嵌入式编程 C

一、程序

1、配置 GPIO 端口，要求 C 端口高八位输出，低八位输入，定时器 TI，T3 输出有效，采用 SPI 串口输出，配置相应的控制寄存器。
解答：rGP*的魔数设定如下：

```
#include<2410lib.h>
#define rGPBCON (*(volatile unsigned *)0x56000010)
#define rGPCCON (*(volatile unsigned *)0x56000020)
#define rGPHCON (*(volatile unsigned *)0x56000040)
Void GPIO_init()
{
    rGPCCON = 0x00005555;
    rGPBCON = 0x28;
    rGPHCON = 0xa0;
}
```

2、用 C 编程实现利用 S3C2410A A/D 控制器的 AIN7 通道采集一个范围在 0-3V 的电压。
解答：

```
#include<2410lib.h>
#include "sys_init.h"
#define UINT8T  unsigned char
#define UINT16T unsigned short
#define ADC_FREQ 2500000
#define PCLK (202800000/4)
#define rADCCON (*(volatile unsigned *)0x58000000)
#define rADCDATA0 (*(volatile unsigned *)0x5800000c)
volatile UINT8T unPreScaler;
volatile char usEndTest;
void adc_test(void)
{
    Int i,j;
    UINT16T usConData;
    float usEndData;
    unPreScaler = PCLK/ADC_FREQ-1;
    rADCCON = (1<<14)|(unPreScaler<<6)|(7<<3)|(0<<2)|(1<<1);
     // ADC 分频允许+ADC 分频值+AIN7+通常模式+自动启动下一次转换
    usConData = rADCDATA0 & 0x3FF;
    for(j=0;j<20;j++);
```

```
    {
        While(!(rADCCON  & 0x8000));
        usConData = rADCDATA0 &0x3ff;
        usEndData = usConData*3.0000/0x3ff;
`       usEndData = usEndData-(int) usEndData;
        for(i=0;i<4;i++);
        {
            usEndData = usEndData*10;
            usEndData = usEndData-(int) usEndData;
        }
        delay(1000);
    }
}
void main()
{
    sys_init();
    while(1)
    {
        adc_test();
    }
}
```

3、利用 S3C2410X 的 GPE0-3 控制 LED，画出相关电路图，要求：LED 从 0-3 顺序点亮，中间延时时间自己决定。

解答：

```
#include "2410lib.h"
#include "sys_init.h" //包含用于初始化系统时钟的函数
int main()
{      int i;
    sys_init();                    // 初化系统时钟
    // 端口初始化
     rGPECON=0x55；// GPE0-3 作为输出端口
     rGPEUP=0xff；// GPE 石油端口都不加上拉电阻

    while(1)
    {
          rGPEDAT=0xE0 ;//点亮 GPE4（LED1）
          for(i=0;i<100000;i++);              // 延时
          rGPEDAT=0xC0 ;//点亮 GPE4、GPE5（LED1、LED2）
          for(i=0;i<100000;i++);              // 延时
          rGPEDAT=0x80 ;//点亮 GPE4、GPE5、GPE6（LED1、LED2、LED3）
          for(i=0;i<100000;i++);              // 延时
          rGPEDAT=0x00 ;//点亮 GPE4、GPE5、GPE6、GPE7（LED1、LED2、
```

```
LED3、LED4）
        for(i=0;i<100000;i++);              // 延时
        rGPEDAT=0xF0 ;//所有 LED 全灭
        for(i=0;i<100000;i++);              // 延时
    }
}
```

4、通过取出 LedStatus 的特定位进行判断选择对端口 B 的数据寄存器进行特定的清零，控制 LED1 和 LED2 灯的点亮，其中端口 B（rPDATAB）的第 2、3 管脚分别连接 LED1、LED2（注：管脚从第 0 管脚开始编号，低电平点亮，程序不更改其他位）

解答：

（1） 取出 LedStatus 的第 0 位进行判断，如果成立则把端口 B 的数据寄存器的第 2 位清零，其余位状态保留，点亮 LED1：

```
If((Ledstatus & 0x01) = = 0x01)
   rPDATAB= rPDATAB & 0Xfffffffb;
```

