**深 圳 大 学 实 验 报 告**

|  |  |  |
| --- | --- | --- |
| **课程名称:** | **DSP原理与应用技术** |  |
| **实验名称:** | **作弊秒表实验** |  |
| **学 院:** | **机电与控制工程学院** |  |
| **专 业:** | **自动化** |  |
| **指导教师:** | **王怀智** |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **姓名:** |  | **学号:** |  | **班级:** |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **分组编号:** |  | **实验台号:** |  |  |

|  |  |  |
| --- | --- | --- |
| **实验时间:** | **2021年 月 日** |  |
| **提交时间:** | **2021年 月 日** |  |

**教务部制**

**实验目的与要求**

实验目的：

熟练使用已经学习的DSP功能模块（GPIO, CPU TIMER, SPI/LED）；

掌握各功能模块的初始化配置；

提高面向DSP的C语言编程能力。

了解什么是PIE向量表？什么是中断操作顺序中的外设级，PIE级和CPU级？

实验要求：

基本秒表功能：数码管显示时间按照1s周期递增，当按下SW6时，秒表停止计时，同时按下SW3和SW6时，秒表清零；

欺诈秒表功能：在基本秒表的功能下，每次按下SW3，秒表周期增加0.2s（最大周期3s）；每次按下SW9，秒表周期减少0.2s（最小周期0.2s）。

利用CPU TIMER和GPIO实现四位欺诈秒表的功能。

**实验原理**

**中断的基本概念**

1. 中断申请

中断请求信号是由软件或硬件驱动的信号，可暂停目前执行的主程序，转而去执行一个中断服务子程序。

通常中断请求由外设和硬件产生，以便CPU实现数据的传送或接收。(ADC,DAC等设备)

中断也可以作为特殊事件发生的标志信号

2.中断分类

可屏蔽中断：可用软件加以屏蔽或使能。是由外设产生的中断

非屏蔽中断：这些中断不能够被屏蔽。所有软件触发的中断都属于该类中断

CPU将立即响应该种中断并转入相应的子程序去执行。

3.中断向量及中断向量表

中断向量:中断服务程序的入口地址

中断向量号：中断向量按一定顺序0，1，2，… n,编排的序号称为中断向量号

中断向量表：按中断向量号的顺序把中断向量放进一块连续的存储器空间，这块存储器空间称为中断向量表

中断向量地址：中断向量所存放的存储器单元的地址称为中断向量地址

4. 中断的处理过程

1. 接收中断请求：

由软件中断(从程序代码中)或者硬件中断(从一个引脚或一个基于芯片的设备)提出请求去暂停当前主程序的执行。

1. 响应中断：

如果中断是可屏蔽的， CPU则必须满足一定的条件。而对于非屏蔽硬件中断和软件中断，CPU会立即作出响应。

1. 准备执行中断服务程序并保存寄存器值。
2. 完整地执行完当前指令，清除流水线中还没有到达第二阶段的所有指令。
3. 将寄存器ST0、T、AH、AL、PH、PL、AR0、ARl、DP、STl、DBGSTAT、PC和IER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（现场保护）。

③ 取回中断向量并把它放入程序寄存器PC中。

1. 执行中断服务子程序：

CPU进入预先规定的中断向量地址，并且执行已写好的中断服务程序。

**中断向量和优先级**

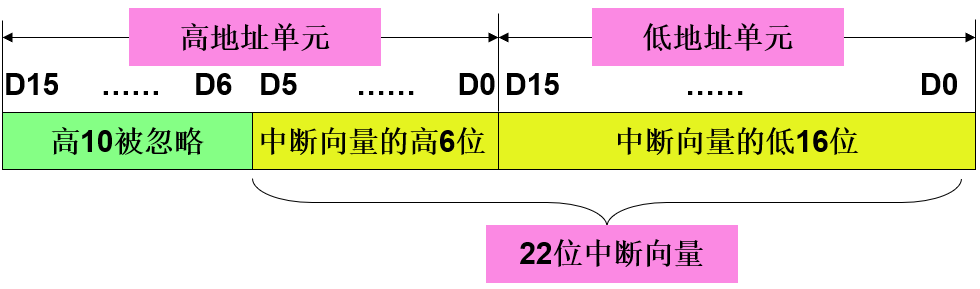
C28x系列芯片支持32个中断向量。每个中断向量占2个存储器单元，共64个存储单元。

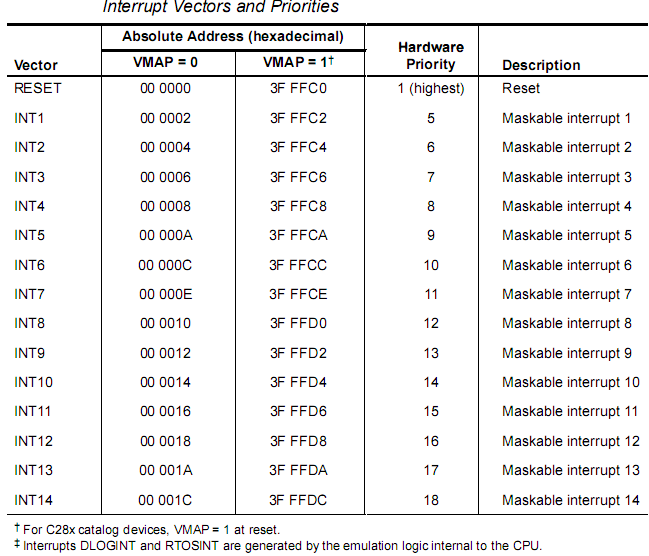
每个中断向量是一个22位的地址，该地址是相应中断服务程序ISR的入口地址。

每个向量被保存在两个地址连续的存储器单元中，每个存储器单元为16位，两个共32位。

低地址单元保存中断向量的低16位，高地址单元则保存它的高6位。

当一个中断被确定后，其22位的中断向量被取回，而地址的高10位被忽略。





**可屏蔽中断**

INTl～INTl4 ——14个通用中断

DLOGINT(数据标志中断)和RTOSINT(实时操作系统中断)是为仿真而设计的中断。

可屏蔽中断也利用状态寄存器STl的D0位，即中断全局屏蔽位INTM，进行全局使能中断和关闭中断。

当INTM＝0时，这些中断全局使能；

当INTM＝1时，这些中断全局关闭。

可以利用SETC INTM和CLRC INTM指令对INTM进行置1和清0.

