	INT14	INT13	INT12	INT10	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

CPU 中断标志寄存器 IFR

注：R=可读，W=可写，–0=复位后的值。

F28335的CPU中断·CPU中断的寄存器

位	名称	说明
15	RTOSINT	实时操作系统标志。该位是RTOS中断的标志位。 0 没有未处理的RTOS中断。 1 至少有一个RTOS中断未处理。
14	DLOGINT	数据记录中断标志。该位是数据记录中断的标志。 0 没有未处理的DLOGINT中断。 1 至少有一个DLOGINT中断未处理。
13	INT14	中断14标志。该位是连接到CPU中断级INT14的中断标志。 0 没有未处理的INT14中断。 1 至少有一个INT14中断未处理。
12	INT13	中断13标志。该位是连接到CPU中断级INT13的中断标志。 0 没有未处理的INT13中断。 1 至少有一个INT13中断未处理。
10	INT12	中断12标志。该位是连接到CPU中断级INT12的中断标志。 0 没有未处理的INT12中断。 1 至少有一个INT12中断未处理。
10	INT10	中断10标志。该位是连接到CPU中断级INT10的中断标志。 0 没有未处理的INT10中断。 1 至少有一个INT10中断未处理。
9	INT10	中断10标志。该位是连接到CPU中断级INT10的中断标志。 0 没有未处理的INT10中断。 1 至少有一个INT10中断未处理。
8	INT9	中断9标志。该位是连接到CPU中断级INT9的中断标志。 0 没有未处理的INT9中断。 1 至少有一个INT9中断未处理。
7	INT8	中断8标志。该位是连接到CPU中断级INT8的中断标志。 0 没有未处理的INT8中断。 1 至少有一个INT8中断未处理。
6	INT7	中断7标志。该位是连接到CPU中断级INT7的中断标志。 0 没有未处理的INT7中断。 1 至少有一个INT7中断未处理。
5	INT6	中断6标志。该位是连接到CPU中断级INT6的中断标志。 0 没有未处理的INT6中断。 1 至少有一个INT6中断未处理。
4	INT5	中断5标志。该位是连接到CPU中断级INT5的中断标志。 0 没有未处理的INT5中断。 1 至少有一个INT5中断未处理。
3	INT4	中断4标志。该位是连接到CPU中断级INT4的中断标志。 0 没有未处理的INT4中断。 1 至少有一个INT4中断未处理。
2	INT3	中断3标志。该位是连接到CPU中断级INT3的中断标志。 0 没有未处理的INT3中断。 1 至少有一个INT3中断未处理。
1	INT2	中断2标志。该位是连接到CPU中断级INT2的中断标志。 0 没有未处理的INT2中断。 1 至少有一个INT2中断未处理。
0	INT1	中断1标志。该位是连接到CPU中断级INT1的中断标志。 0 没有未处理的INT1中断。 1 至少有一个INT1中断未处理。

CPU中断说明

F28335的CPU中断·可屏蔽中断的响应过程

可屏蔽中断的响应过程如图10-6所示。当某个可屏蔽中断提出请求时，将其在中断标志寄存器IFR中的中断标志位自动置位。CPU检测到该中断标志位被置位后，接着会检查该中断是否被使能了，也就是去读CPU中断使能寄存器IER中相应位的值，如果该中断并未使能，那么CPU将不会理会此中断，直到其中断被使能为止。如果该中断已经被使能，则CPU会继续检查全局中断INTM是否被使能，如果没有使能，则依然不会响应中断，如果INTM已经被使能，则CPU就会响应该中断，暂停主程序并转向执行相应的中断服务子程序。CPU响应中断后，IFR中的中断标志位就会被自动清零，目的是使CPU能够去响应其他中断或者是该中断的下一次中断。

