## 目录

**[示例一 中断程序编程示例](#_Toc19029)** [3](#_Toc19029)

[功能描述及适用范围 3](#_Toc15124)

[程序算法框图 3](#_Toc19760)

[各部分代码 3](#_Toc7553)

**[示例二 开关量输入按键例程](#_Toc15762)** [4](#_Toc15762)

[功能描述及适用范围 4](#_Toc31622)

[硬件配置 5](#_Toc13216)

[程序算法框图 5](#_Toc4126)

[各部分代码及算法流程框图 6](#_Toc17272)

[附件：代码 10](#_Toc6795)

**[示例三 开关量输出LED灯显示例程](#_Toc24672)** [14](#_Toc24672)

[功能描述及适用范围 14](#_Toc22695)

[硬件配置 14](#_Toc20910)

[程序算法框图 15](#_Toc17265)

[各部分代码及算法流程框图 16](#_Toc4181)

[附件：代码 20](#_Toc22802)

**[示例四 定时器定时应用例程](#_Toc28366)** [22](#_Toc28366)

[功能描述及适用范围 22](#_Toc9009)

[硬件配置 22](#_Toc5359)

[程序算法框图 23](#_Toc1007)

[各部分代码及算法流程框图 24](#_Toc22386)

[附件：代码 28](#_Toc29648)

**[示例五 PWM输出方波的例程](#_Toc12469)** [33](#_Toc12469)

[功能描述及适用范围 33](#_Toc32762)

[硬件配置 33](#_Toc3196)

[程序算法框图 33](#_Toc4982)

[各部分代码 34](#_Toc26297)

[附件：代码 37](#_Toc11085)

**[示例六 由PWM输出构成 DA模拟量输出和将其采样的AD例程](#_Toc32170)** [39](#_Toc32170)

[功能描述及适用范围 39](#_Toc21689)

[硬件配置 39](#_Toc6602)

[程序算法框图 41](#_Toc3529)

[各部分代码及算法流程框图 41](#_Toc17002)

[附件：代码 45](#_Toc9246)

**[示例七 SPI例程](#_Toc11242)** [48](#_Toc11242)

[功能描述及适用范围 48](#_Toc19127)

[硬件配置 48](#_Toc21743)

[74HC595简介及用法 49](#_Toc9949)

[程序算法框图 49](#_Toc12803)

[各部分代码 50](#_Toc21935)

[附件：代码 53](#_Toc900)

**[示例八 RS-232例程](#_Toc14053)** [55](#_Toc14053)

[功能描述及适用范围 55](#_Toc4053)

[硬件配置 56](#_Toc28345)

[RS-232接口模块简介 56](#_Toc3104)

[程序算法框图 57](#_Toc24608)

[各部分代码及算法流程框图 58](#_Toc4147)

[附件：代码 64](#_Toc10675)

**[示例九 日历时钟电路与例程](#_Toc5691)** [70](#_Toc5691)

[功能描述及适用范围 70](#_Toc17144)

[硬件配置 70](#_Toc19823)

[程序算法框图 71](#_Toc1267)

[各部分代码及算法流程框图 72](#_Toc32481)

[附件：代码 74](#_Toc26)

**[示例十 A/D例程](#_Toc32275)** [77](#_Toc32275)

[功能描述及适用范围 77](#_Toc21293)

[硬件配置 77](#_Toc11013)

[程序算法框图 78](#_Toc4474)

[各部分代码及算法流程框图 79](#_Toc31185)

[附件：代码 81](#_Toc31894)

**[示例十一 USB电路连接图和编程示例](#_Toc21527)** [83](#_Toc21527)

[功能描述及适用范围 84](#_Toc13463)

[硬件配置 84](#_Toc2749)

[程序算法框图 85](#_Toc23369)

[各部分代码及算法流程框图 85](#_Toc729)

[附件1：代码 88](#_Toc18944)

[附件2：示例工程的使用方法 92](#_Toc24250)

[附件3：通信协议 95](#_Toc14058)

**[示例十二 Complex例程](#_Toc11374)** [96](#_Toc11374)

[功能描述及适用范围 96](#_Toc19991)

[硬件配置 97](#_Toc11570)

[七段数码管显示模块 97](#_Toc9098)

[开关量输入按键电路电位变化中断编程 97](#_Toc9651)

[RS-232通信模块 98](#_Toc20245)

[程序算法框图 98](#_Toc25867)

[各部分代码及算法流程框图 99](#_Toc18379)

[附件：代码 106](#_Toc24787)

# 示例一 中断程序编程示例

**功能描述及适用范围**

本节描述了在微芯PIC32MX220F032B型芯片上的中断函数示例。本示例以Timer1的中断函数为引，概述了中断函数的初始化、配置、进入等操作。有关中断函数的更多应用示例，请参照其他章节的示例代码。