**非屏蔽中断**

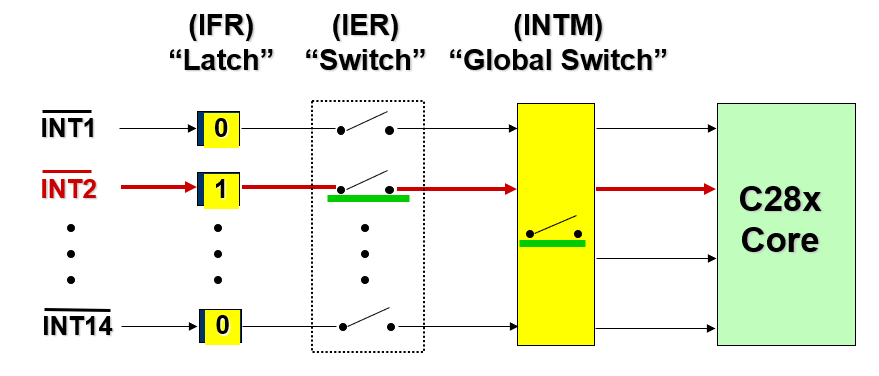
1. 软中断：INTR和TRAP指令

2. 硬件中断：NMT

3. 非法指令陷阱

4. 硬件复位中断 RS

**可屏蔽中断处理过程**



**可屏蔽中断的标准操作**

1. 向CPU发出中断请求。
2. 置位相应的IFR标志位。
3. 当 IER中的相应位是1；STl中的INTM位是0，确认中断；一旦一个中断被使能并且被CPU确认，则其他的中断就不能得到响应直到CPU开始执行中断服务程序，即中断响应的步骤12。
4. 清除相应的IFR位。
5. 清空流水线。

**三个中断寄存器**

中断标志寄存器IFR——l6位寄存器IFR包含的标志位，表明相应中断在等待CPU的确认。

外部输入线INTl～INTl4在CPU的每一个时钟周期都被采样。如果识别出一个中断信号，IFR相应的位就被置位和锁存。

若一个可屏蔽中断等待CPU响应，则IFR的相应位自动置1，否则IFR的相应位是0。

**外设中断扩展模块PIE**

CPU支持17个CPU级硬件中断，包括1个NMI和16（INT1~INT14,RTOSINT, DLOGINT）个可屏蔽中断请求。

F2812有许多个外设，每个外设都可以产生一个或多个外设级中断请求。

在CPU没有足够能力去处理所有外设的中断请求，所以需要一个集中的外设中断控制器PIE，来对各种中断源的请求进行管理和仲裁。

**PIE控制器概述**

外设中断扩展模块PIE把许多中断源多路复用成一个较小的中断输入集，使F2812能够管理更多的中断。

功能：PIE模块支持96个不同的中断，这些中断分成12个组，每个组有8个中断，每个组都被反馈到CPU内核的12条中断线(INTl～INTl2)的一条上。

这96个中断中的每一个都得到了各自向量的支持，这些向量被保存在专用RAM块中并可以进行修改。在PIE模块中可对每个中断分别使能或者使之无效。

1.外设级中断：

某个外设产生中断时，与该事件相关的中断标志(IF)位会在这个外设的寄存器中置为1。如果相应的中断使能(IE)位已经置位，则外设向PIE控制器产生一个中断请求。

如果该中断在外设级使能无效，则相应的IF位会一直保持直到用软件清除它为止。如果在以后使能该中断，且中断标志仍然置位，那么就会向PIE发出一个中断请求。

外设寄存器中的中断标志必须采用软件清除。

2. PIE级中断：

PIE块复用了8个外设和外部引脚的中断进入1个CPU中断。

这些中断被划分为12个组：PIE组1～PIE组12，1个组中的中断被多路复用进入1个CPU中断。

例如，PIE组1被多路复用进入CPU中断INT1，而PIE组12被多路复用进入CPU中断INT12。

与CPU剩余的中断相连接的中断源不是多路复用的

对于非多路复用的中断而言，PIE直接向CPU传送中断请求。

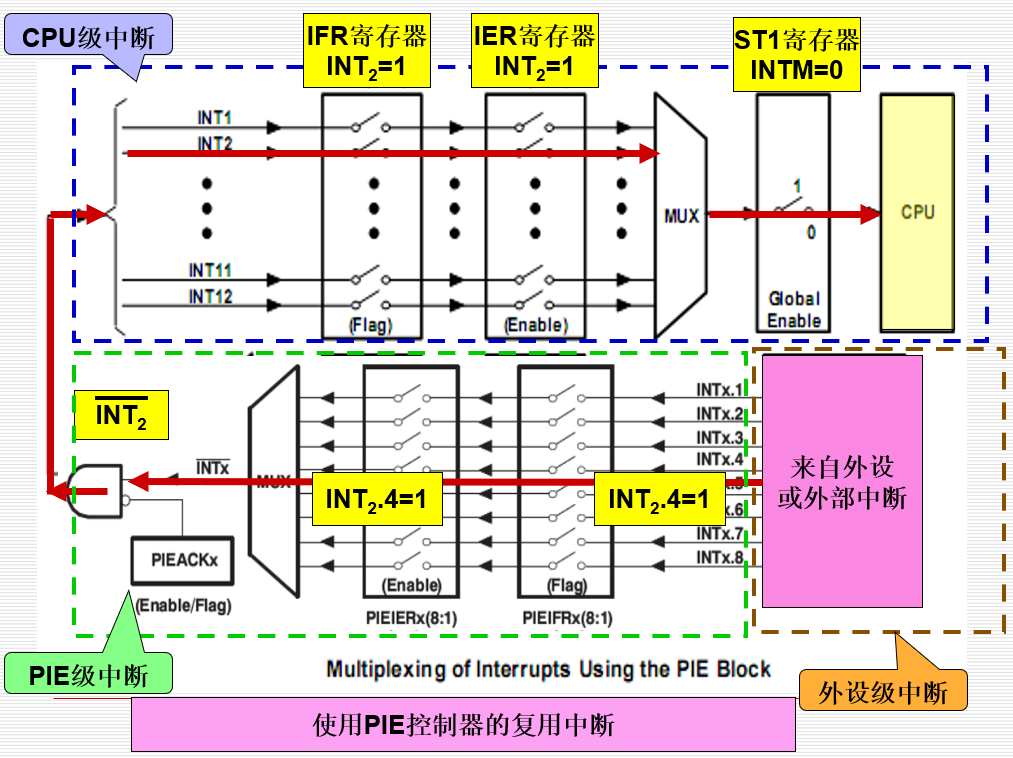
对于多路复用的中断源，PIE块中的每个中断组都有一个相关标志位PIEIFRx.y和使能位PIEIERx.y。