F28335的CPU中断·可屏蔽中断的响应过程

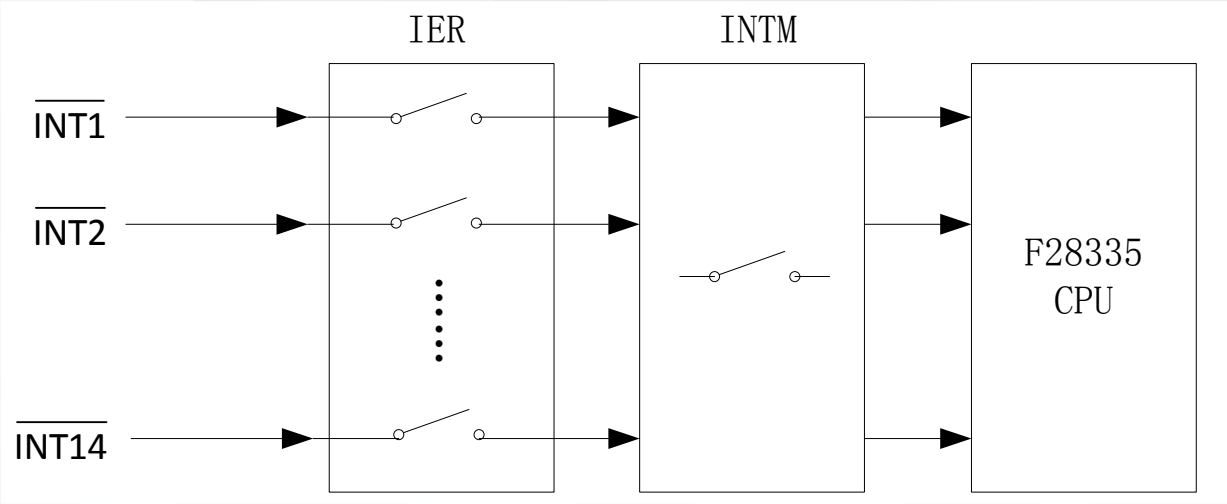


图10-6 可屏蔽中断的响应过程

F28335的CPU中断·可屏蔽中断的响应过程

图10-6中IER和INTM的关系比较简单。就好比是在一个房间里，有好多灯，也有好多开关，一个开关控制着一盏灯，开关闭合时，对应的灯就亮，开关断开时，对应的灯就灭。通常，房间里除了这些开关以外，还会有一个总闸，如果总闸关了，就切断了房间的线路和外面电网的连接，不管房间里的开关是开还是关，灯都不会亮。在CPU中断响应的过程里，IER中的各个位就是控制一个个灯的开关，而INTM就是总闸。如果一个中断被使能了，而全局中断没有被使能，则CPU还是不会去响应中断的。只有当单个中断和全局中断都被使能了，这时候该中断提出请求时，CPU才会去响应。

F28335的CPU中断·可屏蔽中断的响应过程

这里再来讨论下当多个中断同时提出中断请求时，CPU响应的过程。假如有中断A和中断B，中断A的优先级高于中断B的优先级，中断A和中断B都被使能了，而且全局中断INTM也已经使能了。这时当中断A和中断B同时提出中断请求时，CPU就会根据优先级的高低，先来响应中断A，同时清除A的中断标志位。当CPU处理完中断A的服务子程序后，如果这时候中断B的中断标志位还处于置位的状态，那么CPU就会响应中断B，转而去执行中断B的服务子程序。如果CPU在执行中断A的服务子程序时，中断A的标志位又被置位了，也就是中断A又向CPU提出了请求，那当CPU完成中断响应之后，还是会继续先响应中断A，而让中断B继续在队列中等待。

F28335的PIE中断

前面介绍的是F28335 CPU级的中断，图10-7是F28335 DSP的中断源。F28335的CPU一共有16根中断线，其中包括两个不可屏蔽中断，$\overline{\text{RS}}$ 和$\overline{\text{NMI}}$，还有14个可屏蔽中断$\overline{\text{INT1}}$~ $\overline{\text{INT14}}$。

外部中断XINT13和CPU1定时器1的中断分配给了$\overline{\text{INT13}}$，CPU定时器2的中断分配给了$\overline{\text{INT14}}$。两个不可屏蔽中断$\overline{\text{RS}}$和$\overline{\text{NMI}}$也各自都有专用的独立中断。CPU定时器0的周期中断、F28335片内外设的所有中断、外部中断$\overline{\text{XINT1}}$、外部中断$\overline{\text{XINT2}}$共用中断线$\overline{\text{INT1}}$~ $\overline{\text{INT12}}$。通常使用的最多的，也是$\overline{\text{INT1}}$~ $\overline{\text{INT12}}$，因此这些中断是需要重点介绍和探讨的。

