第 1 章

1. 讨论 DSP 的含义

①数字信号处理：采用计算机技术将信号以数字形式进行采集、变换、滤波、估值、增强、压缩以及识别等处理的理论与方法

②数字信号处理器：适合数字信号处理的微处理器

2. 说明 DSP 芯片的特征

采用哈弗结构、流水线技术、硬件乘法器、和特殊的 DSP 指令

3. 比较哈弗结构和冯诺依曼结构的不同

哈弗结构将程序和数据存储在不同的存储空间，对程序和数据独立编址，独立访问

冯诺依曼结构只使用一个存储器存储程序和数据且统一编址

5.如何理解 TMS320F28027 芯片的“小成本，大集成”？

TMS320F28027 是一款 C2000™ 32 位微控制器，具有 60MHz 的频率和 64KB 的闪存。它针对处理、感应和驱动进行了优化，可提高实时控制应用的闭环性能。

它集成了一个高性能的数字信号处理器(DSP)内核，可以执行复杂的算法和控制逻辑。

它集成了一个高效的模拟电路，包括 12 位 ADC、比较器、DAC、PWM 等，可以实现精确的信号采集和输出。

它集成了一个灵活的通信接口，包括 SPI、SCI、I2C 等，可以与外部设备进行数据交换。

它集成了一个板载 USB JTAG 仿真器，可以方便地对芯片进行编程和调试

第 2 章

1.简述 C28XCPU 组成

主要由算数逻辑单元(ALU)、乘法器、移位器、寄存器、地址寄存器算数单元（ARAU）、6 组总线、程序地址产生逻辑以及程序控制逻辑等组成，还包括一些指令队列和指令译码单元、中断管理逻辑单元等。

2.简述 C28xDSP 总线结构

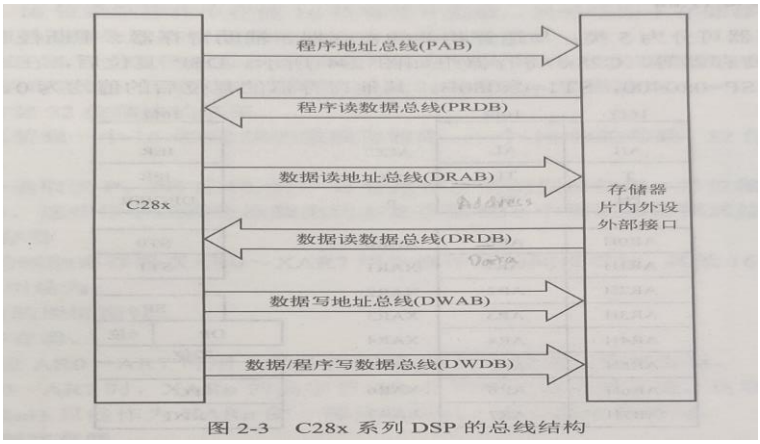


图 2-3　C28x 系列 DSP 的总线结构

在 C28x 系列 DSP 中，将程序存储器和数据存储器以及读和写分别采用不同的总线操作，这样得到了 6 组总线：PAB、PRDB、DRAB、DRDB、DWAB 和 DWDB，其中，PAB、DRAB 和 DWAB 为 3 组地址总线，PRDB、DRDB 和 DWDB 为 3 组数据总线。这 6 组总线的定义如下：

PAB：程序地址总线，22 位，传送对程序存储器读写的地址。

DRAB：数据读地址总线，32 位，传送从数据存储器读数据的地址。

DWAB：数据写地址总线，32 位，传送向数据存储器写数据的地址。

PRDB：程序读数据总线，32 位，传送从程序存储器读取的指令。

DRDB：数据读数据总线，32 位，传送从数据存储器读取的数据。

DWDB：数据/程序写数据总线，32 位，传送向数据存储器或程序存储器写入的数据。

3.什么是 DSP 的原子操作？

DSP 的原子操作是指一种不可分割的指令，即在执行过程中不会被其他指令打断，保证了指令的完整性和正确性。DSP 的原子指令可以用于多核处理器的环境下编程，保证多个核访问和修改同一个变量资源时，不会出现读改写不相一致的情况。