另外，每个中断组(INT1～INTl2)都有一个应答位PIEACKx。

3. CPU级中断：

一旦某个中断请求被送往CPU，CPU级中与INTx相关的中断标志(IFR)位就被置位。

该标志位被锁存在IFR后，CPU不会马上就去执行相应的中断，而是等待CPU使能IER寄存器，并对全局中断屏蔽位INTM进行适当的使能。



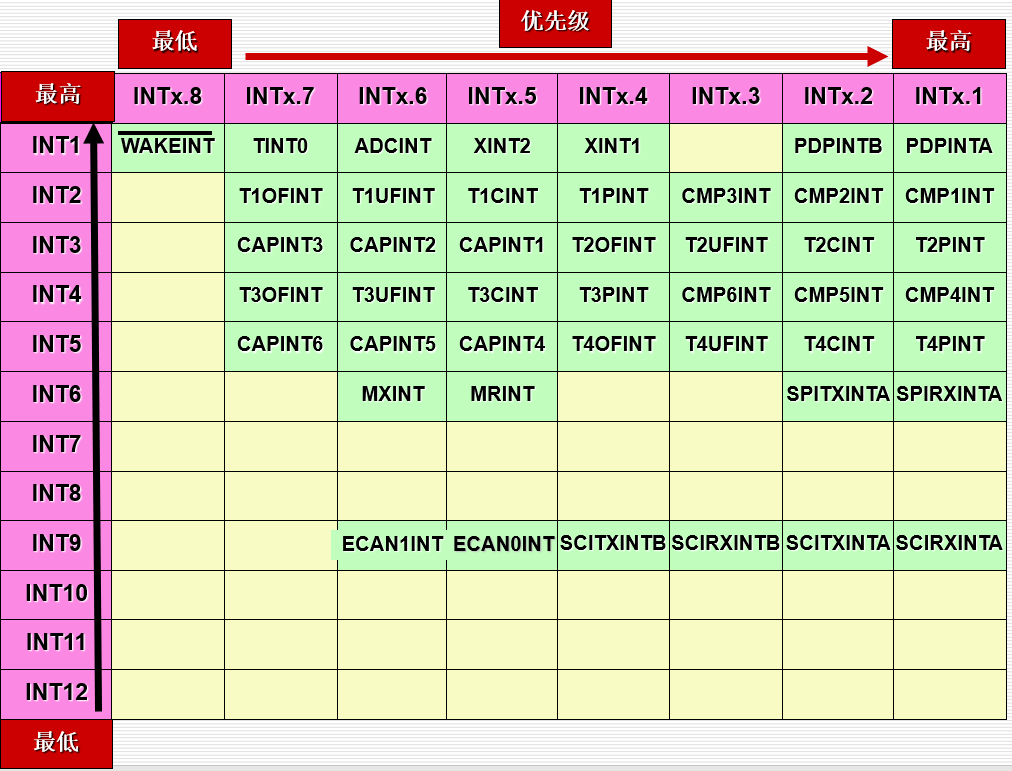
**PIE向量表**

PIE向量表由256×16位的SARAM快组成。

CPU对INT1~INT12的优先级进行定位。

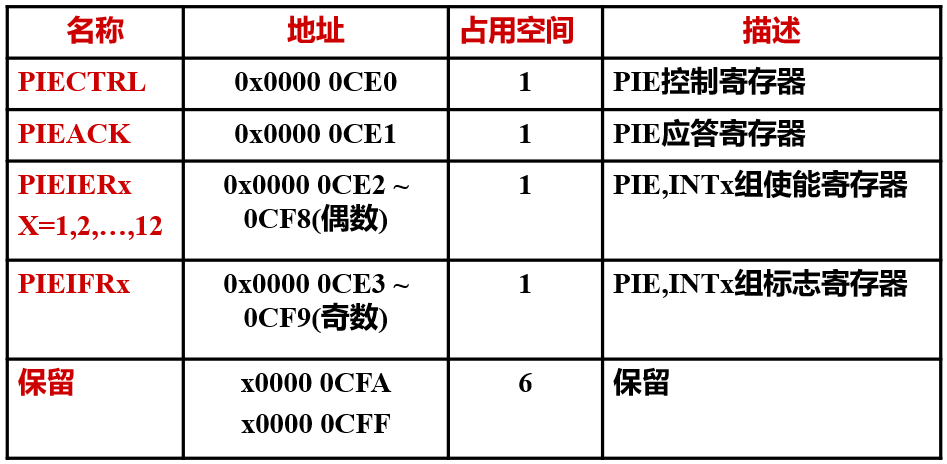
INT1的优先级最高，INT12的优先级最低

PIE每组中INTx.1的优先级最高， INTx.8的优先级最低。

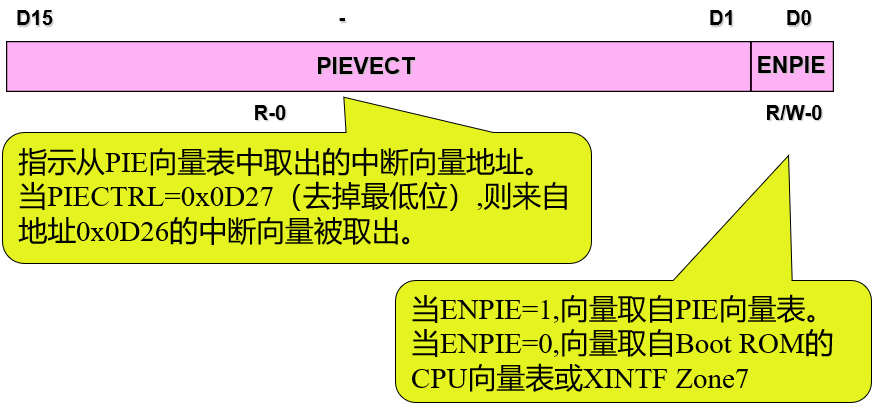


注：表中的每一行表示8个中断复用为一个特定的CPU中断

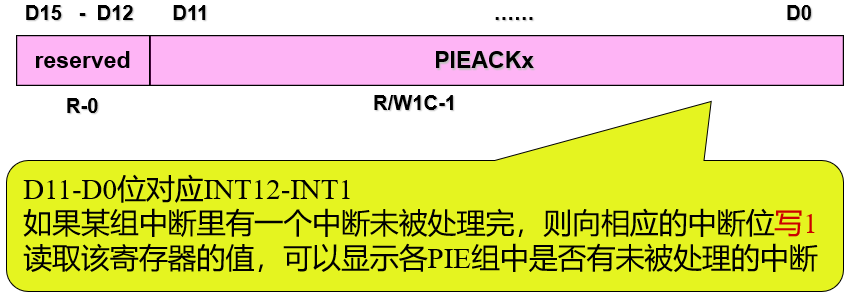
**PIE寄存器**



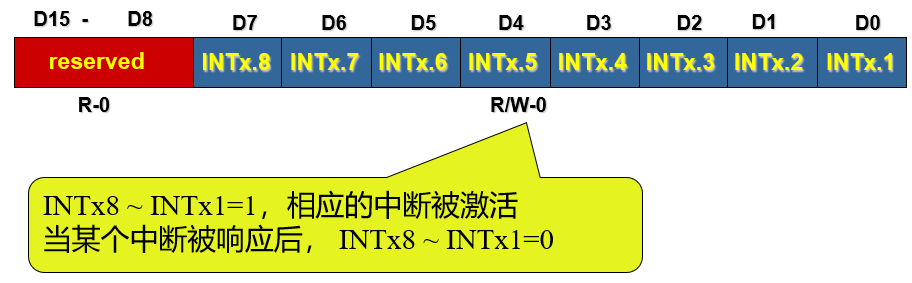
1. PIE控制寄存器（PIECTRL）



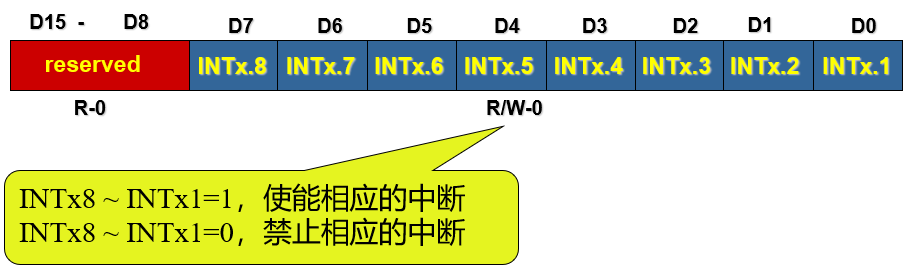
1. PIE中断应答寄存器PIEACK



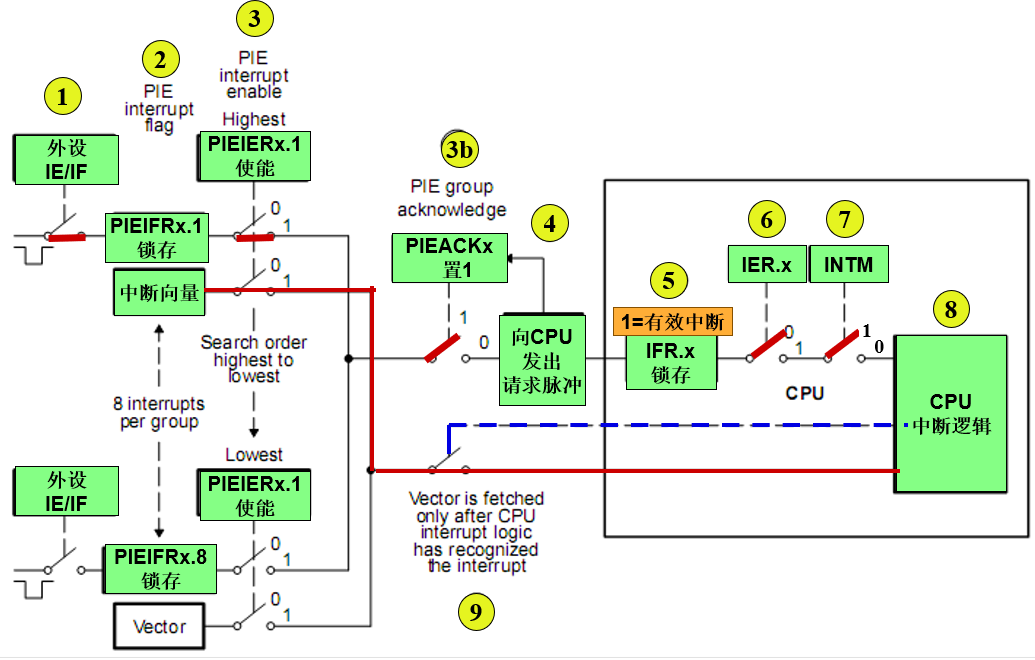
1. PIE中断标志寄存器PIEIFRx （ x = 1 to 12 ）



1. PIE中断使能寄存器PIEIERx（ x = 1 to 12 ）



**中断请求的流程**

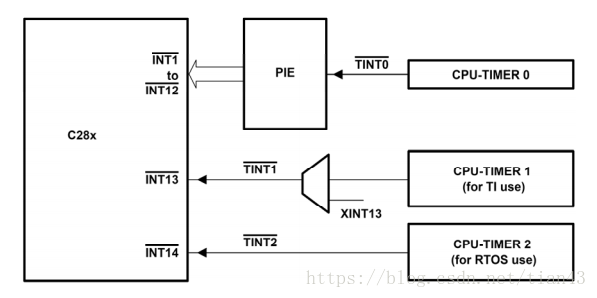


**定时器简介**

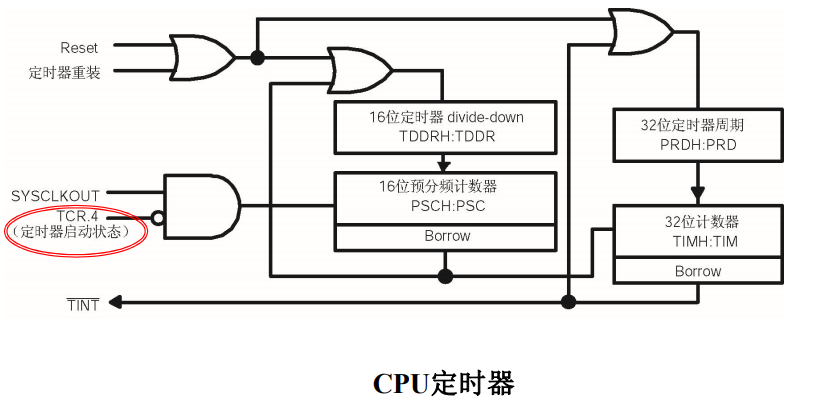
F2812有3个32位CPU定时器(TIMER0/1/2)：

定时器0用户可以在应用程序中使用。

定时器1和定时器2预留给实时操作系统使用(例如 DSP-BIOS)。



**定时器工作原理**



在每个SYSCLKOUT脉冲后PSC减1，一直减到0。