F28335的PIE中断

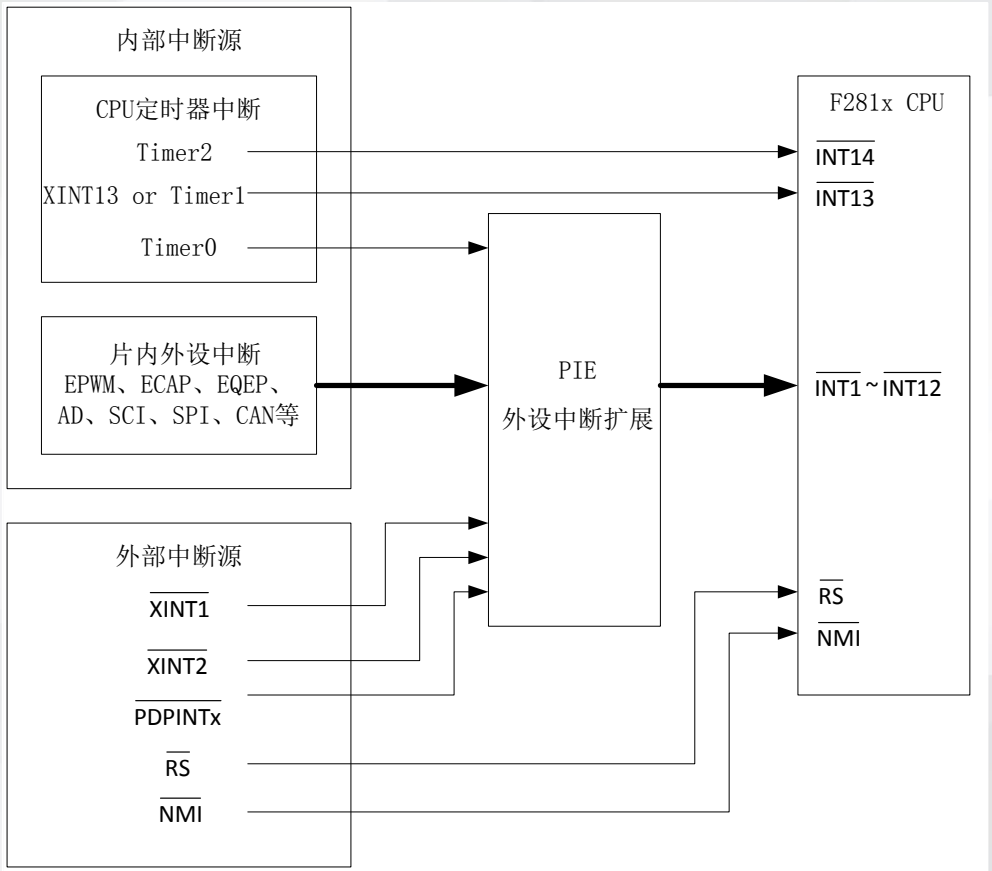


图10-7 F28335 DSP的中断源

F28335的PIE中断·PIE中断概述

通过前面的学习，已经知道F28335内部具有很多外设(EPWM、ECAP、EQEP、AD、SCI、SPI、McBSP和CAN等)，每个外设又可以产生一个或者多个中断请求，对于CPU而言，它没有足够的能力去同时处理所有外设的中断请求。打个比方，这就好比一家大公司，每天会有很多员工向老总提交文件，请求老总处理。老总通常事务繁忙，他一个人没有能力同时去处理所有的事情，那怎么办呢？一般老总会配有秘书，由秘书们将内部员工或者外部人员提交的各种事情进行分类筛选，按照事情的轻重缓急进行安排，然后再提交给老总处理，这样效率就提高上来了，老总也能忙的过来了。同样的，F28335的CPU为了能够及时有效地处理好各个外设的中断请求，特别设计了一个“秘书”——专门处理外设中断的扩展模块(the Peripheral Interrupt Expansion Block)，简称外设中断控制器PIE，它能够对各种中断请求源(来自于外设或者其他外部引脚的请求)做出判断和相应的决策。

F28335的PIE中断·PIE中断概述

PIE一共可以支持96个不同的中断，并把这些中断分成了12个组，每个组有8个中断，而且每个组都被反馈到CPU内核的$\overline{\text{INT1}}$~ $\overline{\text{INT12}}$这12条中断线中的某一条上。平时能够用到的所有的外设中断都被归入了这96个中断中，被分布在不同的组里。外设中断在PIE中的分布情况如表10-2所示。

表10-2是F28335内部的外设中断分布，共8列12行，总共有96个中断，空白部分表示尚未使用的中断，目前已经使用的有58个中断。下面来看看CPU定时器0的周期中断TINT0在表中的哪个位置？很明显，TINT0在行号为INT1，列号为INTx.7的位置，也就是说TINT0对应于INT1，在PIE第一组的第7位。同样的，可以找到所有外设中断在PIE中的所属分组情况以及在该组中的位置。

F28335的PIE中断·PIE中断概述

	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
INT1	WAKEINT	TINT0	ADCINT	XINT2	XINT1		SEQ2INT	SEQ1INT
INT2			EPWM6_TZINT	EPWM5_TZINT	EPWM4_TZINT	EPWM3_TZINT	EPWM2_TZINT	EPWM1_TZINT
INT3			EPMW6_INT	EPWM5_INT	EPWM4_INT	EPWM3_INT	EPWM2_INT	EPWM1_INT
INT4			ECAP6_INT	ECAP5_INT	ECAP4_INT	ECAP3_INT	ECAP2_INT	ECAP1_INT
INT5							EQEP2_INT	EQEP1_INT
INT6			MXINTA	MRINTA	MXINTB	MRINTB	SPITXINTA	SPIRXINTA
INT7			DINTCH6	DINTCH5	DINTCH4	DINTCH3	DINTCH2	DINTCH1
INT8			SCITXINTC	SCIRXINTC			I2CINT2A	I2CINT1A
INT9	ECAN1INTB	ECAN0INTB	ECAN1INTA	ECAN0INTA	SCITXINTB	SCIRXINTB	SCITXINTA	SCIRXINTA
INT10								
INT11								
INT12	LUF(FPU)	LVF(FPU)		XINT7	XINT6	XINT5	XINT4	XINT3

表10-2 外设中断在PIE的分布

F28335的PIE中断·PIE中断概述

PIE第1组的所有外设中断复用CPU中断INT1，PIE第2组的所有外设中断复用CPU中断INT2，以此类推，PIE第12组的所有外设中断复用CPU中断INT12。在前面讲CPU中断的时候，知道INT1的优先级比INT2的优先级高，INT2的优先级比INT3的优先级高，……。那对于PIE同组内的各个中断，是不是也是有优先级高低的呢？答案是肯定的。在PIE同组内，INTx.1的优先级比INTx.2的优先级高，INTx.2的优先级比INTx.3的优先级高，……。也就是说，同组内排在前面的优先级比排在后面的优先级高。而不同组之间，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。例如位于INT1.8的WAKEINT，虽然属于第一组的第8位，但是它的优先级就要比位于INT2.1的EPWM1_TZINT的优先级高。这样表10-2内所有中断的优先级关系应该都清楚了吧。

F28335的PIE中断·PIE中断概述

可屏蔽CPU中断都可以通过中断使能寄存器IER和中断标志寄存器IFR来进行可编程控制，同样的，PIE的每个组都有三个相关的寄存器，分别是PIE中断使能寄存器PIEIERx，PIE中断标志寄存器PIEIFRx和PIE中断应答寄存器PIEACKx。比如，PIE的第一组具有寄存器PIEIER1、PIEIFR1和PIEACK1。寄存器的每个位同中断的对应关系和表10-2中是相同的，例如CPU定时器中断TINT0对应于PIEIER1.7，PIEIFR1.7和PIEACKINT1.7，就是分别在PIEIER1、PIEIFR1和PIEACK1的第7位。下面对各个寄存器进行详细的介绍和说明。

F28335的PIE中断· PIE中断寄存器

PIE控制器相关的寄存器如表10-3所示。

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
PIEIER10	0x0000 0CF6	1	PIE，INT10组使能寄存器
PIEIFR10	0x0000 0CF7	1	PIE，INT10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

表10-3 PIE控制器的寄存器

F28335的PIE中断· PIE中断寄存器

1.PIE中断使能寄存器

PIE控制器一共有12个PIE中断使能寄存器PIEIERx，分别对应于PIE控制器的12个组，每组1个，用来设置组内中断的使能情况。PIE中断使能寄存器PIEIERx的位分布如图所示。

