作业1

查阅文献，检索DSP（数字信号处理芯片）、GPP（通用处理器）、MCU（微控制器 ）在结构、特点、功能以及用途上的区别。

我的答案：

DSP是一种专门用于处理数字信号的处理器，它具有一些优化的指令和寻址模式，可以高效地执行诸如乘法累加、循环缓冲、饱和运算等操作。DSP通常采用哈佛结构，可以同时访问两个或更多的存储空间，以提高数据吞吐量。DSP的运算能力强，擅长很多的重复数据运算，例如滤波、变换、编解码等。

GPP是一种通用的处理器，它可以执行各种类型的程序和任务，但不一定针对特定的应用领域进行优化。GPP通常采用冯诺依曼结构，只有一个存储空间，需要通过总线来传输指令和数据。GPP的运算能力较弱，不擅长处理大量的数字信号运算，但可以适应不同的信息源和数据格式。

MCU是一种集成了片上外围器件（如存储器、定时器、串行接口等）的微型计算机，它可以执行一些简单的控制和逻辑功能，但不具备复杂的运算能力。MCU通常也采用冯诺依曼结构，但有一些特殊的寄存器和指令来控制外围器件。MCU的运算能力较低，不适合处理高速或高精度的数字信号运算，但侧重于控制和低功耗。

DSP、GPP和MCU之间的应用领域也有所不同。DSP主要用于信号处理、通信、多媒体等领域，需要高速、高性能、高精度的数字信号运算；GPP主要用于计算机、服务器、智能手机等领域，需要灵活、多功能、兼容性好的通用计算；MCU主要用于嵌入式系统、物联网、工业控制等领域，需要简单、可靠、低成本、低功耗的控制功能。

举个例子：假如你想设计一个智能音箱，它可以通过语音识别和语音合成来与用户交互，同时也可以控制一些外部设备，例如灯光、空调等。那么可能需要同时使用这三种处理器，其中DSP用于处理音频信号，例如对声音进行采样、滤波、降噪、增强等，提高语音识别和语音合成的效果和质量；GPP用于运行语音识别和语音合成的算法，例如将声音转换为文本，或将文本转换为声音，实现准确识别用户的自然语言；MCU用于控制外部设备，例如通过串行接口或无线通信来发送或接收控制信号，实现对灯光、空调等的开关或调节。

作业2

1

用Q12法完成以下数据的转换（d表示十进制数据，f表示十六进制数据）：

（-6.5）d = 0x（f）

（0.049）d = 0x（f）

0x8004 = （d）

0x14A8 = （d）

$$-6.5 \times 2^{12} = -26624 \approx 0x9800$$

$$0.049 \times 2^{12} = 200.704 \approx 0x00C8$$

$$0x8004 \times 2^{-12} = -32764 \times 2^{-12} = -7.9990$$

$$0x14A8 \times 2^{-12} = 5288 \times 2^{-12} = 1.2910$$

2

16位的定点数据中，当数据范围是[-9.4,1.5]时，最合适用Q多少表示法来确保精度?

我的答案:

对于16位的定点数据，

使用Q12法得到的浮点数范围是[-8,7.9997559]，

Q11法得到的浮点数范围是[-16,15.9995117]，

我们的数据范围是[-9.4,1.5]，为了确保精度应该使用Q11法。

3

完成IEEE 单精度（32位）浮点数据的转换（D表示十进制数据，IEEE_siglefloatpoint用十六进制表示）：

（100）D = 0x（IEEE_siglefloatpoint）

（0.049）D = 0x（IEEE_siglefloatpoint）

（-31）D = 0x（IEEE_siglefloatpoint）

（-126.843）D = 0x（IEEE_siglefloatpoint）

0x3ff80000 = （ ）D

0xbe000000 = （ ）D

我的答案:

（100）D = 0x42 C8 00 00（IEEE_siglefloatpoint）

（0.049）D = 0x3D 48 B4 39（IEEE_siglefloatpoint）

（-31）D = 0xC1 F8 00 00（IEEE_siglefloatpoint）

（-126.843）D = 0xC2 FD AF 9D（IEEE_siglefloatpoint）

0x3ff80000 = （ 1.9375 ）D

0xbe000000 = （ -0.125 ）D

作业3 pll初始化

```
//---------------------------------------------------------------------------
 // Example: InitPll:
 //---------------------------------------------------------------------------
 // This function initializes the PLLCR register.
 void InitPll(Uint16 val, Uint16 divsel)
 {
    // 检查 PLL 是否由于缺失外部时钟信号处于跳闸模式
    while(1)
    {
    if (SysCtrlRegs.PLLSTS.bit.MCLKSTS != 0)
    {
        // 如果 PLL 处于跳闸模式，则继续检查
        continue;
    }
    break;
 }
    // 在更改 PLLCR 寄存器之前，将 DIVSEL 寄存器设置为 0。
    if (SysCtrlRegs.PLLSTS.bit.DIVSEL != 0)
    {
        EALLOW;
        SysCtrlRegs.PLLSTS.bit.DIVSEL = 0;
        EDIS;
    }
    // 如果 PLLCR 寄存器的新值与当前值不同，则更改 PLLCR 寄存器的值
    if (SysCtrlRegs.PLLCR.bit.DIV != val)
    {
        EALLOW;
        // 在设置 PLLCR 寄存器之前，关闭缺失时钟检测逻辑
        SysCtrlRegs.PLLSTS.bit.MCLKOFF = 1;
        SysCtrlRegs.PLLCR.bit.DIV = val;
        EDIS;
        DisableDog();
        while (SysCtrlRegs.PLLSTS.bit.PLLLOCKS != 1)
        {
            // 取消注释以服务看门狗
            // ServiceDog();
        }
        EALLOW;
        SysCtrlRegs.PLLSTS.bit.MCLKOFF = 0;
        EDIS;
    }
    if ((divsel == 1) || (divsel == 2) || (divsel == 3))
    {
        EALLOW;
        SysCtrlRegs.PLLSTS.bit.DIVSEL = divsel;
        EDIS;
    }
 }
```

作业4中断

1. 以下是LAB11_main.c中与中断设置相关的程序语句：

```
DINT; //禁止 CPU中断，禁止全局中断
InitPieCtrl(); //初始化PIE控制寄存器
IER=0x0000; //禁用所有CPU中断并清除CPU中断标志位
IFR=0x0000;
InitPieVectTable(); //初始化PIE向量表 里面包含了 PieCtrlRegs.PIECTRL.bit.ENPIE=1
PieVectTable.EPWM1_INT =&epwm1_timer_adc_isr; //第三组第一中断
PieCtrlRegs.PIEIER3.bit.INTx1 = 1; // Enable ADCINT interrupt in PIE
PieCtrlRegs.PIECTRL.bit.ENPIE=1; //打开PIE中断,使能PIE
IER |= M_INT3; //打开CPU第3组中断
EINT; //使能全局中断，允许中断响应
```

2. 函数InitPieCtrl()实现的功能

函数InitPieCtrl()的功能是将所有的 PIE 控制寄存器初始化为已知状态。它首先禁止了 CPU 级别的中断，并将所有的 PIEIER 和 PIEIFR 寄存器清零，以确保在启用 PIE 之前没有未处理的中断。处理完这些之后，它将 PieCtrlRegs.PIECTRL.bit.ENPIE 设置为0，禁用整个 PIE。

3. 函数InitPieVectTable()实现的功能

函数InitPieVectTable()的功能是将PIE向量表初始化为一个已知状态。

它首先使用两个指针达成了用PieVectTableInit数组的值初始化PieVectTable的目的，然后使能了PIE向量表（PieCtrlRegs.PIECTRL.bit.ENPIE = 1）。

4. 语句PieVectTable.EPWM1_INT =&epwm1_timer_adc_isr; 实现的功能。

这个语句将epwm1_timer_adc_isr函数的地址赋值给EPWM1_INT中断向量，即设置EPWM1_INT中断服务程序的入口地址为epwm1_timer_adc_isr，当EPWM1产生中断时，CPU会跳转到epwm1_timer_adc_isr函数的地址处执行。这样就实现了将用户编写的中断服务函数与相应中断向量关联起来的功能。