在下一个SYSCLKOUT周期 ，TDDR加载新的除数值到PSC，并使TIM减1。

重复前两步操作，即PSC每次减到0后，TIM进行一次减1操作，直到TIM减为0

在下一个SYSCLKOUT周期，将定时器中断（TINT）送到CPU，和TOUT引脚。同时TIM装载来自PRD的新的定时 计数器值，并使PSC再次减1。

定时器的频率计算

每经过（TDDR+1）个SYSCLKOUT 周期，TIM减1。当PRD、TDDR或两者都不为零时，定时器中断频率即TINT的频率（fTINT）为：



**定时器相关寄存器**

1. 定时器计数器 (TIMERxTPR/ TIMERxTPRH)
2. CPU定时器计数器寄存器TIMH:TIM。TIM寄存器保存当前定时器计数器的低16位，TIMH寄存器保存当前定时器计数器的高16位。
3. 每隔（TDDRH:TDDR+1）个时钟脉冲TIMH:TIM减1，直到减1为0时，被自动重装载（PRDH:PRD）寄存器保存的周期值，并产生定时器TINT中断信号
4. 定时器周期寄存器 (TIMRxPRD / TIMRxPRDH)
   1. CPU定时器周期寄存器PRDH:PRD。PRD寄存器保存周期值的低16位，PRDH寄存器保存周期值的高16位。
   2. 当TIMH:TIM递减到0时，在下次定时周期开始之前， TIMH:TIM寄存器将自动重新装载PRDH:PRD寄存器保存的周期值。
5. 定时器控制寄存器 (TIMERxTCR)

TCR功能：

1. 控制定时器模式
2. 重新加载定时器
3. 启动和停止定时器



CPU定时器中断标志位：

当定时器计数器减到0时，**TIF**=1

CPU定时器中断使能位：

当**TIE**=1时，如果定时器计数器减到0，则定时器将会向CPU申请中断



定时器重新装载控制器：

当TRB=1时，TIMH:TIM会自动重装载PRDH:PRD寄存器保存的周期值，并且预定标寄存器PSCH:PSC装载定时器分频寄存器TDDRH:TDDR中的值

CPU定时器停止状态位：

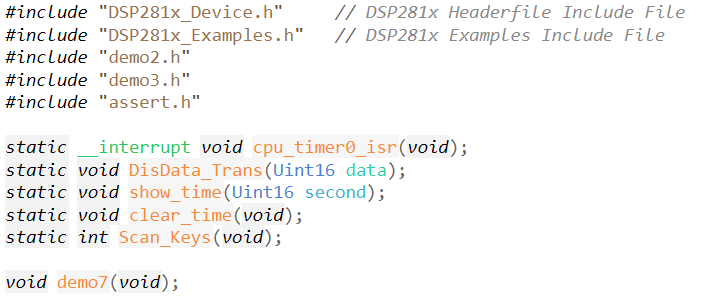
TSS=0, 启动定时器

TSS=1，停止定时器

