### 16.11 RS-232例程

描述了PIC32MX220F032B芯片上的RS232通信综合示例。集成了SPI通信方式控制LED数码管显示、定时器中断、按钮扫描、UART模块的RS232通信（中断方式）、IO端口输出等众多功能。

运行中需要将两块便携式实验板的RS232通信端口接到一起，按动便携式实验板A的按钮，便携式实验板B的LED数码管数字循环加1，按动便携式实验板B的按钮，便携式实验板A的LED数码管数字循环加1。

表16-1UART相关引脚和SPI引脚以及键盘输入引脚的硬件配置表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |
| 4 | RA0 | 2 | ANSELAbits.ANSA0 = 0 | PORTA.0，连接按钮0 |
| 5 | RA1 | 3 | ANSELAbits.ANSA1 = 0 | PORTA.1，连接按钮1 |
| 6 | RB14 | 25 | ANSELBbits.ANSB14 = 0 | PORTB.14，连接按钮2 |
| 7 | RPB7 | 16 | PPSOutput(1,RPB7,U1TX) | 配置为232发送(UART1.TX) |
| 8 | RPB2 | 6 | PPSInput(3,U1RX,RPB2) | 配置为232接收(UART1.RX) |

**RS-232接口模块简介：**RS-232是一种由[电子](http://baike.so.com/doc/3648997.html)工业联合会制定的用于串行通讯的标准。该标准规定采用一个25个脚的DB-25连接器，对连接器的每个引脚的信号内容加以规定，还对各种信号的电平加以规定。后来IBM的PC机将RS232简化成了DB-9连接器，从而成为事实标准。

虽然RS-232是[计算机](http://baike.baidu.com/view/920814.htm)上常用的[通讯接口](http://baike.baidu.com/view/135196.htm)之一，但其传输距离短（最大传输距离标准值为50英尺，实际上也只能用在50米左右，当采用的通信电缆为150pF/m时，那么它的最大的通信距离为15米）、传输速率较低（在异步传输时，波特率一般小于20Kbps）、抗噪声干扰性弱（接口使用一根输出信号线和一根输入信号线而构成共地的传输形式，这种共地传输容易产生共模干扰）。

RS-232电气特性：在RS-232中任何一条信号线的电压均为负逻辑关系，即：逻辑“1”为-3～-15V；逻辑“0”为+3～+15V。由于TTL等电路采用的是正逻辑，则RS-232和TTL的电路之间需要进行逻辑关系与电平的变换，因此通常会采用具有电荷泵的MAX3232等芯片作为RS-232的收发器，以满足TTL电平与RS-232电平之间的转换。

而工业控制的RS-232接口一般只使用RXD、TXD、GND三条线的三线方式。PIC32MX直接提供了与MAX3232等接口芯片的连接引脚，PIC32MX的SCI接口模块实现串行通信的功能。

MAX3232采用专有低压差发送器输出级，利用双电荷泵在3.0V～5.5V电源供电时能够实现真正的[RS-232](http://baike.baidu.com/view/112004.htm)性能，而器件外部仅需使用4个0.1uF的小容量电荷泵电容，只要输入电压在3.0V～5.5V范围以内，即可提供倍压电荷泵+5.5V和反相电荷泵-5.5V输出电压，电荷泵工作在非连续模式，一旦输出电压低于5.5V，将开启电荷泵。输出电压超过5.5V，即可关闭电荷泵，每个电荷泵需要一个电容器和一个储能电容，产生V+和V-的电压。MAX3232确保在120kbps数据速率，同时保持RS-232输出电平。MAX3232具有二路接收器和二路驱动器，提供1uA关断模式，有效降低功效并延迟便携式产品的电池使用寿命。关断模式下，接收器保持有效状态，对外部设备进行监测，仅消耗1uA电源电流，MAX3232的[引脚](http://baike.baidu.com/view/641241.htm)、封装和功能分别与工业标准MAX242和MAX232兼容，即使工作在高数据速率下，MAX3232仍然能保持RS-232标准要求的正负5.0V最小发送器输出电压。

RS-232接口电路如图16-4所示，PIC32MX通过SCI接口与MAX3232相连来实现与外设的RS-232通信。PIC32MX的SCI发送信号端接到MAX3232其中一路接收器接入端，送至PIC32MX SCI接收端，从而实现信号的双向传递。

