#### 18.5 日历时钟电路与例程

当编程日历时钟时，必须使用频率准确稳定的辅助振荡器，将32.768KHZ的晶体振荡器连接到芯片的SOSCI和SOSCO引脚上，见图18-2所示，不在增加其他的电路。

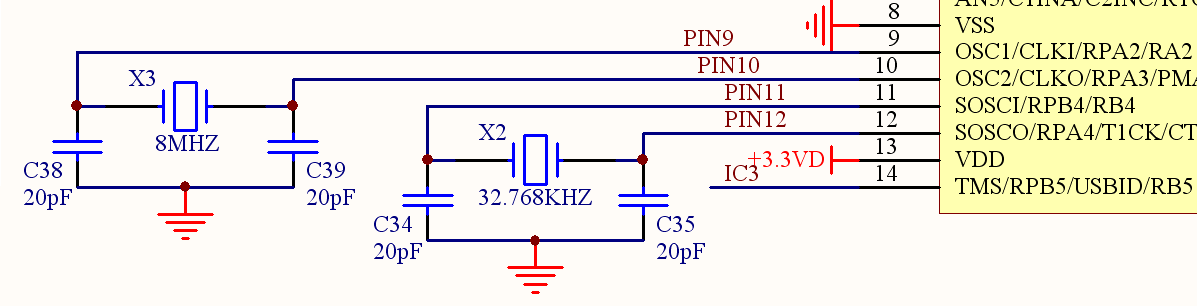


图18-2： 日历时钟晶体振荡器电路

本节描述了在微芯PIC32MX220F032B型芯片上的日历时钟程序示例。示例中利用实时时钟模块，用中断方式产生半秒中断信号，以此启动LED数码管显示RTCC模块的当前时间。

适用范围：本节所描述的代码适用于PIC32MX220F032B型芯片（28 引脚SOIC封装），对于其他型号或封装的芯片，未经测试，不确定其可用性。

表18-1 SPI引脚选择硬件配置表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 功能符号 | 引脚号 | 复用端口选择指定功能所用代码 | 说明 |
| 1 | SCK2 | 26 | 由SPI模块自动选择(SCK2只能选这个引脚) | SPI数据时钟 |
| 2 | SDO2 | 17 | PPSOutput(2, RPB8, SDO2) | SPI数据输出 |
| 3 | SLCK | 18 | PORTSetPinsDigitalOut(IOPORT\_B, BIT\_9) | 外部移位寄存器数据锁存 |