（2） 取出 LedStatus 的第 1 进行判断，如果成立则把端口 B 的数据寄存器的第 3 清零，其余位状态保留，点亮 LED2

```
If((Ledstatus & 0x01) = = 0x02)
   rPDATAB= rPDATAB & 0Xfffffff7;
```

【分频量化计算】5、S3C2410 微处理器，假设 PCLK=40MHZ,(20 分)

1.若需要得到 1MHZ 的定时器输入时钟频率，时钟分频值为 8，则预分频值是多少？

解答：

1=（40/（x+1））/8（课本 243 公式） ---> x=4 预分频值是 4

2.使用定时器 0 产生占空比为 60%，周期 1000us 的 PWM 信号，求 TCMPB0、TCNTB0 的初始值。

解答：

PWM 输出时钟频率=fTclk/ TCNTBn

PWM 输出信号占空比（即高电平持续时间所占信号周期的比例）= TCMPBn/ TCNTBn

可得（TCNTB0=1000，TCMPB=600）

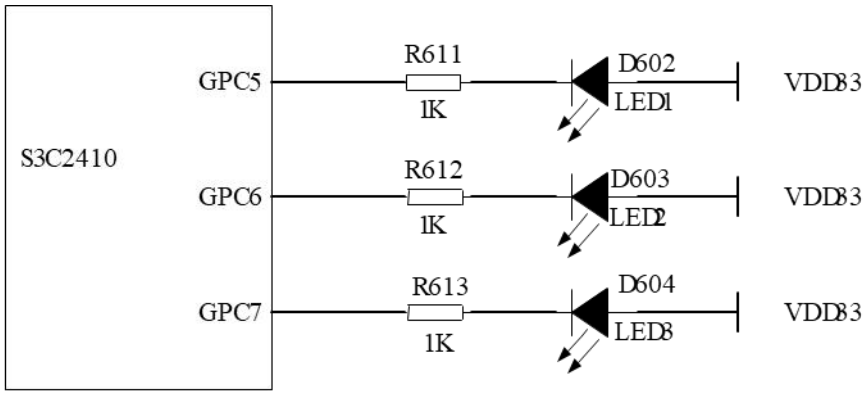
3.用 C 语言编写以上实现 PWM 的程序(相关寄存器地址可以使用头文件包含)

解答：

```
#include "s3c2410.h"
//PWM 定时初始化程序段
void initTimer0(void)
{
    TCFG0 = 0x04; // prescaler = 4
```

```
    TCFG1 = 0x2;// divider = 1/8
    TCON = 0x09;      // 自动加载，清“手动更新”位，启动定时器 0
    TCNTB0 = 0x03E8;   // TCNTB0=1000
    TCMPB0 = 0x0258;   // TCMPB=600
}
```

6、有 3 个 LED 发光二极管，LED1~LED3 由 S3C2410 的 GPC 口 GPC5~GPC7 控制，如图 1 所示。请用 C 语言编写实现 LED1~LED3 间断闪亮的程序。如下图。



解答：

```
#define GPCCON    (*(volatile unsigned *)0x56000020)
#define GPCDAT    (*(volatile unsigned *)0x56000024)
#define GPCUP     (*(volatile unsigned *)0x56000028)
Int LedMain()
{
   void Delay(unsigned int);
   int i;
   GPCUP &=0xffff;          //C 口上拉电阻禁止，未设置扣 2 分
   GPCCON &=0x5454;     // C 口控制字，设置 GPC7-5 为输出，未设置扣 2 分
   while(1)
   {
       GPCDAT = 0x1f;       // GPC7-5 输出 0 ，设置错误扣 2 分
       Delay(70);              //未延时扣 2 分
       GPCDAT = 0xff;      // GPC7-5 输出 1，设置错误扣 2 分
     Delay(70);
   }
     return(0);
```

```
}
viod Delay(unsigned int x)   // 不严格要求间断时间，采用空转的 delay
{
    unsigned int i,j,k;
    for(i=0;i<=x;i++)
       for(j=0;j<=0xff;j++)
           for(k=0;k<=0xff;k++);
}
```