**实验过程**

代码编写

头文件



主程序

|  |
| --- |
| **代码：** |
| **说明：**  全局变量的声明  msg：数组存放数码管各个数字的显示编码  DisData\_Bit数组：用于存放拆分的四位显示数字  time\_T：显示1s的时间周期，改变这个变量可以改变显示1s的实际经过时间  time\_show：存放将要显示在数码管上的秒数  time\_couter：存放定时器触发中断的次数  counter\_is\_stop：计时是否停止的标志位 |

各功能函数

|  |
| --- |
| **代码：** |
| **说明：**  数据转换函数：对将要显示在数码管上的四位数字拆分开，分别存放 |

|  |
| --- |
| **代码：** |
| **说明：**  数码管位选函数：  在扫描显示数字的过程中，需要选择不同的数码管进行输出。  该函数通过传入的参数(0…3)分别选通4位数码管 |

|  |
| --- |
| **代码：** |
| **说明：**  按键扫描函数：  拉高SW3 SW6 SW9 按键这一列的GPIO输出电压；  读取GPIO\_B的电压，根据所读的二进制数，可知哪几个按键被按下；  返回读取到的数字 |

|  |
| --- |
| **代码：** |
| **说明：**  时间显示函数：  限制所显示的最高秒数位 1000s  数据转换，将一个十进制数的个位 十位 百位 千位分别取出，分别存入数组中  使用一个for循环，读取数组中存储的数字，选通数码管，通过spi 写入数字的位码；  延时一段时间，以适应人眼； |

|  |
| --- |
| **代码：** |
| **说明：**  时间清零函数：  将计时计数器和显示的秒数置0 |

|  |
| --- |
| **代码：** |
| **说明：**  Cpu\_timer 中断处理函数：  判断计时停止位，确定没有被设为停止；  将时间中断计数变量加一；  如果 时间中断计数变量 为 time\_T (1s周期) 的整数倍，则说明经过了新的一秒，  数码管上显示的秒数要加一；  最后设置PIE 中断标志位清除，允许接收下一次中断 |