15	14	13	12	11	10	9	8
保留							
R－0							

7	6	5	4	3	2	1	0
INTx. 8	INTx. 7	INTx. 6	INTx. 5	INTx. 4	INTx. 3	INTx. 2	INTx. 1
R/W－0	R/W－0	R/W－0	R/W－0	R/W－0	R/W－0	R/W－0	R/W－0

PIE 中断使能寄存器 PIEIERx

F28335的PIE中断· PIE中断寄存器

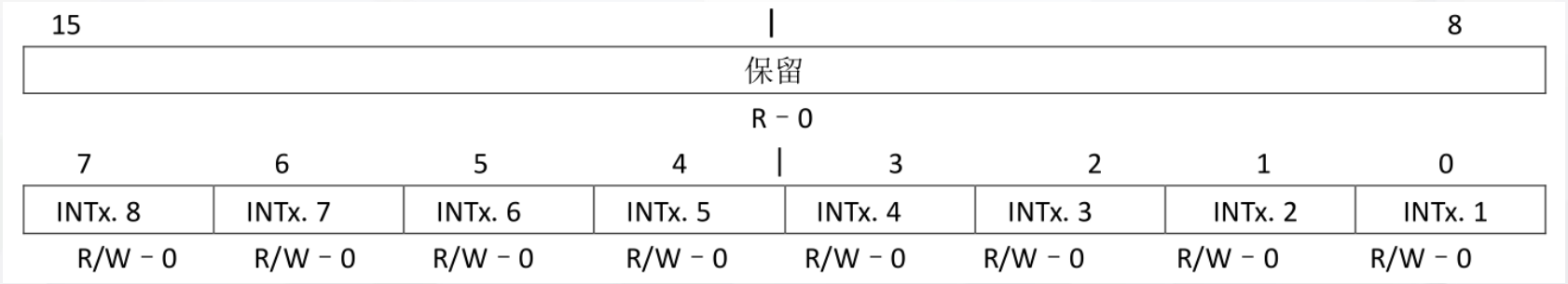
1.PIE中断使能寄存器

PIE控制器一共有12个PIE中断使能寄存器PIEIERx，分别对应于PIE控制器的12个组，每组1个，用来设置组内中断的使能情况。PIE中断使能寄存器PIEIERx的位分布如图所示。

位	名称	说明
15 ~ 8	Reserved	保留。
7	INTx.8	对PIE组内各个中断单独使能，和CPU中断使能寄存器IER类似。把某位置1，可以使能中断服务；将某位置0将使该中断服务禁止。 x=1～12，INTx表示CPU的INT1～INT12。
6	INTx.7	
5	INTx.6	
4	INTx.5	
3	INTx.4	
2	INTx.3	
1	INTx.2	
0	INTx.1	

2.PIE中断标志寄存器

PIE控制器一共有12个PIE中断标志寄存器PIEIFRx，分别对应于PIE控制器的12个组，每组1个。PIEIFR寄存器的每一个位代表对应中断的请求信号，当该位置1，表示相应的中断提出了请求，需要CPU响应。CPU取出相应的中断向量的时候，也就是说当CPU响应该中断的时候，该标志位被清0。PIE中断标志寄存器PIEIFRx的位分布如图10-9所示。



PIE 中断标记寄存器 PIEIERx

注：R=可读，W=可写，–0=复位后的值。

2.PIE中断标志寄存器

PIE控制器一共有12个PIE中断标志寄存器PIEIFRx，分别对应于PIE控制器的12个组，每组1个。PIEIFR寄存器的每一个位代表对应中断的请求信号，当该位置1，表示相应的中断提出了请求，需要CPU响应。CPU取出相应的中断向量的时候，也就是说当CPU响应该中断的时候，该标志位被清0。PIE中断标志寄存器PIEIFRx的位分布如图10-9所示。

位	名称	说明
15～8	Reserved	保留。
7	INTx.8	这些位表示一个中断当前是否被激活，向CPU提出了中断请求。它们和CPU中断标志寄存器IFR类似。当中断激活时，各个寄存器位置1。当一个中断被处理完成或向该寄存器位写0时，该位清0。该寄存器还可以被读取以确定哪个中断被激活或未处理。 x=1～12，INTx表示CPU的INT1～INT12。
6	INTx.7	
5	INTx.6	
4	INTx.5	
3	INTx.4	
2	INTx.3	
1	INTx.2	
0	INTx.1	

3.PIE中断应答寄存器

如果PIE中断控制器有中断产生，则相应的中断标志位将置1。如果相应的PIE中断使能位也置1，则PIE将检查PIE中断应答寄存器PIEACK，以确定CPU是否准备响应该中断。如果相应的PIEACKx清0，PIE便向CPU申请中断；如果相应的PIEACKx置1，那么PIE将等待直到相应的PIEACKx清0才向CPU申请中断。PIE中断应答寄存器PIEACK的位情况如图所示。

图10-10 PIE中断应答寄存器PIEACK

注：R=可读，W=可写，–1=复位后的值。

F28335的PIE中断· PIE中断寄存器

3.PIE中断应答寄存器

如果PIE中断控制器有中断产生，则相应的中断标志位将置1。如果相应的PIE中断使能位也置1，则PIE将检查PIE中断应答寄存器PIEACK，以确定CPU是否准备响应该中断。如果相应的PIEACKx清0，PIE便向CPU申请中断；如果相应的PIEACKx置1，那么PIE将等待直到相应的PIEACKx清0才向CPU申请中断。PIE中断应答寄存器PIEACK的位情况如图所示。

位	名称	说明
15~12	Reserved	保留。
10~0	PIEACK	该寄存器的第0位表示PIE第一组中断的CPU响应情况，第1位表示PIE第二组中断的CPU响应情况，……，第10位表示PIE第12组中断的CPU响应情况。向该寄存器的某一位写1，可使该位清0，此时如果该组内有CPU尚未响应的中断，则PIE向CPU提出中断请求。

