2020 年春季学期

《DSP 技术与工程应用》课程

课程设计报告

院系：航天学院

班号：1704104

学号：1170400423

姓名：尉前进

2020 年 5 月 25 日

审阅教师：_____　　考核成绩：_____

题目一：

结合学习过的 DSP 基本知识，试论述采用 DSP 为核心器件设计系统，需要考虑综合哪些性能指标、遵循哪些设计原则？（10 分）

一、运算速度：首先我们要确定数字信号处理的算法，算法确定以后其运算量和完成时间也就大体确定了，根据运算量及其时间要求就可以估算 DSP 芯片运算速度的下限。在选择 DSP 芯片时，各个芯片运算速度的衡量标准主要有：

1、MIPS(Millions of Instructions Per Second)，百万条指令/秒，一般 DSP 为 20~100MIPS，使用超长指令字的 TMS320B2XX 为 2400MIPS。必须指出的是这是定点 DSP 芯片运算速度的衡量指标，应注意的是，厂家提供的该指标一般是指峰值指标，因此，系统设计时应留有一定的裕量。

2、MOPS(Millions of Operations Per Second)，每秒执行百万操作。这个指标的问题是什么是一次操作，通常操作包括 CPU 操作外，还包括地址计算、DMA 访问数据传输、I/O 操作等。一般说 MOPS 越高意味着乘积-累加和运算速度越快。MOPS 可以对 DSP 芯片的性能进行综合描述。

3、MFLOPS（Million Floating Point Operations Per Second），百万次浮点操作/秒，这是衡量浮点 DSP 芯片的重要指标。例如 TMS320C31 在主频为 40MHz 时，处理能力为 40MFLOPS，TMS320C6701 在指令周期为 6ns 时，单精度运算可达 1GFLOPS。浮点操作包括浮点乘法、加法、减法、存储等操作。应注意的是，厂家提供的该指标一般是指峰值指标，因此，系统设计时应注意留有一定的裕量。

4、MBPS(Million Bit Per Second)，它是对总线和 I/O 口数据吞吐率的度量，也就是某个总线或 I/O 的带宽。例如对 TMS320C6XXX、200MHz 时钟、32bit 总线时，总线数据吞吐率则为 800Mbyte/s 或 6400MBPS。

5、指令周期，即执行一条指令所需的时间，通常以 ns（纳秒）为单位，如 TMS320LC549-80 在主频为 80MHz 是的指令周期为 12.5ns。MAC 时间，执行一次乘法和加法运算所花费的时间：大多数 DSP 芯片可以在一个指令周期内完成一次 MAC 运算。

6、FFT/FIR 执行时间，运行一个 N 点 FFT 或 N 点 FIR 程序的运算时间。由于 FFT 运算/FIR 运算是数字信号处理的一个典型算法，因此，该指标可以作为衡量芯片性能的综合指标。

