### 14.5 SPI例程

本节描述了在微芯PIC32MX220F032B型芯片上的SPI主控模式的示例代码。代码中，实现了通过SPI主控模式下控制8段LED数码管序列输出一个999.9s的秒表计时器。

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。

表14-1 SPI引脚选择硬件配置表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | SPI功能描述 | 引脚号 | 端口复用选择指定功能 | 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14-2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。

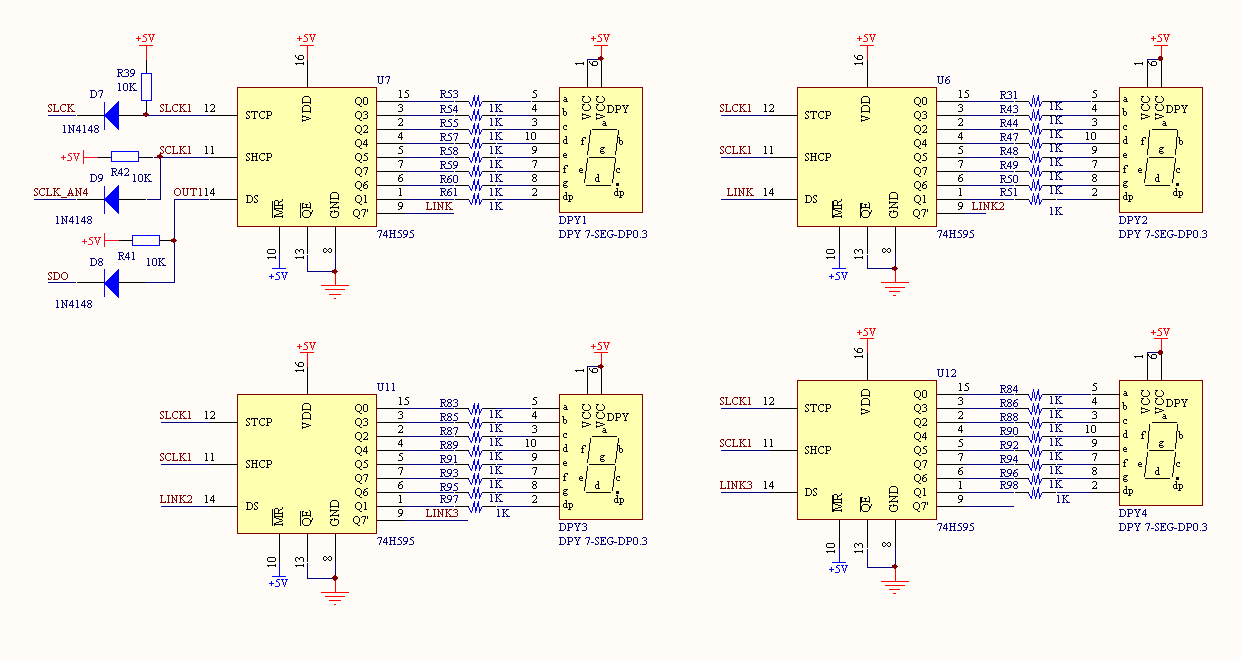


图14-2：3.3V输出电平转换到5V输入电平的转换电路及LED七段数码管驱动电路

74HC595内部有8位移位寄存器和一个存储器，具有高阻关断状态及三态输出状态， 8位串行输入与8位并行输出的特性。移位寄存器和存储寄存器具有独立的时钟信号，数据在移位寄存器时钟信号SHCP的上升沿输入，在存储寄存器时钟信号STCP的上升沿进入到存储寄存器中去，如果两个时钟连在一起，则移位寄存器总是比存储寄存器早一个脉冲。移位寄存器有一个串行移位输入DS、一个串行输出Q7’和一个异步的低电平复位MR，存储寄存器有一个并行8位具备三态的输出端Q0～Q7，当使能OE为低电平时，存储寄存器的数据输出到输出端上，输出端的驱动电流较强能够驱动LED大于10mA以上，需要的单片机接口引脚较少，可以扩展较多个数的LED七段数码管，为静态驱动LED模式，LED的亮度不受扩展数目的影响，是一种较好的LED七段数码管显示驱动方法，见图14-2。

由于PIC32MX的输入/输出端口的输出高电平为电压3.3V，不能直接驱动5V供电的芯片74HC595，采用了的二极管与上拉电阻构成的电平转换匹配电路，电路简单可靠成本低。PIC32MX的SPI口通过匹配电路与74HC595相连，见图14-2，其中，74HC595的SHCP引脚接于SPI串行外设模块时钟引脚，图14-2中标示为SCLK\_AN4，74HC595的STCP引脚接于通用输入输出引脚，图14-2中标示为SLCK，74HC595的DS引脚接于SPI 数据输出，图14-2中标示为SDO。当SDO脚接收到PIC32MX的SPI输出的一个低电平信号时，二极管导通，此时OUT1为一个低电平信号；当SDO脚接收到一个高电平信号时，二极管的OUT1端经过上拉电阻连接到5V，以输出高于4V以上的高电平信号，从而实现电平转换的匹配功能。

当需要使用多个七段数码管显示时，可进行如下处理：MR引脚接高电平，禁止74HC595复位；OE引脚接地，使得存储寄存器的数据能直接输出到输出端；各个74HC595共用SHCP与STCP时钟信号，前一级74HC595的Q7’依次接到下一级74HC595的DS，数据从第一级的DS输入，从本级的Q7’输出到下一级的DS，依次类推，从最后一级的Q7’输出，最后一级的Q7’输出可以不用接任何器件。当数据全部移入所有74HC595的移位寄存器时，所有74HC595的移位寄存器都已经更新后，利用SLCK信号将数据全部移入锁存到存储寄存器，从而实现LED显示信号的锁存与显示。