4.PIE控制寄存器

PIE控制寄存器PIECTRL的位情况如图所示。

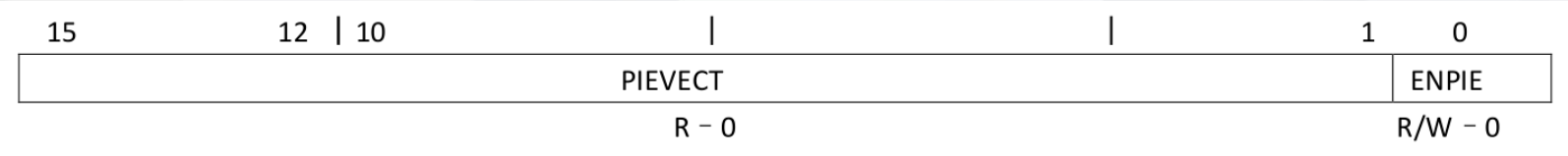


图10-11 PIE控制寄存器PIECTRL

注：R=可读，W=可写，–1=复位后的值。

位	名称	说明
15～1	PIEVECT	这些位表示从PIE向量表取回的向量地址。最低位忽略只显示位1到位15地址。用户可以读取向量值，以确定取回的向量是由哪一个中断产生的。
0	ENPIE	从PIE块取回向量使能。当该位置1时，所有向量取自PIE向量表。如果该位置0，PIE块无效，向量取自引导ROM的CPU向量表或XINTF 7区外部接口。

F28335的PIE中断·外部中断控制寄存器

F28335支持XINT1~XINT7和XNMI，一共8路外部引脚中断，其中XNMI为非可屏蔽中断。通过控制寄存器XINTnCR可使能或禁止8路中的任何一路，同时可为每一路选择配置为上升沿触发或下降沿触发。寄存器XINTnCR的具体信息请查看C2000助手。

F28335的PIE中断·PIE中断向量表

PIE一共可以支持96个中断，每个中断都会有中断服务子程序ISR，那CPU去响应中断时是如何找到对应的中断服务子程序的呢？解决方法是将DSP的各个中断服务子程序的地址存储在一片连续的RAM空间内，这就是PIE中断向量表。F28335的PIE中断向量表是由256*16的RAM空间组成，如果不使用PIE模块，则这个空间也可以作为通用的RAM使用。

*F28335的PIE中断向量表略。

F28335的三级中断系统分析

如图10-12所示，F28335的中断采用的是三级中断机制，分别为外设级、PIE级和CPU级。对于某一个具体的外设中断请求，任意一级的不许可，CPU最终都不会响应该外设中断。这就好比一个文件需要三级领导的批示一样，任意一级领导的不同意，都不能被送至上一级领导，更不可能得到最终的批复，中断机制的原理也是如此。上一章里介绍了CPU定时器0，也提及了当CPU定时器0完成一个周期的计数后就会产生一个中断信号，也就是CPU定时器0 的周期中断。接下来，以CPU定时器0的周期中断为例来探讨F28335的三级中断系统。

F28335的三级中断系统分析

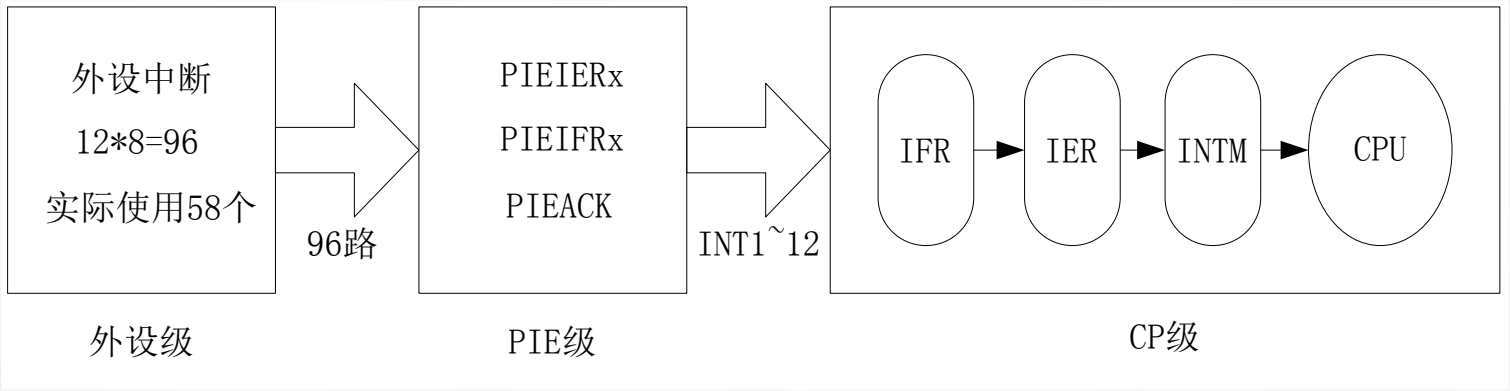


图10-12 F28335的三级中断机制

1.外设级

假如在程序执行的过程中，某一个外设产生了一个中断事件，那么在这个外设的某个寄存器中与该中断事件相关的中断标志位(IF=Interrupt Flag)被置为1。此时，如果该中断相应的中断使能位(IE=Interrupt Enable)已经被置位，也就是值为1，该外设就会向PIE控制器发出一个中断请求。相反，如果虽然中断事件已经发生了，相应的中断标志位也被置位了，但是该中断没有被使能，也就是中断使能位的值为0，那么外设就不会向PIE控制器提出中断请求。值得一提的是，这时候虽然外设不会向PIE控制器提出中断请求，但是相应的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，在中断标志位保持置位状态的时候，一旦该中断被使能了，那么外设会立即向PIE发出中断请求。