七段数码管显示模块如图18-3所示，采用PIC32MX的SPI口传送数据，并通过74HC595芯片驱动七段数码管进行显示。

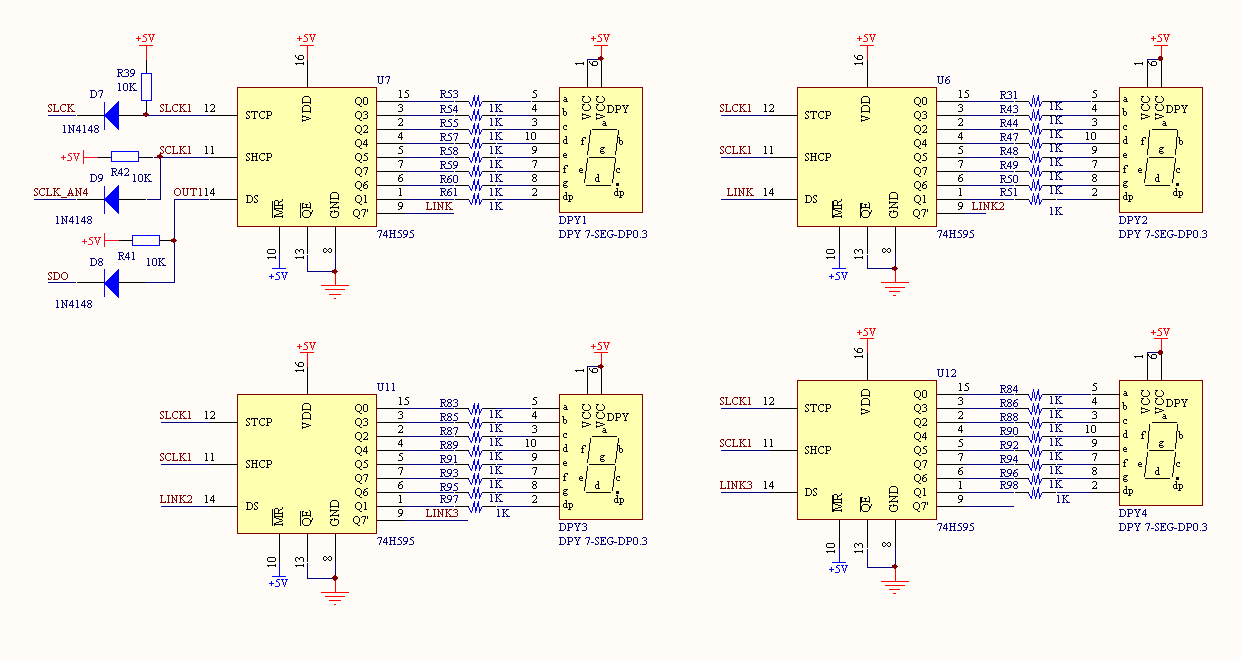


图18-3：3.3V输出电平转换到5V输入电平的转换电路及LED七段数码管驱动电路



图18-4 主函数流程框图

**1、主函数例程（程序流程框图见图18-4所示）**

|  |
| --- |
| int **main**(void)  {  rtccTime tAlrm; // 时间结构体变量  rtccDate dAlrm; // 日期结构体变量  //系统初始化  SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  SpiInitDevice();  //初始化RTCC模块  RtccInit();  //等待辅助振荡器启动及RTCC时钟源稳定  while (RtccGetClkStat() != RTCC\_CLK\_ON);  //设置时间，日期  //第一个变量为时间:用UINT32表示，由高到低的4个字节依次表示：小时，分钟，秒钟，保留  //其中，保留值必须设置为0.下例中0x0D000000表示：12:00:00  //第二个变量未日期:用UINT32表示，由高到低的4个字节依次表示：年，月，日，星期  //下例中0x0D010102表示：2013-01-01,星期二  RtccOpen(0x0D000000, 0x0D010102, 0);  //配置中断模式  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  //使能中断  INTEnableInterrupts();  //设置报警时间  do {  RtccGetTimeDate(&tm, &dt);  } while ((tm.sec & 0xf) > 0x7);  tAlrm.l = tm.l;  dAlrm.l = dt.l;  //允许连续报警  RtccChimeEnable();  //报警次数计数器清零  RtccSetAlarmRptCount(0);  //设置报警间隔：每个0.5s  RtccSetAlarmRpt(RTCC\_RPT\_HALF\_SEC);  //设置报警时间  RtccSetAlarmTimeDate(tAlrm.l, dAlrm.l);  //使能报警  RtccAlarmEnable();  //报警已使能？  if (RtccGetAlarmEnable())  {  //设置RTCC中断、使能中断  INTSetVectorPriority(INT\_RTCC\_VECTOR, INT\_PRIORITY\_LEVEL\_4);  INTSetVectorSubPriority(INT\_RTCC\_VECTOR, INT\_SUB\_PRIORITY\_LEVEL\_1);  INTEnable(INT\_RTCC, INT\_ENABLED);  }  //主循环  while(1)  {  if(led\_flag > 0)  {  led\_flag = 0;  RtccGetTimeDate(&tm, &dt);  Led();  }  }  return 1;  } |



图18-5 RTCC中断函数流程框图

**2、RTCC中断函数例程（程序流程框图见图18-5所示）**

|  |
| --- |
| void \_\_ISR(\_RTCC\_VECTOR, ipl4) **RtccIsr**(void) {  //清中断标志  INTClearFlag(INT\_RTCC);  //翻转秒小数点，用来指示秒钟的变化  point=~point;  //数码管输出计数器：每0.5秒输出一次  led\_flag = 1;  } |