【定时器】7、如果要求严格控制间断时间，需要使用一个定时器 Timer1 控制一只 LED 发光二极管每 1 秒钟改变一次状态(假设 LED 连接到 GPC0)。

解答：

(1)对定时器 1 初始化，并设定定时器的中断时间为 1s，具体代码如下：

```
void Timer1_init（void）
{
    rGPCCON = (rGPCCON | 0x00000001) & 0xfffffffd; //配置输出
    rGPCDAT = rGPCDAT | 0x00000001; //rGPCDAT: Port C data

    rTCFG0 = 255;                    //rTCG0:Timer 0/1 configuration
    rTCFG1 = 0x01<<4;                //rTCG1:Timer 1 configuration
    rTCNTB1 = 48828;                 //rTCNTB1:Timer count buffer1
                                    //在 pclk = 50MHz 下, 1s 的记数值
                                    //rTCNTB1 = 50000000/4/256=48828
    rTCMPB1 ＝ 0x00;                 //rTCMPB1:Timer compare buffer 1
    rTCON ＝(1+11) | (1<<9) | (0<<8);     //禁用定时器 1, 手动加载
    rTCON ＝(1+11) | (0<<9) | (1<<8);     //启动定时器 1, 自动加载
}
```

(2) 为了使 CPU 响应中断，在中断服务子程序执行之前，必须打开 S3C2410 的 CPSR 中的 I 位，以及相应的中断屏蔽寄存器中的位。打开相应的中断屏蔽寄存器中的位是在 Timer1INT Init()函数中实现的，代码如下：

```
void Timer1INT_Init(void)
  {  //定时器接口使能
     if (rINTPND＆BIT_TIMER1)     //若有 INT_TIMER1 中断请求
        rSRCPND | =  BIT_TIMER1;
     pISR_TIMER1=(int) Timer1_ISR; //写入定时器 1 中断服务子程序的
入口地址
     rINTMSK ＆＝ ~ BIT_TIMER1;          //开中断;
  }
```

【UART 串行通信】8、在某种应用中有两台 S3C2410 开发板，需要用其中一台 S3C2410 开发板的 UART0 串口和另外一台 S3C2410 开发板的 UART2 串口实现**全双工**点对点串行通信。假设串行通信采用**脉冲请求的中断方式**、使用收/发 **FIFO** ，试给出你的设计方案及串行通信程序设计的关键步骤。

解答：ref 书 P212

UART 背景知识：

通用异步收发器(Universal Asynchronous Receiver Transmitter，UART)：它用来实现串行数据的传输，相当于我们通常所说的串口。

1.UART 工作原理说明

数据传输概况---数据发送时，CPU 将并行数据写入 UART，UART 按照一定的格式在一根线(TxD)上串行发出；数据接收时，UART 检测另一根线(RxD)上的信号，将串行数据放在缓冲区中供 CPU 读取 UART 获得这些数据。UART 之间以全双工方式传输数据，最精简的连线方式只使用 3 根线(TxD 用于发送数据，RxD 用于接收数据，GND 用于提供双方参考电平)。

数据传输流程---通常数据线处于空闲(1)状态；当要发送数据时，UART 改变 TxD 数据线的状态(1--->0)，并维持 1 位的时间，接收方在检测到这一开始位的时候，再等待 1.5 位时间就开始一位一位地检测数据线的状态以得到数据；UART 一帧中可以有 5/6/7/8 位的数据，发送方一位一位地改变数据线的状态发送数据(首先发送最低位)；如果使用校验功能，UART 在发送完数据位后，还要发送 1 个校验位(分奇校验和偶校验两种校验方法：数据线连同校验位中"1"的个数等于奇/偶数)；最后发送停止位(长度有 1 位/1.5 位/2 位 3 种)，数据线恢复到空闲(1)状态。

2.S3C2410 UART 说明

S3C2410 UART 的特性：3 个独立通道，每个通道可以工作于中断模式或 DMA 模式，UART 由波特率发生器、发送器、接收器、控制逻辑组成。时钟源有两种选择 PCLK 和 UEXTCLK，使用系统时钟时波特率可达 230.4Kbit/s，其波特率可通过编程来控制。