1.外设级

下面结合具体的T0INT来进行进一步的说明。当CPU定时器0的计数器寄存器TIMH:TIM计数到0时，就产生了一个T0INT事件，即CPU定时器0的周期中断。这时候，CPU定时器0的控制寄存器TIMER0TCR的第15位定时器中断标志TIF被置位1。这时候，如果TIMER0TCR的第14位，也就是定时器中断使能位TIE是1的话，则CPU定时器0就会向PIE控制器发出中断请求，当然如果TIE的值是0，也就是该中断未被使能，则CPU定时器0不会向PIE发出中断请求，而中断标志位TIF将一直保持为1，除非通过程序将其清除。需要注意的是，不管在什么情况下，外设寄存器中的中断标志位都必须手工清除。（CI、SPI除外，讲解到具体内容时会做介绍。

1.外设级

例如，清除CPU定时器0中断标志位TIF的语句如下：

```
CpuTimer0Regs.TCR.bit.TIF=1;  //清除定时器中断标志位
```

看了上面的语句，是否会有疑问，不是说清除中断标志位么，这个语句却明明是对TIF位写1呀？其实，在F28335的编程中，很多时候都是通过对寄存器的位写1来清除该位的。写0是无效的，只有写1才能将该标志位复位，在应用的时候请查阅各个寄存器位的具体说明。

F28335的三级中断系统分析

1.外设级

介绍了这么多之后，接下来总结一下在外设级需要在编程时手动的地方有：

- 外设中断的使能，需要将与该中断相关的外设寄存器中的中断使能位置1；
- 外设中断的屏蔽，需要将与该中断相关的外设寄存器中的中断使能位置0；
- 外设中断标志位的清除，需要将与该中断相关的外设寄存器中的中断标志位置1。

2.PIE级

当外设产生中断事件，相关中断标志位置位，中断使能位使能之后，外设就会把中断请求提交给PIE控制器。前面已经讲过，PIE控制器将96个外设和外部引脚的中断进行了分组，每8个中断为1组，一共是12组，分别是PIE1~PIE12。每个组的中断被多路汇集进入了1个CPU中断，例如SEQ1INT、SEQ2INT、XINT1、XINT2、ADCINT、TINT0、WAKEINT这7个中断都在PIE1组内，这些中断也都汇集到了CPU中断的INT1，同样的，PIE2组的中断都被汇集到了CPU中断的INT2，……，PIE12组的中断都被汇集到了CPU中断的INT12。

2.PIE级

和外设级相类似的，PIE控制器中的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=1，2，……，12。每个寄存器的低8位对应于8个外设中断，高8位保留。这些寄存器在前面的PIE中断寄存器部分已经介绍到，例如CPU定时器0的周期中断T0INT对应于PIEIFR1的第7位和PIEIER1的第7位。

由于PIE控制器是多路复用的，每一组内有许多不同的外设中断共同使用一个CPU中断，但是每一个组在同一个时间内只能有一个中断被响应，那么PIE控制器是如何实现的呢。

2.PIE级

首先，PIE组内的各个中断也是有优先级的，位置在前面的中断的优先级比位置在后面的中断的优先级来的高，这样，如果同时有多个中断提出请求的话，PIE先处理优先级高的，后处理优先级低的。同时，PIE控制器除了每组有PIEIFR和PIEIER寄存器之外，还有一个PIE中断应答寄存器PIEACK，如前面的图10-10所示，它的低12位分别对应着12个组，即PIE1~PIE12，也就是INT1~INT12，高位保留。这些位的状态就表示了PIE是否准备好了去响应这些组内的中断。比如CPU定时器0的周期中断被响应了，则PIEACK的第0位(对应于PIE1，即INT1)就会被置位，并且一直保持直到手动清除这个标志位。

2.PIE级

当CPU在响应T0INT的时候，PIEACK的第0位一直是1，这时候如果PIE1组内发生了其他的外设中断，则暂时不会被PIE控制器响应并发送给CPU，必须等到PIEACK的第0位被复位之后，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的其他中断。清除PIEACK中与T0INT相关的应答位的语句如下所示：

```
PieCtrl.PIEACK.bit.ACK1=1;    //响应PIE组1内的其他中断。
```

2.PIE级

因此，当外设中断向PIE提出中断请求之后，PIE中断标志寄存器PIEIFRx的相关标志位被置位，这时候如果相应的PIEIERx相关的中断使能位被置位，PIEACK相应位的值为0，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相关的中断使能位没有被置位，就是没有被使能，或者PIEACK相应位的值为1，就是PIE控制器正在处理同组的其他中断，PIE控制器都暂时不会响应外设的中断请求。

2.PIE级

通过上面的分析，在PIE级需要编程时手动处理的地方有：

- PIE中断的使能。需要使能某个外设中断，就得将其相应组的使能寄存器PIEIERx的相应位进行置位；
- PIE中断的屏蔽。这是和使能相反的操作；
- PIE应答寄存器PIEACK相关位的清除，以使得CPU能够响应同组内的其他中断。

2.PIE级

将PIE级的中断和外设级的中断相比较之后发现，外设中断的中断标志位是需要手工清除的，而PIE级的中断标志位都是自动置位或者是清除的。但是PIE级多了一个PIEACK寄存器，它相当于一个关卡，同一时间只能放一个中断过去，只有等到这个中断被响应完成之后，再给关卡一个放行命令之后，才能让同组的下一个中断过去，被CPU响应。