主函数

|  |
| --- |
| **代码：** |
| **说明：**  InitSysCtrl() 初始化系统设置； |

|  |
| --- |
| **代码：** |
| **说明：**  初始化SPI引脚；  InitSysCtrl() 初始化系统设置；  Init\_LEDS\_Gpio() 设置数码管GPIO；  Init\_Keys\_Gpio() 设置按键 GPIO； |

|  |
| --- |
| **代码：** |
| **说明：**  对系统中断进行一些初始化工作；  禁止CPU全局中断；  初始化PIE控制寄存器到他们的默认状态，就是禁止PIE中断及清除所有PIE中断标志；  禁止CPU中断，清除CPU中断标志位；  初始化PIE中断向量表； |

|  |
| --- |
| **代码：** |
| **说明：**  始化片内外设；  初始化Spi FIFO；  初始化 SPI；  PIE向量表初始化：PIE中断向量指向DSP281x\_DefaultIsr.c中定义的默认空函数模版。需要指向自定义中断函数 (cpu\_timer0\_isr) |

|  |
| --- |
| **代码：**    InitCpuTimers() 函数 |
| **说明：**  初始化CPU 定时器：  CpuTimer0Regs.PRD.all = 0xFFFFFFFF; 将PRD设置到最大数值  CpuTimer0Regs.TCR.bit.TSS = 1; 使计时器停止  CpuTimer0Regs.TCR.bit.TRB = 1; 重置计时器计数器的寄存器  可以看到：CpuTimer2 是用于DSP BIOS 或者其他实时操作系统的  配置CpuTimer: 设置运行频率为150MHz，中断触发周期为 200ms  StartCpuTimer0()：cpu计数器开始运行；    IER |= M\_INT1; 开启CPU INT1    PieCtrlRegs.PIEIER1.bit.INTx7 = 1; 通过查询PIE 中断向量表，TINT0在INT1和INTx.7的位置 |

|  |
| --- |
| **代码：** |
| **说明：**  设置1s 的实际时间周期。因为定时器以0.2s 为周期，准确的1s应该是5个中断周期。 |

|  |
| --- |
| **代码：** |
| **说明：**  在无限循环中，执行数码管的显示，按键扫描。  对按键事件做出反应；  首先一个延时函数对按键进行一个去抖；  当检测到 SW3 按下时，1s时间周期增加，秒表走时变慢  当检测到 SW6 按下时，计时停止标志取反，计时停止  当检测到 SW9 按下时，1s时间周期减少，秒表走时变快  当检测到 SW3 和 SW6 同时按下时，执行计时清零函数，秒表清零 |