5 如何理解 DSP 中的单指令周期

C28X 的流水线操作分为 8 个阶段，而每一条指令占据 8 个周期，每一条指令执行的有效时间是单周期的，即单指令周期。

第 4 章

1.TMS320C28X 的 C 语言中常用的重要标识符有哪些？举例说明其用法。

2．重要标识符

（1）const　优化存储器的分配，表示变量内容为常数，不会改变。

使用：const+数据类型+变量名

举例：const char tab[1024]={显示数据};

（2）volatile

用于声明存储器或外设寄存器，以此来说明所定义的变量是“可变的”，是可以被 DSP 的其他硬件修改的，而不仅仅是由 C 程序本身修改。

使用：volatile+数据类型+变量名；

举例：volatile struct SYS_CTRL_REGS

（3）cregister　允许采用高级语言直接访问控制寄存器。

使用：cregister+数据类型+变量名；

举例：extern cregister volatile unsigned int IFR;

extern cregister volatile unsigned int IER;

（4）interrupt　用于说明函数是中断函数，这样编译器会自动为中断函数产生保护现场和恢复现场所需执行的操作。

使用：interrupt+void+中断函数名(void);

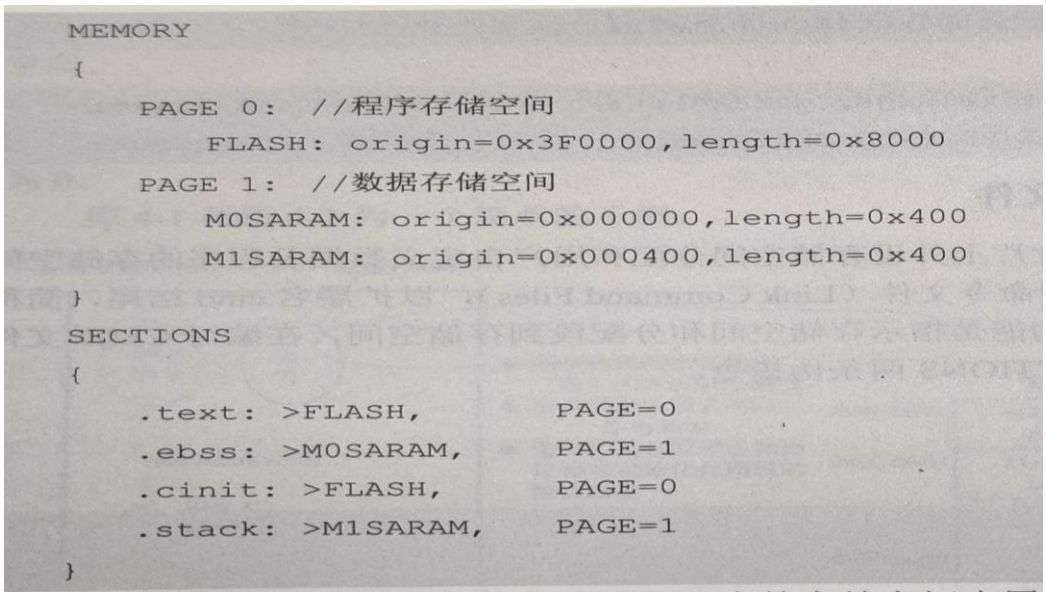
举例：interrupt void XINT1_ISR(void);

2. 编译器产生的初始化段和非初始化段各有那些？

初始化段：.tsxt 段、.cinit 段、.pint 段、.const 段、econst 段、.switch 段

非初始化段：.bss 段、.ebss 段、.stack 段、.sysmen 段、.esysmen 段

3. 编写 F28027.CMD 文件，把.ebss 放到 M0，.stack 放到 M1



```
MEMORY
{
    PAGE 0:  //程序存储空间
        FLASH: origin=0x3F0000,length=0x8000
    PAGE 1:  //数据存储空间
        M0SARAM: origin=0x000000,length=0x400
        M1SARAM: origin=0x000400,length=0x400
}
SECTIONS
{
    .text: >FLASH,        PAGE=0
    .ebss: >M0SARAM,      PAGE=1
    .cinit: >FLASH,       PAGE=0
    .stack: >M1SARAM,     PAGE=1
}
```

第 5 章

1.如何配置 CPU 时钟频率？

①选择时钟源：时钟源可以是内部晶体振荡器、外部晶体振荡器、外部直接时钟输入

②配置 PLL 控制寄存器的 DIVSEL 设置 PLL 的倍频因子

3.如何配置 CPU 定时器计数频率

在寄存器 TDDRH：TDDR 中装在分频系数，然后预分频计数器 PSCH:PSC 将基于系统时钟进行减计数。这样对系统时钟进行分频得到定时器工作时钟。再配置周期寄存器 PRDH:PRD 来配置定时器的定时时间。