S3C2410 UART 的使用：在使用 UART 前，需要**设置波特率，传输格式，**设置管脚为 UART 功能(比如 UART 通道 0 中，GPH2、GPH3 分别用作 TXD0、RXD0，要使用 UART 通道 0 时，先设置 GPHCON 寄存器将 GPH2、GPH3 引脚功能设置为 TXD0、RXD0)，选择 UART 通道的工作模式为中断模式或者 DMA 模式。设置好之后，往某个寄存器中写入数据即可发送，读取某寄存器即可得到接收的数据，可以通过查询状态寄存器或者设置中断来获知数据是否已经发送完毕，是否已经接收到数据。

3.UART 相关的寄存器

- UBRDIVn 寄存器(n=0~2)---用于设置波特率（以下 bps 表示波特率）。
  算法：UBRDIVn =(int)(PCLK/(bps x 16))-1
  或 UBRDIVn=(int)(UCLK/(bps x 16))-1。
- ULCONn 寄存器---设置传输格式。注意下设为红外模式与正常模式稍有不同。
- UCONn 寄存器---用于选择 UART 的时钟源，设置 UART 中断方式等。
- UFCONn 和 UFSTATn 寄存器---用于设置是否使用 FIFO，设置各 FIFO 的触发阈值，可以通过调协 UFCONn 寄存器来复位各个 FIFO。读取 UFSTATn 寄存器可以知道各个 FIFO 是否已满，其中有多少数据。如果不使用 FIFO 时，可以认为

FIFO 的深度为 1；如果使用 FIFO 时，S3C2410 的 FIFO 深度为 16。
- UMCONn 和 UMSTATn 寄存器---用于流量控制。
- UTRSTATn 寄存器---用来表明数据是否已经发送完毕，是否已经接收到数据。
- USERSTATn 寄存器---用于表示各种错误的发生。
- UTXHn 寄存器---CPU 将数据写入到这个寄存器，UART 即会将它保存到缓冲区中，并自动发送出去。
- URXHn 寄存器---当 UART 接收到数据时，CPU 读取这个寄存器，即可获得数据。

本题要求：uart0 <-> uart2 全双工；脉冲请求的中断方式；使用 FIFO。

```
int UART_BRD = (int)(PCLK/(115200*16))-1; //波特率为 115200
//初始化板 1 的 UART0
void init_uart0(void)
{
    GPHCON  |= 0xa0;    //GPH2、GPH3 用作 TXD1、RXD1
    GPHUP   = 0x0c;     //GPH2、GPH3 内部上拉
    ULCON0  = 0x03;     //8N1(8 个数据位、无较验、1 个停止位)
    UCON0   = 0x05;     //中断请求方式，UART 时钟源为 PCLK
    UFCON0  = 0x08;     //FIFO 大小为 8
    UMCON0  = 0x00;     //不使用流控
    UBRDIV0 = UART_BRD; //波特率为 115200
}

//初始化板 2 的 UART2
void init_uart2(void)
{
    GPHCON  |= 0xa0;    //GPH2、GPH3 用作 TXD2、RXD2
    GPHUP   = 0x0c;     //GPH2、GPH3 内部上拉
    ULCON2  = 0x03;     //8N1(8 个数据位、无较验、1 个停止位)
    UCON2   = 0x05;     //中断请求方式，UART 时钟源为 PCLK
    UFCON2  = 0x0a;     //FIFO 大小为 10
    UMCON2  = 0x00;     //不使用流控
    UBRDIV2 = UART_BRD; //波特率为 115200
}

//发送一个字符
void putc(unsigned char c)
{
    //等待，直到发送缓冲区中的数据已经全部发送出去
    while (!(UTRSTAT0 & TXD0READY));
    //向 UTXH0 寄存器中写入数据，UART 即自动将它发送出去
    UTXH0 = c;
}
```

```
//接收字符
unsigned char getc(void)
{
    //等待，直到接收缓冲区中的有数据
    while (!(UTRSTAT0 & RXD0READY));
    //直接读取 URXH0 寄存器，即可获得接收到的数据
    return URXH0;
}
```