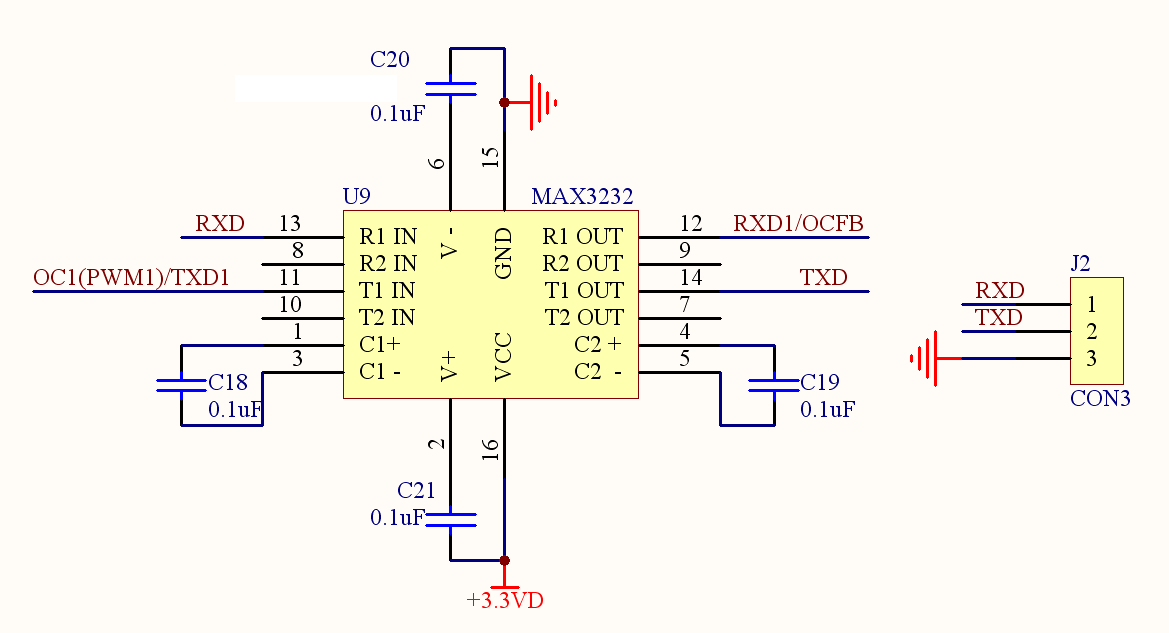


图16-4：RS-232接口电路图



图16-5：主函数流程框图

**1、主函数例程（程序流程框图见16-5所示）**

|  |
| --- |
| int **main**(void)  {  int task=0;  //系统时钟初始化  SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  //禁止中断、配置中断模式  INTDisableInterrupts();  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  //初始化各个模块  UARTinit();  SpiInitDevice();  BtnInit();  Timer1Init();  //允许中断  INTEnableInterrupts();  //主循环  while(1)  {  switch(task)  {  case 0:  if(led\_flag > 0)  {  led\_flag = 0;  Led();  }  break;  case 1:  if(btn\_flag > 0)  {  btn\_flag = 0;  Button();  }  default:  break;  }  task ++;  if(task > 1) task = 0;  }  } |



图16-6：数码管显示函数流程框图

**2、数码管显示函数例程（程序流程框图见16-6所示）**

|  |
| --- |
| void **Led**()  {  static unsigned char ledBuff[4] = {0x00, 0x00, 0x00, 0x00};  static int led = 0,ledt=0;  int i;  SpiDoBurst(ledBuff, 4);  ledt ++;  if(ledt > 9)  {  ledt = 0;  led++;  if (led > 9) led = 0;  }  for (i = 0; i < 3; i++)  ledBuff[i] = Led\_lib[BtnCnt\_t[i]];  ledBuff[3] = Led\_lib[led];  } |



图16-7：按键扫描函数流程框图

**3、按键扫描函数例程（程序流程框图见16-7所示）**

|  |
| --- |
| void **Button**(void) {  static int btn1=0,btn2=0,btn3=0;  if(PORTAbits.RA0 == 0)  {  btn1 ++;  if(btn1 == BTN\_DELAY)  {  PutCharacter(0xA0);  }  }  else  btn1 = 0;  if(PORTAbits.RA1 == 0)  {  btn2++;  if(btn2 == BTN\_DELAY)  {  PutCharacter(0xA1);  }  }  else  btn2 = 0;  if(PORTBbits.RB14 == 0)  {  btn3 ++;  if(btn3 == BTN\_DELAY)  {  PutCharacter(0xA2);  }  }  else  btn3 = 0;  } |