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。

**程序算法框图**

图9-2：使用中断的主函数流程框图 图9-3：Timer1初始化函数流程框图 图9-4：Timer1中断函数流程框图

### 各部分代码

**1、使用中断的主函数例程（程序流程框图见图9-2所示）**

|  |
| --- |
| int **main**(void)  {  //申明变量、系统时钟初始化等  …  …  //禁止中断（全局）  INTDisableInterrupts();  //配置中断模式  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  //在这里调用具体的中断初始化  Timer1Init();  … //其他中断初始化函数:这些函数必须放在中断禁止/允许里面  … //其他初始化函数：这些函数可放在中断禁止/允许的外面  //允许中断（全局）  INTEnableInterrupts();  //主循环  while(1)  {  … //主循环：处理主要逻辑  } |

**2、Timer1初始化函数例程（程序流程框图见图9-3所示）**

|  |
| --- |
| void **Timer1Init**()  {  // 打开Timer1  OpenTimer1(T1\_ON | T1\_SOURCE\_INT | T1\_PS\_1\_1, PERIOD);  // 允许Timer1中断  INTEnable(INT\_T1, INT\_ENABLED);  // 设置Timer1中断优先级：包括两个  INTSetVectorPriority(INT\_TIMER\_1\_VECTOR, INT\_PRIORITY\_LEVEL\_2);  INTSetVectorSubPriority(INT\_TIMER\_1\_VECTOR, INT\_SUB\_PRIORITY\_LEVEL\_0);  } |

**3、Timer1中断函数例程（程序流程框图见图9-4所示）**

|  |
| --- |
| void \_\_ISR(\_TIMER\_1\_VECTOR, ipl2) **Timer1Handler**(void)  {  // 清除中断标志  INTClearFlag(INT\_T1);  // 以下写其他的用户操作程序  …  } |

# 示例二 开关量输入按键例程

**功能描述及适用范围**

本节描述了在微芯PIC32MX220F032B型芯片上的数字IO输入示例。通过按键连接IO输入，用SPI主控输出8段数码管（LED）显示3个按键计数值，同时用一个LED显示秒定时计数。

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。

**硬件配置**

表10-3输入引脚硬件配置表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 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 |
| 4 | RA0 | 2 | ANSELAbits.ANSA0 = 0 | PORTA.0，按键K1 |
| 5 | RA1 | 7 | ANSELBbits.ANSB3 = 0 | PORTB.3，按键K4 |

开关量输入按键电路电位变化中断编程，在硬件系统中，通过按键开通或闭合来产生高或低电平，从而实现控制信号的键入。如图10-4所示，采用4.7千欧的电阻与SW-PB限位开关串联的方案设计按键模块，当某按键断开的时候，其相应的输出信号（K1和K4）呈现出高电压；当按键闭合时，输出信号则呈现出低电压。其输出信号直接与PIC32MX输入输出端口相连，提供相应的控制信息。