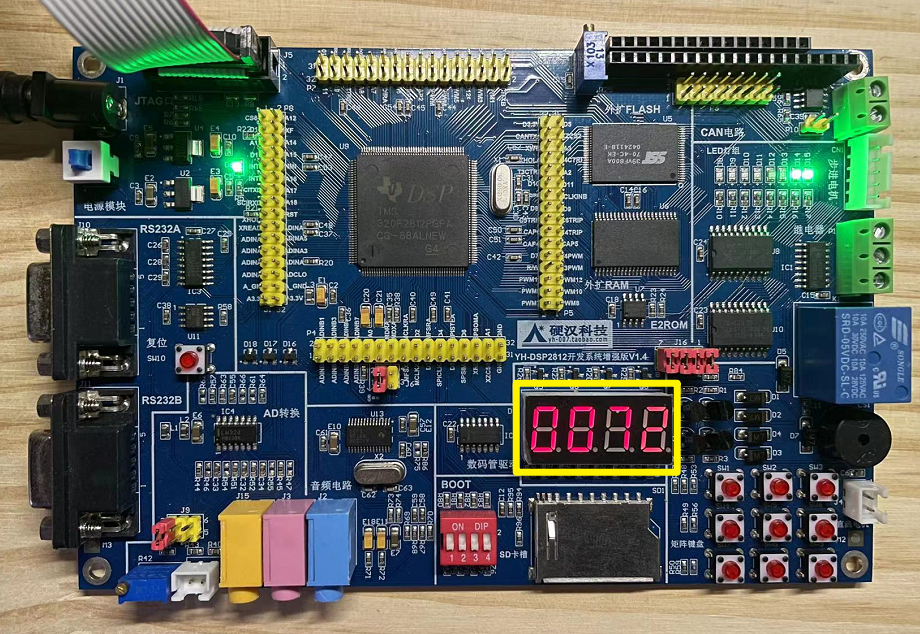
流程图

Diagram, engineering drawing

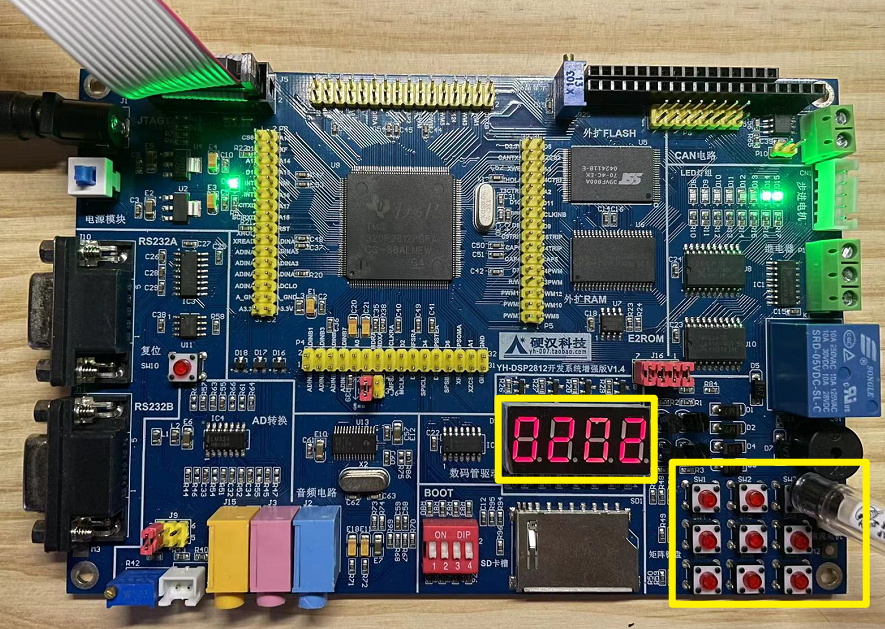
Description automatically generated

**实验结果**

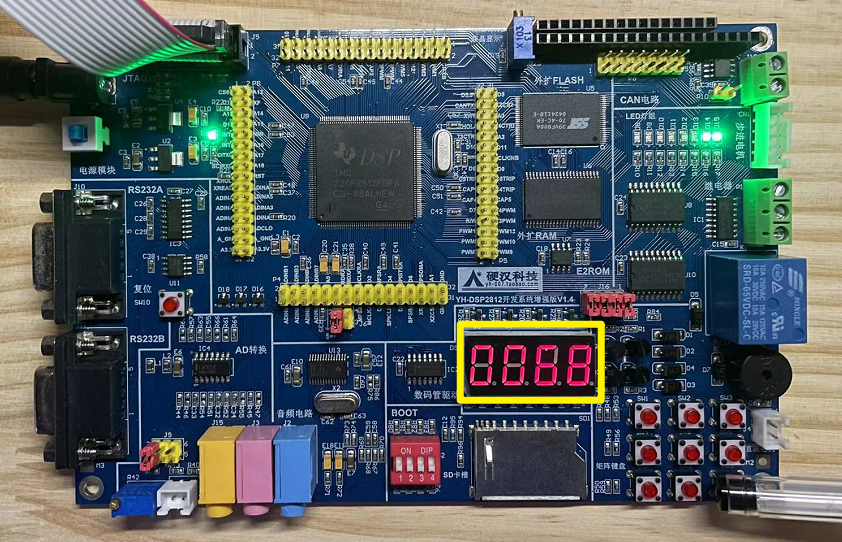
1. 运行程序，可以看到一个从0开始的正常秒表计数。秒数显示在数码管上



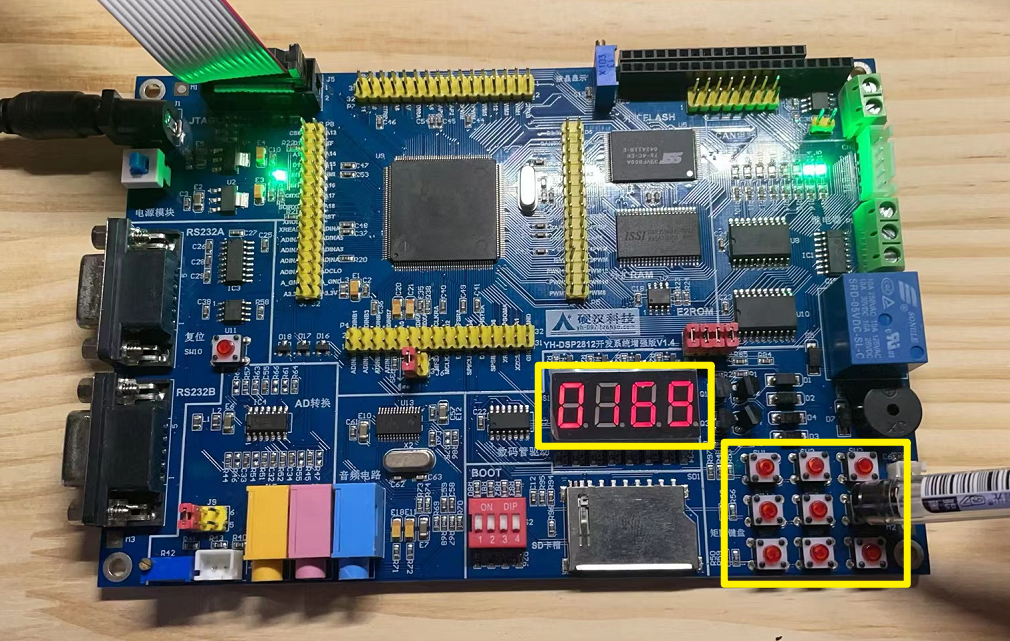
1. 连续按下按键 SW3 ，T\_timer 增加，可以观察到计时变慢



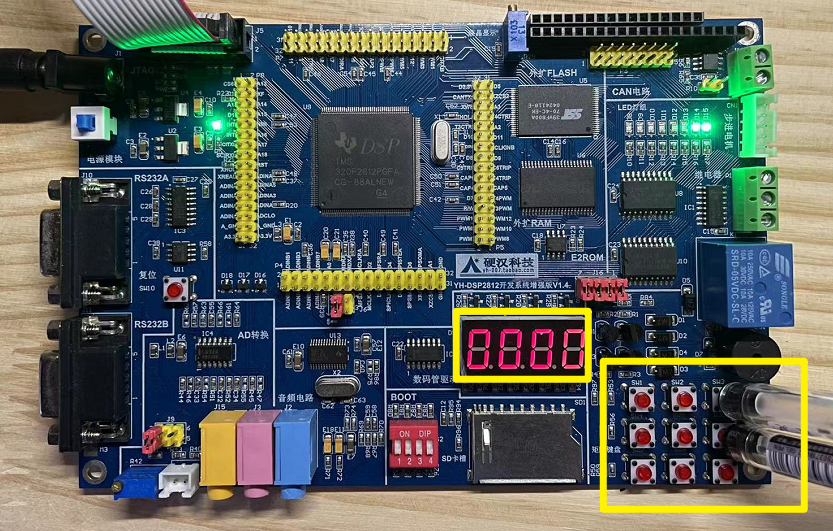
1. 连续按下按键 SW9 ，T\_timer 减小，可以观察到计时变快



1. 按下按键 SW6 ，可以观察到计时停止



1. 同时按下按键SW3 和 SW6 ，可以观察到计时器被清零，从0重新开始计数



心得体会

代码附录

#include "demo7.h"

//// ======================变量声明=========================== ////

static unsigned int msg**[**10**]={**0xC0**,**0xf9**,**0xA4**,**0xB0**,**0x99**,**0x92**,**0x82**,**0xF8**,**0x80**,**0x90**};**

//段码：0~9

static unsigned int DisData\_Bit**[**4**]** **=** **{**0**};** //存放拆分后的四位数字

static unsigned int time\_T**;**

static Uint16 time\_show **=** 0**;**

static unsigned int time\_counter **=** 0**;**

static Uint16 counter\_is\_stop **=** 0**;**

// 主函数

void demo7**(**void**)**

**{**

// step 1. 初始化系统控制:

// 设置PLL, WatchDog, 使能外设时钟

// 下面这个函数可以从DSP281x\_SysCtrl.c文件中找到.

InitSysCtrl**();**

// step 2. 初始化通用输入输出多路复用器GPIO:

// 这个函数在DSP281x\_Gpio.c源文件中被定义了

//初始化SPI

EALLOW**;**

GpioMuxRegs**.**GPFMUX**.**all**=**0x000F**;** // 选择GPIO为SPI引脚

// 端口F MUX - x000 0000 0000 1111

// 初始化Gpio

Init\_LEDS\_Gpio**();** //数码管

Init\_Keys\_Gpio**();** //按键

// step 3. 清除所有中断,初始化中断向量表:

// 禁止CPU全局中断

DINT**;**

// 初始化PIE控制寄存器到他们的默认状态.

// 这个默认状态就是禁止PIE中断及清除所有PIE中断标志

// 这个函数放在DSP281x\_PieCtrl.c源文件里

InitPieCtrl**();**

// 禁止CPU中断，清除CPU中断标志位

IER **=** 0x0000**;**

IFR **=** 0x0000**;**

//初始化PIE中断向量表，并使其指向中断服务子程序（ISR）

// 这些中断服务子程序被放在了DSP281x\_DefaultIsr.c源文件中

// 这个函数放在了DSP281x\_PieVect.c源文件里面.