图16-8：定时器中断函数流程框图

**4、定时器中断函数例程（程序流程框图见16-8所示）**

|  |
| --- |
| 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;  }  btn\_cnt++;  if(btn\_cnt > 5) //5ms  {  btn\_cnt = 0;  btn\_flag = 1;  }  } |



图16-9：UART中断函数流程框图

**5、UART中断函数例程（程序流程框图见16-9所示）**

|  |
| --- |
| void \_\_ISR(\_UART\_1\_VECTOR, ipl2) IntUart1Handler(void)  {  // Is this an RX interrupt?  if(INTGetFlag(INT\_SOURCE\_UART\_RX(UART\_MODULE\_ID)))  {  int i;  BYTE t;  t = UARTGetDataByte(UART\_MODULE\_ID);  switch(t)  {  case 0xA0:  i=0;  break;  case 0xA1:  i=1;  break;  case 0xA2:  i=2;  break;  default:  i=0xff;  break;  }  if(i < 0xff)  {  BtnCnt\_t[i]++;  if(BtnCnt\_t[i] > 9)  BtnCnt\_t[i] = 0;  }  // Clear the RX interrupt Flag  INTClearFlag(INT\_SOURCE\_UART\_RX(UART\_MODULE\_ID));  }  // We don't care about TX interrupt  if (INTGetFlag(INT\_SOURCE\_UART\_TX(UART\_MODULE\_ID)))  {  INTClearFlag(INT\_SOURCE\_UART\_TX(UART\_MODULE\_ID));  }  } |