二、运算精度：一般情况下，浮点 DSP 芯片的运算精度要高于定点 DSP 芯片的运算精度，但是功耗和价格也随之上升。

三、字长的选择：一般浮点 DSP 芯片都用 32 位的数据字，大多数定点 DSP 芯片是 16 位数据字。

四、存储器等片内硬件资源安排：包括存储器的大小，片内存储器的数量，总线寻址空间等。

五、开发调试工具：完善、方便的的开发工具和相关支持软件是开发大型、复杂 DSP 系统的必备条件，对缩短产品的开发周期有很重要的作用。

六、功耗与电源管理：一般来说个人数字产品、便携设备和户外设备等对功耗有特殊要求，因此这也是一个该考虑的问题。

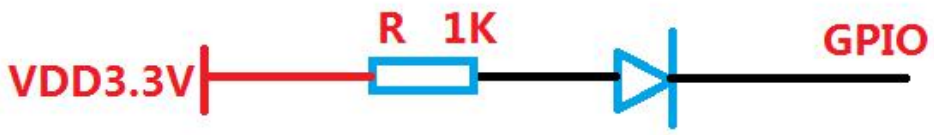
七、价格及厂家的售后服务因素：价格包括 DSP 芯片的价格和开发工具的价格。

八、其他因素：包括 DSP 芯片的封装形式、环境要求、供货周

期、生命周期等。

题目二：

如果采用 DSP 处理器的 GPIO 端口点亮小灯，试画出基本的电路原理图。（10 分）



当 GPIO 端为低电平时，小灯被点亮。

题目三：

编写基本 C 语言程序，把 DSP 的 GPIOA 口设置成输出端口，GPIOB 口设置成输入端口。（10 分）

```
GpioCtrlRegs.GPBMUX2.bit.GPIOA = 0;

    GpioCtrlRegs.GPBDIR.bit.GPIOA=1;

GpioCtrlRegs.GPBMUX2.bit.GPIOB = 0;

  GpioCtrlRegs.GPBDIR.bit.GPIOB=0;
```