### 附件：代码

|  |
| --- |
| /\*  \* File: rctt.c  \* 实时时钟示例：利用实时时钟模块，实现24小时实时时钟显示  \* 程序起始时间为：2013-01-01 00:00:00  \* 利用SPI驱动的4个8段数码管显示时间（仅显示分钟：秒钟）  \*/  #include <plib.h>  // Configuration Bit settings  // SYSCLK = 48 MHz (8MHz Crystal / FPLLIDIV \* FPLLMUL / FPLLODIV)  // PBCLK = 48 MHz (SYSCLK / FPBDIV)  // Primary Osc w/PLL (XT+,HS+,EC+PLL)  // WDT OFF  #pragma config FPLLMUL = MUL\_24, FPLLIDIV = DIV\_2, FPLLODIV = DIV\_2, FWDTEN = OFF  #pragma config POSCMOD = HS, FNOSC = PRIPLL, FPBDIV = DIV\_1  #define SYS\_FREQ (48000000L)  #pragma config JTAGEN = OFF //JTAG disable  unsigned int led\_flag=1,point=0;  rtccTime tm; // 时间结构体变量  rtccDate dt; // 日期结构体变量  //8段LED数码管字库：0~9 0.~9.FSEt-yno  unsigned char Led\_lib[] = {0x42, 0xf3, 0x86, 0xa2, 0x33, 0x2a, 0x0a, 0xf2, 0x02, 0x22, 0x40, 0xf1, 0x84, 0xa0, 0x31, 0x28, 0x08, 0xf0, 0x00, 0x20, 0x1e, 0x0e, 0x0f, 0xbf, 0x23, 0x9b, 0x8b}; //小LED字库  /\*-------LED段码分布图------  ---0---  | |  7 3  | |  ---6---  | |  5 2  | |  ---4--- 1  ----------------------------\*/  /\*  \* SPI 初始化函数  \*/  void **SpiInitDevice**()  {  // SPI标志位  SpiOpenFlags oFlags = SPI\_OPEN\_MSTEN | SPI\_OPEN\_CKP\_HIGH | SPI\_OPEN\_MODE8 | SPI\_OPEN\_ON;  //输出锁存信号引脚  PORTSetPinsDigitalOut(IOPORT\_B, BIT\_9);  //数据输出引脚  PPSOutput(2, RPB8, SDO2);  //打开SPI模块  SpiChnOpen(2, oFlags, 6);  }  /\*  \* SPI输出多个字符  \*/  void **SpiDoBurst**(unsigned char \*pBuff, unsigned char Len)  {  if (pBuff)  {  unsigned int i;  PORTClearBits(IOPORT\_B, BIT\_9);  for (i = 0; i < Len; i++)  {  SpiChnPutC(2, pBuff[i]);  }  PORTSetBits(IOPORT\_B, BIT\_9);  }  }  /\*  \* LED数码管驱动输出  \*/  void **Led**()  {  static unsigned char ledcnt[4]={0x00, 0x00, 0x00, 0x00};  int fg = tm.min & 0x0f;  if(point) //0.5秒切换一次秒点翻转  fg += 10;  ledcnt[2] = Led\_lib[tm.sec & 0x0f]; //秒：个位  ledcnt[1] = Led\_lib[(tm.sec & 0xf0)>>4]; //秒：十位  ledcnt[0] = Led\_lib[fg]; //分：个位  ledcnt[3] = Led\_lib[(tm.min & 0xf0)>>4]; //分：十位  SpiDoBurst(ledcnt,4);  }  /\*  \* 主函数  \*/  int **main**(void)  {  rtccTime tAlrm; // 时间结构体变量  rtccDate dAlrm; // 日期结构体变量  //系统初始化  SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  SpiInitDevice();  //初始化RTCC模块  RtccInit();  //等待辅助振荡器启动及RTCC时钟源稳定  while (RtccGetClkStat() != RTCC\_CLK\_ON);  //设置时间，日期  //第一个变量为时间:用UINT32表示，由高到低的4个字节依次表示：小时，分钟，秒钟，保留  //其中，保留值必须设置为0.下例中0x0D000000表示：12:00:00  //第二个变量未日期:用UINT32表示，由高到低的4个字节依次表示：年，月，日，星期  //下例中0x0D010102表示：2013-01-01,星期二  RtccOpen(0x0D000000, 0x0D010102, 0);  //配置中断模式  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  //使能中断  INTEnableInterrupts();  //设置报警时间  do {  RtccGetTimeDate(&tm, &dt);  } while ((tm.sec & 0xf) > 0x7);  tAlrm.l = tm.l;  dAlrm.l = dt.l;  //允许连续报警  RtccChimeEnable();  //报警次数计数器清零  RtccSetAlarmRptCount(0);  //设置报警间隔：每个0.5s  RtccSetAlarmRpt(RTCC\_RPT\_HALF\_SEC);  //设置报警时间  RtccSetAlarmTimeDate(tAlrm.l, dAlrm.l);  //使能报警  RtccAlarmEnable();  //报警已使能？  if (RtccGetAlarmEnable())  {  //设置RTCC中断、使能中断  INTSetVectorPriority(INT\_RTCC\_VECTOR, INT\_PRIORITY\_LEVEL\_4);  INTSetVectorSubPriority(INT\_RTCC\_VECTOR, INT\_SUB\_PRIORITY\_LEVEL\_1);  INTEnable(INT\_RTCC, INT\_ENABLED);  }  //主循环  while(1)  {  if(led\_flag > 0)  {  led\_flag = 0;  RtccGetTimeDate(&tm, &dt);  Led();  }  }  return 1;  }  /\*  \* RTCC中断函数  \*/  void \_\_ISR(\_RTCC\_VECTOR, ipl4) **RtccIsr**(void) {  //清中断标志  INTClearFlag(INT\_RTCC);  //翻转秒小数点，用来指示秒钟的变化  point=~point;  //数码管输出计数器：每0.5秒输出一次  led\_flag = 1;  } |