### 附件：代码

|  |
| --- |
| /\*  \* File: RS232Example.c  \*/  #include <plib.h>  // 时钟配置:使用内部振荡器，禁用辅助振荡器，禁用看门狗，总线始终倍频到48MHz  // SYSCLK = 48 MHz (8MHz Crystal / FPLLIDIV \* FPLLMUL / FPLLODIV)  // PBCLK = 48 MHz (SYSCLK / FPBDIV)  // Primary Osc w/PLL (XT+,HS+,EC+PLL)  // WDT OFF  // Disable SOSC  #pragma config FPLLMUL = MUL\_24, FPLLIDIV = DIV\_2, FPLLODIV = DIV\_2, FWDTEN = OFF  #pragma config POSCMOD = OFF, FNOSC = FRCPLL, FPBDIV = DIV\_1, FSOSCEN = OFF  #define SYS\_FREQ (48000000L)  // 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  // 关闭JTAG功能，将相关端口释放给IO口控制，禁用USBIDIO，将对应端口释放给IO控制  #pragma config JTAGEN = OFF,FUSBIDIO = OFF  // UART1宏定义  #define GetPeripheralClock() (SYS\_FREQ/(1 << OSCCONbits.PBDIV))  #define UART\_MODULE\_ID UART1 // 选择模块1  #define DESIRED\_BAUDRATE (1200) // 波特率  //按钮扫描延时：按钮状态持续BTN\_DELAY个定时周期，则触发一次按钮动作  #define BTN\_DELAY 2 //2\*1=2ms  //全局变量定义  unsigned int led\_cnt=0,led\_flag=1;  unsigned int btn\_cnt=0,BtnCnt\_t[]={0,0,0},btn\_flag=0;  //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}; /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* SPI初始化  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  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);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 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数码管显示数据装载和输出  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void **Led**()  {  static unsigned char ledBuff[4] = {0x00, 0x00, 0x00, 0x00};  static int led = 0,ledt=0;  int i;  SpiDoBurst(ledBuff, 4);  ledt ++;  if(ledt > 9)  {  ledt = 0;  led++;  if (led > 9) led = 0;  }  for (i = 0; i < 3; i++)  ledBuff[i] = Led\_lib[BtnCnt\_t[i]];  ledBuff[3] = Led\_lib[led];  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Timer1初始化  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  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);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Timer1中断程序  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  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;  }  btn\_cnt++;  if(btn\_cnt > 5) //5ms  {  btn\_cnt = 0;  btn\_flag = 1;  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 按钮端口初始化  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void **BtnInit**()  {  ANSELAbits.ANSA0 = 0;  ANSELAbits.ANSA1 = 0;  ANSELBbits.ANSB14 = 0;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 按钮扫描程序：按钮按下时，向UART输出一个指定字符  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void **Button**(void) {  static int btn1=0,btn2=0,btn3=0;  if(PORTAbits.RA0 == 0)  {  btn1 ++;  if(btn1 == BTN\_DELAY)  {  PutCharacter(0xA0);  }  }  else  btn1 = 0;  if(PORTAbits.RA1 == 0)  {  btn2++;  if(btn2 == BTN\_DELAY)  {  PutCharacter(0xA1);  }  }  else  btn2 = 0;  if(PORTBbits.RB14 == 0)  {  btn3 ++;  if(btn3 == BTN\_DELAY)  {  PutCharacter(0xA2);  }  }  else  btn3 = 0;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* UART输出一个字符  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void **PutCharacter**(const char character)  {  while(!UARTTransmitterIsReady(UART\_MODULE\_ID))  ;  UARTSendDataByte(UART\_MODULE\_ID, character);  while(!UARTTransmissionHasCompleted(UART\_MODULE\_ID))  ;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* UART1中断服务程序  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void \_\_ISR(\_UART\_1\_VECTOR, ipl2) IntUart1Handler(void)  {  // Is this an RX interrupt?  if(INTGetFlag(INT\_SOURCE\_UART\_RX(UART\_MODULE\_ID)))  {  int i;  BYTE t;  t = UARTGetDataByte(UART\_MODULE\_ID);  switch(t)  {  case 0xA0:  i=0;  break;  case 0xA1:  i=1;  break;  case 0xA2:  i=2;  break;  default:  i=0xff;  break;  }  if(i < 0xff)  {  BtnCnt\_t[i]++;  if(BtnCnt\_t[i] > 9)  BtnCnt\_t[i] = 0;  }  // Clear the RX interrupt Flag  INTClearFlag(INT\_SOURCE\_UART\_RX(UART\_MODULE\_ID));  }  // We don't care about TX interrupt  if (INTGetFlag(INT\_SOURCE\_UART\_TX(UART\_MODULE\_ID)))  {  INTClearFlag(INT\_SOURCE\_UART\_TX(UART\_MODULE\_ID));  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* UART初始化  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void **UARTinit**()  {  //关联引脚  PPSInput(3,U1RX,RPA2); // Assign RPA2 as input pin for U1RX  PPSOutput(1,RPB3,U1TX); // Set RB3 pin as output for U1TX  //UART1模块初始化：配置为串口通信、8位数据、1位停止、无校验、仅用TX和RX引脚...等  UARTConfigure(UART\_MODULE\_ID, UART\_ENABLE\_PINS\_TX\_RX\_ONLY);  UARTSetFifoMode(UART\_MODULE\_ID, UART\_INTERRUPT\_ON\_TX\_NOT\_FULL | UART\_INTERRUPT\_ON\_RX\_NOT\_EMPTY);  UARTSetLineControl(UART\_MODULE\_ID, UART\_DATA\_SIZE\_8\_BITS | UART\_PARITY\_NONE | UART\_STOP\_BITS\_1);  UARTSetDataRate(UART\_MODULE\_ID, GetPeripheralClock(), DESIRED\_BAUDRATE);  UARTEnable(UART\_MODULE\_ID, UART\_ENABLE\_FLAGS(UART\_PERIPHERAL | UART\_RX | UART\_TX));  //UART1中断配置  INTEnable(INT\_SOURCE\_UART\_RX(UART\_MODULE\_ID), INT\_ENABLED);  INTSetVectorPriority(INT\_VECTOR\_UART(UART\_MODULE\_ID), INT\_PRIORITY\_LEVEL\_2);  INTSetVectorSubPriority(INT\_VECTOR\_UART(UART\_MODULE\_ID), INT\_SUB\_PRIORITY\_LEVEL\_0);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 主程序  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int **main**(void)  {  int task=0;  //系统时钟初始化  SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  //禁止中断、配置中断模式  INTDisableInterrupts();  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  //初始化各个模块  UARTinit();  SpiInitDevice();  BtnInit();  Timer1Init();  //允许中断  INTEnableInterrupts();  //主循环  while(1)  {  switch(task)  {  case 0:  if(led\_flag > 0)  {  led\_flag = 0;  Led();  }  break;  case 1:  if(btn\_flag > 0)  {  btn\_flag = 0;  Button();  }  default:  break;  }  task ++;  if(task > 1) task = 0;  }  } |