4. 看门狗的作用是什么，如何避免它溢出

防止芯片出现软硬件故障，若程序跑飞或进入死循环，看门狗计数器就不能按时被清零，触发系统复位。

用户可以通过向看门狗关键字寄存器 WDKEY 定期写入 0X55+0XAA 序列来避免溢出。或者禁止看门狗。

5.3 个定时器的作用分别是什么

CPU 定时器 0 和 1 可被应用系统使用，定时器 2 为 DSP/BIOS 预留，如果不应用 DSP/BIOS，也可被系统使用。

定时器作用有：定时、计数、产生脉冲、产生定时中断等

6.如何设置 CPU 定时器的定时时间

①配置系统时钟

②配置分频值寄存器 TDDRH:TDDR

③配置周期寄存器 PRDH：PRD，开始计时 PRDH:PRD 中的值加载到计数寄存器 TIMH:TIM 中，TIMH：TIM 进行减计数直到 0.

7.EALLOW 的作用是什么？

EALLOW 保护某些控制寄存器，防止虚假的 CPU 写入。寄存器被保护时，CPU 对受保护 寄存器执行的所有写操作均被忽略

第 6 章

1. 外设中断的响应条件

外设级：外设事件发生，中断标志位（IF）置 1，中断使能（IE）置 1

PIE 级：PIE 中断标志（PIEIFRX.Y）置 1，PIE 中断使能（PIEIERX.Y）置 1，PIE 应答寄存器 PIEACKx 为 0

CPU 级：中断标志寄存器（IFR）置 1、中断允许寄存器（IER）置 1，INTM=0

2. 外设中断扩展模块的作用

外设中断扩展（PIE）模块将许多中断源复用成一个较小的中断输入集合。PIE 将高达 96 个外设中断源每 8 个一组、共 12 个中断信号送给 CPU 的 INT1~INT12。

4. 响应中断后 CPU 如何找到中断服务程序的入口地址？

当 PIE 向量表没有被使能时，中断向量映射模式为引导 ROM 向量模式，从引导 ROM 取中断服务程序的入口地址；

当 PIE 向量表使能时，中断向量将从 PIE 向量表读取，从 PIE 向量表中得到中断服务程序的入口地址。

5. C 编程中如何设置一个中断的中断向量？

先初始化 PIE 向量表，然后将写好的中断服务程序的入口地址填到 PIE 向量表的对应位置

第 7 章

1.AIO2、AIO4、AIO6 配置成输出引脚，且输出低电平。

```
void InitLEDGpio(void) {
EALLOW;
GpioDataRegs.AIOCLEAR.bit.AIO6 = 1；
GpioDataRegs.AIOCLEAR.bit.AIO4 = 1；
GpioDataRegs.AIOCLEAR.bit.AIO2 = 1；
GpioCtrlRegs.AIOMUX1.bit.AIO2 = 0;
GpioCtrlRegs.AIOMUX1.bit.AIO4 = 0;
GpioCtrlRegs.AIOMUX1.bit.AIO6 = 0;
GpioCtrlRegs.AIODIR.bit.AIO2 = 1;
GpioCtrlRegs.AIODIR.bit.AIO4 = 1;
GpioCtrlRegs.AIODIR.bit.AIO6 = 1;
EDIS;
}
```

2. 将 GPIO0~GPIO7 配置成 PWM 功能输出。

```
void InitGPIO(void) {
GpioCtrlRegs.GPAMUX1.bit.GPIO0=0x01;
GpioCtrlRegs.GPAMUX1.bit.GPIO1=0x01;
GpioCtrlRegs.GPAMUX1.bit.GPIO2=0x01;
GpioCtrlRegs.GPAMUX1.bit.GPIO3=0x01;
GpioCtrlRegs.GPAMUX1.bit.GPIO4=0x01;
GpioCtrlRegs.GPAMUX1.bit.GPIO4=0x01;
```

```
GpioCtrlRegs.GPAMUX1.bit.GPIO6=0x01;
GpioCtrlRegs.GPAMUX1.bit.GPIO7=0x01;
GpioCtrlRegs.GPADIR.bit.GPIO0=1;
GpioCtrlRegs.GPADIR.bit.GPIO1=1;
GpioCtrlRegs.GPADIR.bit.GPIO2=1;
GpioCtrlRegs.GPADIR.bit.GPIO3=1;
GpioCtrlRegs.GPADIR.bit.GPIO4=1;
GpioCtrlRegs.GPADIR.bit.GPIO4=1;
GpioCtrlRegs.GPADIR.bit.GPIO6=1;
GpioCtrlRegs.GPADIR.bit.GPIO7=1;
}
```