题目四：

设计 DSP 处理器 SCI 串口同计算机通信，采用中断方式收发数据，间隔 3 秒发送一次字符"The F2812-UART is fine !"；要求 SCI 配置为波特率 4800，数据长度 8Bit，无极性，2 位停止位；使用 TX 缓冲寄存器空触发 SCI-TX INT 中断，CPU 定时器 0 中断触发第一次传输，试给出相关程序代码。（15 分）

```
#include "DSP281x_Device.h"

// 使用的函数原型声明

void Gpio_select(void);
void InitSystem(void);
void SCI_Init(void);
```

```
interrupt void cpu_timer0_isr(void);
interrupt void SCI_TX_isr(void);

// 全局变量
char message[]={"The F2812-UART is fine !\n\r"};
int index =0;           // 字符串指针

void main(void)
{
    InitSystem();       // 初始化 DSP 内核寄存器

    Gpio_select();      // 配置 GPIO 复用功能寄存器


    InitPieCtrl();      // 调用外设中断扩展初始化单元 PIE-unit ( 代码 : DSP281x_PieCtrl.c)

    InitPieVectTable();             // 初始化 PIE vector 向量表 ( 代码 : DSP281x_PieVect.c )

    // 重新映射 PIE - Timer 0 的中断
    EALLOW;  // 解除寄存器保护
    PieVectTable.TINT0 = &cpu_timer0_isr;
    EDIS;    // 使能寄存器保护

    InitCpuTimers();

   // 配置 CPU-Timer 0 周期 50 ms:
   // 150MHz CPU 频率, 50000 微秒中断周期
    ConfigCpuTimer(&CpuTimer0, 150, 50000);

    // 使能 PIE 内的 TINT0 : Group 1 interrupt 7
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
```

```
// 使能 CPU INT1 （连接到 CPU-Timer 0 中断）
   IER = 1;

   EALLOW; // 解除寄存器保护
   PieVectTable.TXAINT = &SCI_TX_isr;
   EDIS;          // 使能寄存器保护

   // 使能 PIE 内的 SCI_A_TX_INT 中断
   PieCtrlRegs.PIEIER9.bit.INTx2 = 1;

   // 使能 CPU INT 9
   IER |= 0x100;

   // 全局中断使能和更高优先级的实时调试事件
   EINT;    // 全局中断使能 INTM
   ERTM;    // 使能实时调试中断 DBGM

   CpuTimer0Regs.TCR.bit.TSS = 0;  // 启动定时器 0

   SCI_Init();
   while(1)
  {
                  while(CpuTimer0.InterruptCount < 60)   // 等待
50ms * 60
       {
       EALLOW;
       SysCtrlRegs.WDKEY = 0x55;
     SysCtrlRegs.WDKEY = 0xAA;       // 看门狗控制
       EDIS;
       }
       CpuTimer0.InterruptCount = 0;    // 复位计数器
       index = 0;
```

```
            SciaRegs.SCITXBUF= message[index++];
        }
    }


    void Gpio_select(void)
    {
        EALLOW;
        GpioMuxRegs.GPAMUX.all = 0x0;   // 所有 GPIO 端口配置为 I/O
        GpioMuxRegs.GPBMUX.all = 0x0;
        GpioMuxRegs.GPDMUX.all = 0x0;
        GpioMuxRegs.GPFMUX.all = 0x0;
        GpioMuxRegs.GPFMUX.bit.SCIRXDA_GPIOF5 = 1; //配置 SCI-RX
        GpioMuxRegs.GPFMUX.bit.SCITXDA_GPIOF4 = 1; //配置 SCI-TX
        GpioMuxRegs.GPEMUX.all = 0x0;
        GpioMuxRegs.GPGMUX.all = 0x0;

        GpioMuxRegs.GPADIR.all = 0x0;    // GPIO PORT    配置为输入
        GpioMuxRegs.GPBDIR.all = 0x0;
        GpioMuxRegs.GPDDIR.all = 0x0;
        GpioMuxRegs.GPEDIR.all = 0x0;
        GpioMuxRegs.GPFDIR.all = 0x00FF;
        GpioMuxRegs.GPGDIR.all = 0x0;

        GpioMuxRegs.GPAQUAL.all = 0x0;     // 设置所有 GPIO 输入的量化值等于 0
        GpioMuxRegs.GPBQUAL.all = 0x0;
        GpioMuxRegs.GPDQUAL.all = 0x0;
        GpioMuxRegs.GPEQUAL.all = 0x0;
```

```
    EDIS;
}

void InitSystem(void)
{
    EALLOW;
    SysCtrlRegs.WDCR= 0x00AF;       // 配置看门狗
                                    // 0x00E8  禁止看门狗，预定标系数 Prescaler = 1
                                    // 0x00AF  不禁止看门狗，预定标系数 Prescaler = 64
    SysCtrlRegs.SCSR = 0;           // 看门狗产生复位
    SysCtrlRegs.PLLCR.bit.DIV = 10; // 配置处理器锁相环，倍频系数为 5

    SysCtrlRegs.HISPCP.all = 0x1;   // 配置高速外设时钟分频系数： 2
    SysCtrlRegs.LOSPCP.all = 0x2;   // 配置低速外设时钟分频系数： 4

    // 使用的外设时钟时钟设置：
    // 一般不使用的外设的时钟禁止，降低系统功耗
    SysCtrlRegs.PCLKCR.bit.EVAENCLK=0;
    SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
    SysCtrlRegs.PCLKCR.bit.SCIAENCLK=1; // 使能 SCI 模块的时钟
    SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
    SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
    SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
    SysCtrlRegs.PCLKCR.bit.ECANENCLK=0;
    SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
    EDIS;
```

```
}

void SCI_Init(void)
{
   SciaRegs.SCICCR.all =0x13;     // 2bit 停止位       无循环模式
                                                    //   无   极   性   ，
字符长度：8 bits,
                                                    //   异   步   模   式   ，
空闲线协议
    SciaRegs.SCICTL1.all =0x0003;     //   使   能    TX,  RX,   内   部
SCICLK,
                                                    //   禁   止    RX   ERR,
SLEEP, TXWAKE
    SciaRegs.SCIHBAUD = 0x03;   //  波特率：4800（LSPCLK =
37.5MHz）  ;
    SciaRegs.SCILBAUD = 0xd0;
    SciaRegs.SCICTL2.bit.TXINTENA = 1;//  使能 SCI 发送中断

   SciaRegs.SCICTL1.all =0x0023;       //  使 SCI 退出复位
}

interrupt void cpu_timer0_isr(void)
{
     CpuTimer0.InterruptCount++;
      //  每个定时器中断清除一次看门狗计数器

    //  响应中断并允许系统接收更多的中断
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

//=============================================================
============================
// SCI_A  发送中断服务程序
```

```
// 发送字符串  message[]
//=============================================================================
interrupt void SCI_TX_isr(void)
{
   if (index < 26) SciaRegs.SCITXBUF=message[index++];
   // 重新初始化 PIE 为下一次  SCI-A TX 准备接收下一次中断
      PieCtrlRegs.PIEACK.all = 0x0100;      //响应中断
}
```