七段数码管与74HC595之间通过300欧姆电阻连接，该电阻起限流作用，使得数码管流过的电流在5～10mA以内，LED亮度随电流大小和LED是否为高亮型而改变。

图14-3：主函数流程框图 图14-4：数码管显示函数流程框图 图14-5：定时器中断函数流程框图

**1、主函数例程（程序流程框图见图14-3）**

|  |
| --- |
| int **main**(void)  {  SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  INTDisableInterrupts();  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  SpiInitDevice();  Timer1Init();  INTEnableInterrupts();  while(1)  {  if(led\_flag > 0)  {  led\_flag = 0;  Led();  }  }  return 1;  } |

**2、数码管显示函数例程（程序流程框图见图14-4）**

|  |
| --- |
| void **Led**()  {  unsigned char ledBuff[4] = {0x00, 0x00, 0x00, 0x00};  static unsigned char ledcnt[4]={0x00, 0x0A, 0x00, 0x00};  int i;  for (i = 0; i < 4; i++)  ledBuff[i] = Led\_lib[ledcnt[i]];  SpiDoBurst(ledBuff, 4);  ledcnt[2] ++;  if(ledcnt[2] > 9)  {  ledcnt[2] = 0;  ledcnt[1] ++;  if(ledcnt[1] > 19)  {  ledcnt[1] = 10;  ledcnt[0] ++;  if(ledcnt[0] > 9)  {  ledcnt[0] = 0;  ledcnt[3] ++;  if(ledcnt[3] > 9)  {  ledcnt[3] = 0;  }  }  }  }  } |

**3、定时器中断函数例程（程序流程框图见图14-5）**

|  |
| --- |
| void \_\_ISR(\_TIMER\_1\_VECTOR, ipl2) **Timer1Handler**(void)  {  // Clear the interrupt flag  INTClearFlag(INT\_T1);  led\_cnt++;  if(led\_cnt > 100) //0.1s  {  led\_cnt = 0;  led\_flag = 1;  }  } |

### 附件：代码

|  |
| --- |
| /\*  \* File: SPIExample.c  \*/  #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 = OFF, FNOSC = FRCPLL, FPBDIV = DIV\_1  #pragma config FUSBIDIO = OFF //FUSBIDIO为端口控制  #pragma config JTAGEN = OFF //JTAG disable  #pragma config CP = OFF  #pragma config DEBUG = ON  // Period needed for timer 1 to trigger an interrupt every 0.1 second  // (48MHz PBCLK / 1 = 48000000KHz Timer 1 clock)  #define PERIOD 48000 //48000/48000000 = 0.001s = 1ms  #define SYS\_FREQ (48000000L)  unsigned int led\_cnt=0,led\_flag=1;  //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  ----------------------------\*/  void **SpiInitDevice**() {  // 8 bits/char, input data sampled at end of data output time  SpiOpenFlags oFlags = SPI\_OPEN\_MSTEN | SPI\_OPEN\_CKP\_HIGH | SPI\_OPEN\_MODE8 | SPI\_OPEN\_ON;  PORTSetPinsDigitalOut(IOPORT\_B, BIT\_9);  PPSOutput(2, RPB8, SDO2); // Set RB8 pin as output for SDO2  // Open SPI module, use SPI channel 2, use flags set above, Divide Fpb by 6  SpiChnOpen(2, oFlags, 6);  }  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);  }  }  void **Led**()  {  unsigned char ledBuff[4] = {0x00, 0x00, 0x00, 0x00};  static unsigned char ledcnt[4]={0x00, 0x0A, 0x00, 0x00};  int i;  for (i = 0; i < 4; i++)  ledBuff[i] = Led\_lib[ledcnt[i]];  SpiDoBurst(ledBuff, 4);  ledcnt[2] ++;  if(ledcnt[2] > 9)  {  ledcnt[2] = 0;  ledcnt[1] ++;  if(ledcnt[1] > 19)  {  ledcnt[1] = 10;  ledcnt[0] ++;  if(ledcnt[0] > 9)  {  ledcnt[0] = 0;  ledcnt[3] ++;  if(ledcnt[3] > 9)  {  ledcnt[3] = 0;  }  }  }  }  }  void **Timer1Init**()  {  // Timer1@1ms  OpenTimer1(T1\_ON | T1\_SOURCE\_INT | T1\_PS\_1\_1, PERIOD);  // Set up the timer interrupt with a priority of 2  INTEnable(INT\_T1, INT\_ENABLED);  INTSetVectorPriority(INT\_TIMER\_1\_VECTOR, INT\_PRIORITY\_LEVEL\_2);  INTSetVectorSubPriority(INT\_TIMER\_1\_VECTOR, INT\_SUB\_PRIORITY\_LEVEL\_0);  }  void \_\_ISR(\_TIMER\_1\_VECTOR, ipl2) **Timer1Handler**(void)  {  // Clear the interrupt flag  INTClearFlag(INT\_T1);  led\_cnt++;  if(led\_cnt > 100) //0.1s  {  led\_cnt = 0;  led\_flag = 1;  }  }  int **main**(void)  {  SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  INTDisableInterrupts();  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  SpiInitDevice();  Timer1Init();  INTEnableInterrupts();  while(1)  {  if(led\_flag > 0)  {  led\_flag = 0;  Led();  }  }  return 1;  } |