3.CPU级

和前面两级类似，CPU级也有中断标志寄存器IFR和中断使能寄存器IER。当某一个外设中断请求通过PIE发送到CPU时，CPU中断标志寄存器IFR中相对应的中断标志位INTx就会被置位。例如，当CPU定时器0的周期中断T0INT发送到CPU时，IFR的第0位INT1就会被置位，然后该状态就会被锁存在寄存器IFR中。这时候，CPU不会马上去执行相应的中断，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中的相关位被置位了，并且INTM的值为0，则中断就会被CPU响应。在CPU定时器0的周期中断的例子里，当IER的第0位INT1被置位，INTM的值为0，则CPU就会响应定时器0的周期中断T0INT。

3.CPU级

CPU接到了中断请求，并发现可以去响应的时候，就得暂停正在执行的程序，转而去响应中断程序，但是此时，它必须得做一些准备工作，以便于执行完中断程序之后回过头来还能找到原来的地方和原来的状态。CPU会将相应的IFR位进行清除，EALLOW也被清除，INTM被置位，就是不能响应其他中断了，等于CPU向其他中断发出了通知，现在正在忙，没有时间处理别的请求了，得等到处理完手上的中断之后才能再来处理。然后，CPU会存储返回地址并自动保存相关的信息，例如将正在处理的数据放入堆栈等等，做好这些准备工作之后，CPU会从PIE向量表中取出对应的中断向量ISR，从而转去执行中断服务子程序。

3.CPU级

可以看到，CPU级中断标志位的置位和清除也都是自动完成的。图10-13很形象的表示了F28335的三级中断，能够帮助更好的理解这部分内容，可以结合此图反复的对照琢磨。

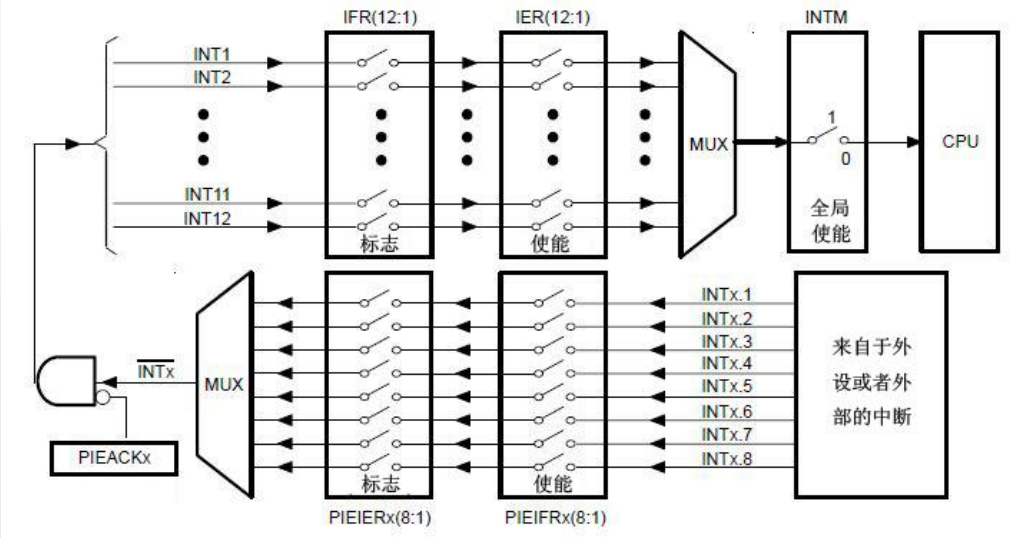


图10-13 F28335中断的工作过程

成功实现中断的必要步骤

对于刚刚使用DSP的用户可能会常常遇到中断无法进入的问题，这确实是一件非常郁闷的事情。接下来，将详细介绍怎样编写中断程序才能够顺利进入中断，这部分内容可能需要大家在实际使用的时候才能够有所体会，因为毕竟“绝知此事需躬行”。

成功实现中断的必要步骤

先来看看光盘的编程素材文件夹内一些在建立工程时需要使用的文件，也是推荐的工程结构所需的文件。首先来看看DSP2833x_Piectrl.h，这一个文件定义了PIE寄存器的数据结构，如果对照书中所介绍的相关寄存器的定义，可以发现两者是一样的。然后看看DSP2833x_PieVect.h，这个头文件定义了PIE的中断向量。接下来看源文件。DSP2833x_PieCtrl.c文件里只有1个函数，InitPieCtrl（ ），其作用是对PIE控制器进行初始化的，例如在程序开始的时候使能某些外设中断。DSP2833x_PieVect.c文件是对PIE中断向量表进行初始化的。执行完这个程序后，各个中断函数就有了明确的入口地址了，这样CPU执行起来也方便了。最后，需要来关注下DSP2833x_DefaultIsr.c这个文件，大家或许会惊讶的发现， F28335所有的与外设相关的中断函数都已经在这个文件里预定义好了，在编写中断函数的时候，只需将具体的函数内容写进去就可以了。图10-14是ADC中断函数。

成功实现中断的必要步骤

```
interrupt void  ADCINT_ISR(void)     // ADC中断函数
{
  // 在这里插入中断函数的代码

  // 注意退出中断函数时需要先释放PIE，使得PIE能够响应同组其他中断
  // PieCtrl.PIEACK.all = PIEACK_GROUP1;

  // 下面两行只是为了编译而写的，插入代码后请将其删除

  // 中断函数代码
     asm ("      ESTOP0");
     for(;;);

  // 返回;
}
```

图10-14 DSP2833x_Default.c文件中的ADC中断函数

成功实现中断的必要步骤

除了采用上述的文件结构外，接下来，介绍下具体的写法，以保证中断能够成功进入。仍然以CPU定时器0的周期中断T0INT为例。其实编写一个成功的中断并不难，书写时请按照下面的步骤来：