题目五：

采用查询方式实现题目四中功能，SCI 配置要求相同，使用发送移位寄存器空标志位 TXEMPTY 触发发送数据，软件延时方法控制间隔时间 4 秒，试给出相关程序代码。（15 分）

```
#include "DSP281x_Device.h"

// 使用的函数声明

void Gpio_select(void);
void InitSystem(void);
void SCI_Init(void);

void main(void)
{
      char message[]={"The F2812-UART is fine !\n\r"};
      int index =0;          // 字符指针定义
      long i;

      InitSystem();          // 初始化 DSP 内核寄存器

      Gpio_select();         // 配置 GPIO 复用功能寄存器

      SCI_Init();              // SCI 接口初始化
```

```
        while(1)
        {
            SciaRegs.SCITXBUF=message[index++];
            while ( SciaRegs.SCICTL2.bit.TXEMPTY == 0);
         //状态检测模式：
         //状态检测，等待发送标识为空：TXEMPTY = 0
            EALLOW;
            SysCtrlRegs.WDKEY = 0x55;      // 看门狗控制
            SysCtrlRegs.WDKEY = 0xAA;
            EDIS;
            if (index > 26)
            {
                index =0;
             for(i=0;i<20000000;i++)        // 软件延时，近似 4 秒.
             {
                EALLOW;
                SysCtrlRegs.WDKEY = 0x55;    // 看门狗控制
                SysCtrlRegs.WDKEY = 0xAA;            // 看门狗控
制
                EDIS;
             }
            }
        }
    }


    void Gpio_select(void)
    {
        EALLOW;
        GpioMuxRegs.GPAMUX.all = 0x0;   // 所有  GPIO  端口配
置为 I/O
        GpioMuxRegs.GPBMUX.all = 0x0;
        GpioMuxRegs.GPDMUX.all = 0x0;
```

```
        GpioMuxRegs.GPFMUX.all = 0x0;
        GpioMuxRegs.GPFMUX.bit.SCIRXDA_GPIOF5 = 1; //配置
SCI-RX
        GpioMuxRegs.GPFMUX.bit.SCITXDA_GPIOF4 = 1; //配置
SCI-TX
        GpioMuxRegs.GPEMUX.all = 0x0;
        GpioMuxRegs.GPGMUX.all = 0x0;

        GpioMuxRegs.GPADIR.all = 0x0;    // GPIO PORT     配置为
输入
        GpioMuxRegs.GPBDIR.all = 0x0;
        GpioMuxRegs.GPDDIR.all = 0x0;
        GpioMuxRegs.GPEDIR.all = 0x0;
        GpioMuxRegs.GPFDIR.all = 0x0;
        GpioMuxRegs.GPGDIR.all = 0x0;

        GpioMuxRegs.GPAQUAL.all = 0x0;      //  设置所有  GPIO
输入的量化值等于 0
        GpioMuxRegs.GPBQUAL.all = 0x0;
        GpioMuxRegs.GPDQUAL.all = 0x0;
        GpioMuxRegs.GPEQUAL.all = 0x0;
        EDIS;
    }

    void InitSystem(void)
    {
         EALLOW;
         SysCtrlRegs.WDCR= 0x00AF;       //  配置看门狗
                                        // 0x00E8   禁止看门狗，预定
标系数 Prescaler = 1
                                        // 0x00AF   不禁止看门狗，预
定标系数 Prescaler = 64
         SysCtrlRegs.SCSR = 0;           //  看门狗产生复位
```

```
        SysCtrlRegs.PLLCR.bit.DIV = 10;   // 配置处理器锁相环，倍频系数为 5

        SysCtrlRegs.HISPCP.all = 0x1;     // 配置高速外设时钟分频系数： 2
        SysCtrlRegs.LOSPCP.all = 0x2;     // 配置低速外设时钟分频系数： 4

        // 使用的外设时钟时钟设置：
        // 一般不使用的外设的时钟禁止，降低系统功耗
        SysCtrlRegs.PCLKCR.bit.EVAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIAENCLK=1; // 使能 SCI 模块的时钟
        SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ECANENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
        EDIS;
    }

    void SCI_Init(void)
    {
      SciaRegs.SCICCR.all =0x13;     // 2bit 停止位      无循环模式
                                     //  无 极 性 ，字符长度：8 bits,
                                     //  异 步 模 式 ，空闲线协议
       SciaRegs.SCICTL1.all =0x0003;     // 使 能  TX, RX, 内 部 SCICLK,
                                     //  禁 止  RX ERR,
```