3. 将 GPIO0~7 配置成一个 8 位输出端口，编写一个函数 void ByteOutput(unsigned char out)，将 out 的低 8 位在 GPIO0~7 上输出。

```
void ByteOutput(unsigned char out) {
for(int i=0;i<8;i++){
if( out&(1<<i) ){ GpioDataRegs.GPASET.all=1<<i;}
   else{ GpioDataRegs.GPACLEAR.all =1<<i;
   }
}
```

或者：

```
void ByteOutput(unsigned char out)
{
GpioDataRegs.GPASET.all    = (unsigned long int)(out & 0xff);
GpioDataRegs.GPACLEAR.all    = (unsigned long int) ((~out) & 0xff) ;
}
```

4. 编写一个外部中断的初始化函数，要求：使用 XINT2 和 GPIO28，有效极性为低电平，中断服务程序名为 Xint2IntISR

```
Void InitXint2 (void){
EALLOW;
GpioCtrlRegs.GPAMUX2.bit.GPIO28=0;
GpioCtrlRegs.GPADIR.bit.GPIO28=0;
GpioCtrlRegs.GPAPUD.bit.GPIO28=0;
GpioIntRegs.GPIOXINT2SEL.all=28;
XIntruptRegs.XINT2CR.bit.POLARITY =0 ; //  下降沿
XIntruptRegs.XINT2CR.bit.ENABLE =1 ;
PieVectTable.XINT2=Xint2IntISR;
PieCtrlRegs.PIECTRL.bit.ENPIE=1;
PieCtrlRegs.PIEIER1.bit.INTx5=1;
EDIS;
}
```

第 8 章

方波的频率和波形对称性检测。包括： 1） 将 eCAP1 配置成：双跳沿捕获，1 分频，绝对模式，每次捕获都中断 2） 变量初始化

```
void IniteCAP ()
{
ECap1Regs.ECCTL1.bit.CAP1POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CAP2POL = EC_FALLING;
ECap1Regs.ECCTL1.bit.CAP3POL = EC_RISING;
ECap1Regs.ECCTL1.bit.CAP4POL = EC_FALLING;
ECap1Regs.ECCTL1.bit.CTRRST1 = EC_ABS_MODE;
ECap1Regs.ECCTL1.bit.CTRRST2 = EC_ABS_MODE;
ECap1Regs.ECCTL1.bit.CTRRST3 = EC_ABS_MODE;
ECap1Regs.ECCTL1.bit.CTRRST4 = EC_ABS_MODE;
ECap1Regs.ECCTL1.bit.CAPLDEN = EC_ENABLE;
ECap1Regs.ECCTL1.bit.PRESCALE = EC_DIV1;
ECap1Regs.ECCTL2.bit.CAP_APWN = EC_CAP_MODE;
ECap1Regs.ECCTL2.bit.CONT_ONESHT = EC_CONTINOUS;
ECap1RegS.ECCTL2.bit.SYNCO_SEL = EC_SYNCO DIS;
ECap1Regs.ECCTL2.bit.SYNCO_EN = EC_DISABLE;
Ecap1Regs.ECEINT.bit.CEVT4=1；
ECap1Regs.ECCTL2.bit.TSCTRSTOP = EC_RUN;
}

Uint32 t1,t2,t3;
float period;
float duty,ontime,offtime;
interrupt void Ecap1Int_isr (void)
{
  t1=ECap1Regs.CAP2;
  t2=ECap1Regs.CAP3;
  t3=ECap1Regs.CAP4;

Period=t3-t1; //第一个周期
Frequency=60000000/Period;//计算周期
Ontime = t1; //导通时间
Offtime = t2; //断开时间
if(duty > 0.499)&&(duty < 0.501) Summetry=1;
else Summetry=0;
ECap1Regs.ECCLR.bit.CEVT4 = 1;
ECap1Regs.ECCLR.bit.INT = 1;
PieCtrlRegs.PIEACK.all=PIEACK_ GROUP4 ;
}
```