1.在外设初始化函数中使能外设中断。

外设初始化函数

```
void InitCpuTimers(void)
{
            ……
            CpuTimer0Regs.TCR.bit.TIE=1; //使能CPU定时器0的周期中断
   ……
}
```

成功实现中断的必要步骤

主函数中的处理

```
void main(void)
{
  .........
  .........

//初始化CPU定时器0
 InitCpuTimers();

//禁止和清除所有CPU中断
   DINT;
   IER=0x0000;
   IFR=0x0000;

//初始化中断向量
   InitPieCtrl();

//初始化中断向量表
   InitPieVectTable();

//使能PIE中断
 PieCtrl.PIEIER1.bit.INTx7 =1; //使能PIE模块中的CPU定时器0的中断

//开CPU中断
   IER |=M_INT1;                        //开中断1

EINT;                                 //使能全局中断
 ERTM;                                  //使能实时中断

}
```

2.在主函数里需要注意一些步骤，不可缺少，主要是初始化外设，使能PIE和CPU中断等。

这里，来分析一下开CPU中断的语句：“IER |=M_INT1 ”。为什么这个语句表示开CPU中断1呢？首先，M_INT1的值为0x0001，这是在DSP2833x_Device.h文件内定义的。这样，“IER |=M_INT1 ”就等于是“IER|=0x0001”，也就等于“IER=IER|0x0001”。也就是IER的最低位与1进行或运算，然后把结果赋给IER的最低位，显然这样一运算之后，IER的最低位变成了1。IER的最低位代表的就是CPU中断1的使能位，现在这个位的值为1，也就是说使能了CPU中断1。

成功实现中断的必要步骤

3.在文件DSP2833x_DefaultIsr.c的中断函数中需要注意的一些步骤，必须要手动清除外设中断的标志位和复位PIE应答寄存器PIEACK相关的位，使得CPU能够响应PIE控制器同组内的其他中断。

中断函数的处理

```
interrupt void  TINT0_ISR(void)      // CPU-Timer0中断函数
{
   ……
   ……
   CpuTimer0Regs.TCR.bit.TIF=1;  //清除定时器中断标志位
   PieCtrl.PIEACK.bit.ACK1=1;    //响应同组其他中断
   EINT;  //开全局中断
}
```

成功实现中断的必要步骤

如果按照上述的方法来编写中断程序，一般是不会出错的。当然，万一出现了中断无法进入的时候，也不用着急，一定要学会分析，通过分析找到问题，然后加以解决。首先，应该检查上述的一些程序处理，是不是有疏忽弄错的地方，其次要分析是不是有中断源，就是中断事件是不是确实发生了，如果中断事件都没有发生，那么也就不可能进入中断程序。

前面介绍的是使用TI已经提供的文件架构来实现的中断函数，用户当然也可以自定义中断函数，如何自定义中断函数详见下面的例程。

使用CPU定时器0的周期中断控制LED灯的闪烁

上一章中由于还没有介绍中断的知识，所以也没有讲CPU定时器的应用实例，现在来看看如何使用CPU定时器0的周期中断来控制LED灯的闪烁。硬件电路图比较简单，如图10-14所示。

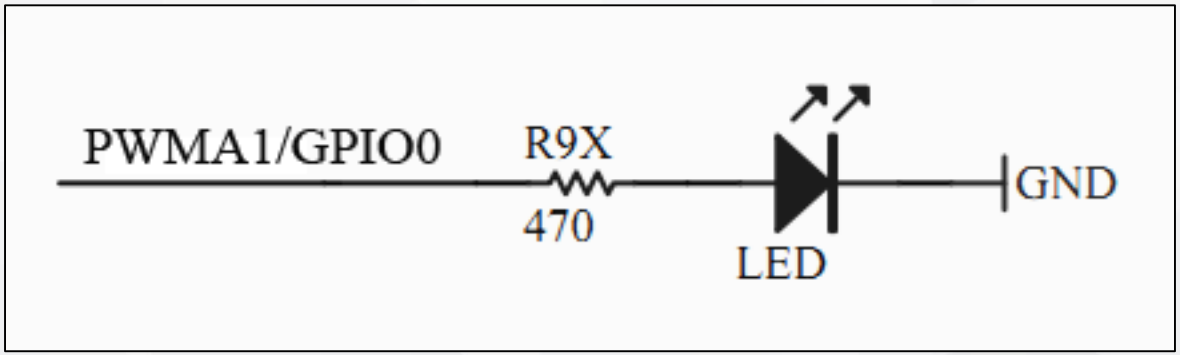


图10-14 GPIO引脚驱动LED灯

使用CPU定时器0的周期中断控制LED灯的闪烁

从图10-14不难看出，当引脚GPIO0为高电平时，LED灯点亮；当引脚GPIO0为低电平时，LED灯熄灭。

CPU定时器0在完成一个周期的计数之后，会产生一个周期中断。设置CPU定时器0的周期为1s，这样每隔1s时间就会就会进入一次周期中断，然后在中断函数中改变GPIO引脚的电平，这样就能实现每隔1s钟LED灯闪烁一次的功能。此实验的例程在配套资源的project example文件夹内。

F28335的中断系统

本章首先从上至下详细介绍了F28335的CPU中断、PIE中断、中断向量表等内容，然后又从下至上详细分析了F28335 DSP的三级中断系统，了解了在DSP中是如何从外设产生中断事件到PIE控制器再到CPU响应中断事件的整个过程，并详细介绍了成功进入DSP中断必须的一些步骤。最后，介绍了如何使用CPU定时器0的周期中断来控制LED灯的周期性闪烁。下一章，将详细介绍F28335的模数转换器ADC。