SLEEP, TXWAKE

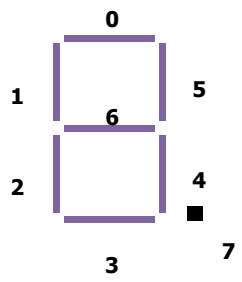
SciaRegs.SCIHBAUD = 0x03; // 波特率：4800 ;

SciaRegs.SCILBAUD = 0xd0;

SciaRegs.SCICTL1.all =0x0023; // 使 SCI 退出复位

}

题目六：



应用 DSP 处理器的 SPI 接口，以查询方式实现低电平点亮 LED 循环显示 16 进制字符“0~F”功能（LED 定义如左图所示），SPI 配置为上升沿无延时发送，数据长度为 8 位，波特率最小值，试设计给出相关程序代码。（20 分）

```
#include "DSP28_Device.h"
Uint16
table[15]={0xc000,0xf900,0xA400,0xB000,0x9900,0x9200,0x8200,0xF8
00,0x8000,0x9000,0x8800,0x8300,0xc600,0xa100,0x8600,0x8e00};
void spi_intial()                    //   SPI 初始化子程序
{
    EALLOW;
    SpiaRegs.SPICCR.all =0x0047;      //   使 SPI 处于复位模式，  下降沿，  8 位数据
        SpiaRegs.SPICTL.all =0x0006;      //   主控模式，一般时钟模式，使能 talk，关闭 SPI 中断
    SpiaRegs.SPIBRR =0x007F;          //   配置波特率
    SpiaRegs.SPICCR.all =SpiaRegs.SPICCR.all|0x00C7;   //   退出复
```

```
位状态
    EALLOW;
        GpioMuxRegs.GPFMUX.all=0x000F;                    //  设置
通用引脚为 SPI 引脚
        EDIS;
   }
void gpio_init()
{
    EALLOW;
    GpioMuxRegs.GPAMUX.bit.TDIRA_GPIOA11=0;    //    GPIOA11
设置为一般 I/O 端口
    GpioMuxRegs.GPADIR.bit.GPIOA11=1;      //   把 GPIOA11 设置
为输出
        EDIS;
        GpioDataRegs.GPADAT.bit.GPIOA11=0;            //   GPIOA11
端口为 74HC595 锁存信号
}
void main(void)
{
    unsigned int k=0;
    InitSysCtrl();                //   系统初始化子程序，该程序包含在
DSP28_SysCtrl.C 中
```

```
    DINT;                           //  关闭总中断
    IER = 0x0000;
    IFR = 0x0000;
    spi_intial();                   //  SPI 初始化子程序
    gpio_init();                    //  GPIO 初始化子程序
    while(1)
    {
        GpioDataRegs.GPADAT.bit.GPIOA11=0;          //  给 LOCK
信号一个低电平
        for(k=0;k<8;k++)
        {
            SpiaRegs.SPITXBUF = table[k+1];         //  给数码管
送数
                while(SpiaRegs.SPISTS.bit.INT_FLAG !=1){}
                SpiaRegs.SPIRXBUF = SpiaRegs.SPIRXBUF;  //  空读
清中断标志
            }
            GpioDataRegs.GPADAT.bit.GPIOA11=1;          //  给
LOCK 信号一个高电平为锁存 74HC595
                for(k=0;k<15;k++){}
    }
}
```