InitPieVectTable**();**

// step 4.初始化片内外设:

spi\_fifo\_init**();** // 初始化Spi FIFO

spi\_init**();** // 初始化 SPI

// Interrupts that are used in this example are re-mapped to

EALLOW**;**

// This is needed to write to EALLOW protected registers

PieVectTable**.**TINT0 **=** **&**cpu\_timer0\_isr**;**

// ISR functions found within this file.

EDIS**;**

// This is needed to disable write to EALLOW protected registers

// Step 5. Initialize all the Device Peripherals:

// This function is found in DSP281x\_InitPeripherals.c

// InitPeripherals(); // Not required for this example

InitCpuTimers**();**

// For this example, only initialize the Cpu Timers

// Configure CPU-Timer 0 to interrupt :

// 150MHz CPU Freq, 0.2 second Period (in uSeconds)

ConfigCpuTimer**(&**CpuTimer0**,** 150**,** 200000**);**

// ConfigCpuTimer(&CpuTimer0, 150, 500000);

StartCpuTimer0**();**

// Step 6. User specific code, enable interrupts:

// Enable CPU INT1 which is connected to CPU-Timer 0:

IER **|=** M\_INT1**;**

// Enable TINT0 in the PIE: Group 1 interrupt 7

PieCtrlRegs**.**PIEIER1**.**bit**.**INTx7 **=** 1**;**

// PieCtrlRegs.PIEIER2.bit.INTx7 = 1;

// Enable global Interrupts and higher priority real-time debug events:

EINT**;** // Enable Global interrupt INTM

ERTM**;** // Enable Global realtime interrupt DBGM

time\_T **=** 5**;** //default: 5 -> 5\*200ms=1s

// Step 6. IDLE loop. Just sit and loop forever (optional):

**while(**1**){**

show\_time**(**time\_show**);**

int key **=** 0**;**

**if(**Scan\_Keys**()** **!=** 0**){**

delay1**(**25000**);**

**if(**Scan\_Keys**()** **!=** 0**){**

key **=** Scan\_Keys**();**

**}**

**}**

**if(**key **==** 4 **&&** time\_T**<**15**){** //sw3

time\_T**++;**

printf**(**"time\_T:%d\n"**,** time\_T**);**

**continue;**

**}**

**if(**key **==** 2**){** //sw6

counter\_is\_stop **=** **~**counter\_is\_stop**;**

printf**(**"stop/continue\n"**);**

**continue;**

**}**

**if(**key **==** 1 **&&** time\_T**>**1**){** //sw9

time\_T**--;**

printf**(**"time\_T:%d\n"**,** time\_T**);**

**continue;**

**}**

**if(**key **==** 6**){** //sw3+sw6

clear\_time**();**

**continue;**

**}**

**}**

**}**

static void DisData\_Trans**(**Uint16 data**)**

**{**

DisData\_Bit**[**3**]** **=** data **/** 1000**;** //千位数

DisData\_Bit**[**2**]** **=** data **%** 1000 **/** 100 **;** //百位数

DisData\_Bit**[**1**]** **=** data **%** 100 **/** 10**;** //十位数

DisData\_Bit**[**0**]** **=** data **%** 10**;** //个位数

**}**

static void show\_time**(**Uint16 second**)**

**{**

// assert(second >= 0);

**if(**second **>** 1000**)** second **=** 1000**;**

DisData\_Trans**(**second**);**

printf**(**"time:"**);**

int Loop**;**

**for(**Loop **=** 0**;**Loop **<** 4**;**Loop**++){**

Sellect\_Bit**(**Loop**%**4**);**

//选择要扫描的数码管位

spi\_xmit**(**msg**[**DisData\_Bit**[**Loop**%**4**]]);**

//串行输出要显示的数字

**if(**Loop **<** 4**)** printf**(**"%d"**,** DisData\_Bit**[**3**-**Loop**%**4**]);**

delay1**(**1500**);**

**}**

printf**(**"s\n"**);**

**}**

static void clear\_time**(**void**){**

time\_show **=** 0**;**

time\_counter **=** 0**;**

printf**(**"clear\n"**);**

**}**

static int Scan\_Keys**(**void**){**

GpioDataRegs**.**GPBCLEAR**.**all **=** 0x1**;** // |= 0000 0000 0000 0001

int key\_press **=** **(~(**GpioDataRegs**.**GPBDAT**.**all **>>** 3**)** **&** 0x07**);**

// printf("keys:%d\n",key\_press);

**return** key\_press**;**

**}**

void print\_bin1**(**unsigned int number**){**

int bit **=** **sizeof(**unsigned int**)\***8**;**

int i**;**

**for(**i **=** bit **-** 1**;**i **>=** 0**;**i**--){**

int bin **=** **(**number **&** **(**1 **<<** i**))** **>>** i**;**

printf**(**"%d"**,** bin**);**

**}**

printf**(**"\n"**);**

**}**

static \_\_interrupt void cpu\_timer0\_isr**(**void**)**

**{**

**if(**counter\_is\_stop **==** 0**){**

time\_counter**++;**

// GpioDataRegs.GPACLEAR.all = 0x00FF;

**if(**time\_counter **%** time\_T **==** 0**)** **{**

time\_show**++;**

**}**

**}**

// printf("time\_counter:%d\n", time\_counter);

// Acknowledge this interrupt to receive more interrupts from group 1

PieCtrlRegs**.**PIEACK**.**all **=** PIEACK\_GROUP1**;**

**}**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指导教师批阅意见：  验证性实验（报告）评分细则表   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 评分项目 | | 满分标准 | | | 成绩 | | 权重 | | 原始数据 | | 准确、真实可信，记录完整 | | |  | | 40％ | | 实验数据分析与处理 | | 分析与处理正确，有必要的过程，能恰当运用图表，分析全面、正确结论合理。 | | |  | | 20％ | | 实验报告 | | 实验报告格式规范，内容完整 | | |  | | 10％ | | 撰写质量 | | 撰写认真、报告整洁、清晰 | | |  | | 10％ | | 实验心得与思考题 | | 有心得体会，完成思考题 | | |  | | 20％ | | A（100~85） | B（84~75） | | C（74~65） | D（64~60） | | F（<60） | |   成绩评定：  指导教师签字：  年 月 日 |
| 备注： |

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。