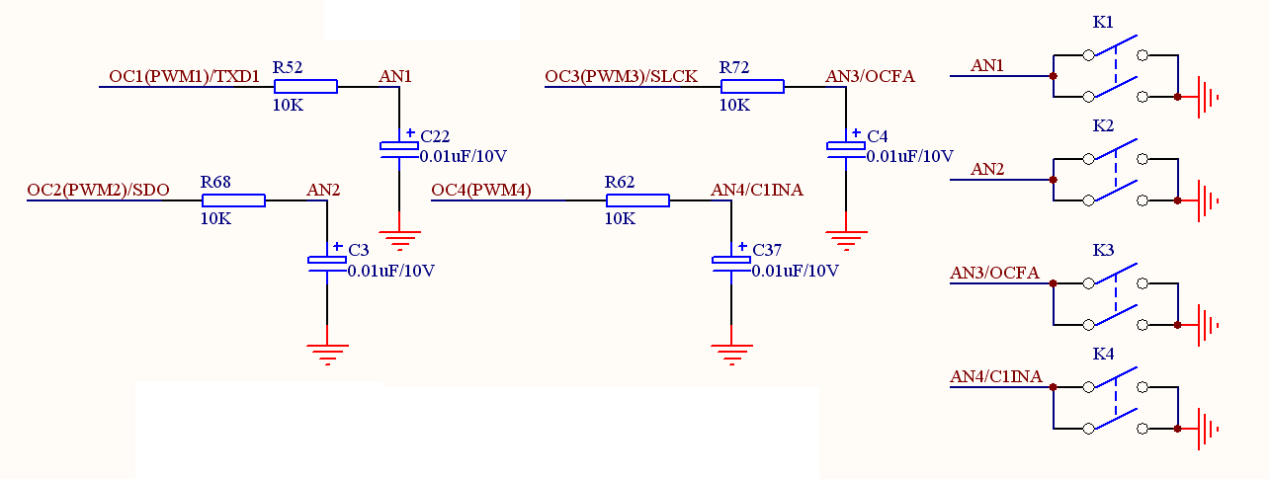


图10-4：按键电路

七段数码管显示模块如图10-5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。

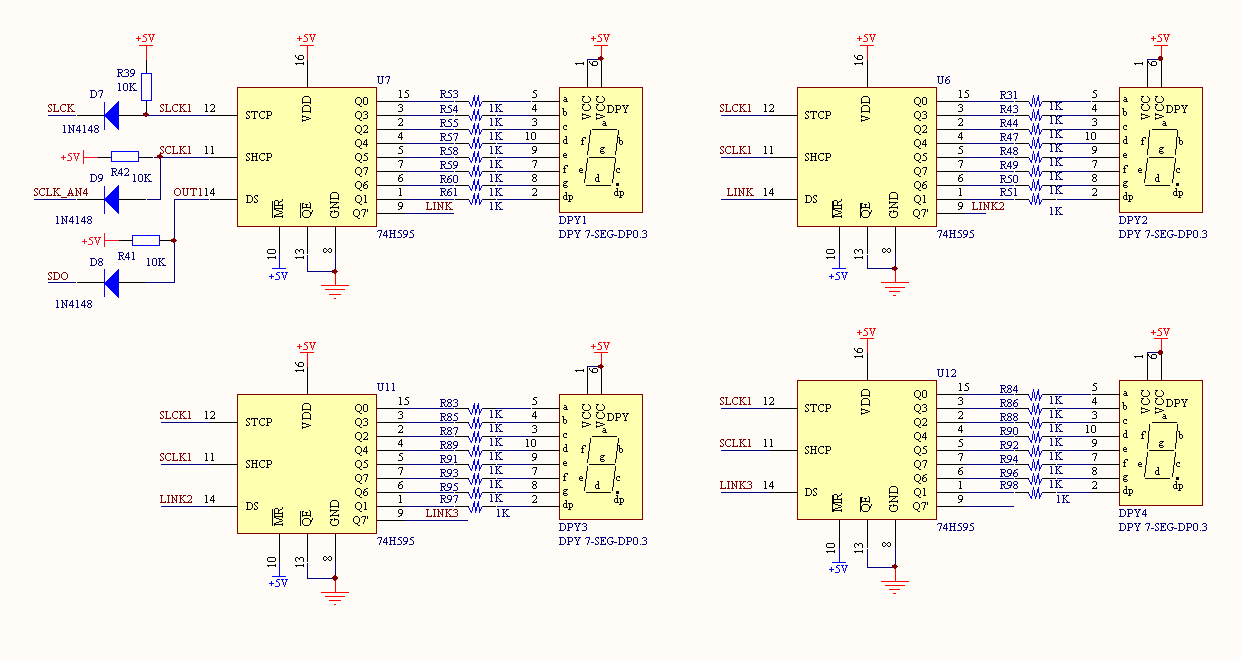


图10-5：3.3V输出电平转换到5V输入电平的转换电路及LED七段数码管驱动电路

**程序算法框图**



图10-6：主函数流程框图

### 各部分代码及算法流程框图

**1、主函数例程（程序流程框图见图10-6所示）**

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



图10-7：数码管显示函数流程框图

**2、数码管显示函数例程（程序流程框图见图10-7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[i]];  ledBuff[3] = Led\_lib[led];  } |



图10-8：按键扫描函数流程框图

**3、按键扫描函数例程（程序流程框图见图10-8所示）**

|  |
| --- |
| void **Button**(void)  {  static int btn1=0,btn2=0,btn3=0;  if (PORTAbits.RA0 == 0)  {  btn1 ++;  if(btn1 == BTN\_DELAY)  {  BtnCnt[0]++;  if (BtnCnt[0] > 9)  BtnCnt[0] = 0;  }  }  else  btn1 = 0;  if (PORTBbits.RB3 == 0)  {  btn2++;  if(btn2 == BTN\_DELAY)  {  BtnCnt[1]++;  if (BtnCnt[1] > 9)  BtnCnt[1] = 0;  }  }  else  btn2 = 0;  } |



图10-9定时器中断函数流程框图

**4、定时器中断函数例程（程序流程框图见图10-9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;  }  } |

### 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 = HS, FNOSC = FRCPLL, FPBDIV = DIV\_1,FSOSCEN = OFF  #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 BTN\_DELAY 5 //1\*5=5ms  #define SYS\_FREQ (48000000L)  unsigned int led\_cnt=0,btn\_cnt=0,led\_flag=1,BtnCnt[]={0,0,0},btn\_flag=0;  //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**()  {  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[i]];  ledBuff[3] = Led\_lib[led];  }  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;  }  btn\_cnt++;  if(btn\_cnt > 5) //5ms  {  btn\_cnt = 0;  btn\_flag = 1;  }  }  void **BtnInit**()  {  ANSELAbits.ANSA0 = 0;  ANSELBbits.ANSB3 = 0;  }  void **Button**(void)  {  static int btn1=0,btn2=0,btn3=0;  if (PORTAbits.RA0 == 0)  {  btn1 ++;  if(btn1 == BTN\_DELAY)  {  BtnCnt[0]++;  if (BtnCnt[0] > 9)  BtnCnt[0] = 0;  }  }  else  btn1 = 0;  if (PORTBbits.RB3 == 0)  {  btn2++;  if(btn2 == BTN\_DELAY)  {  BtnCnt[1]++;  if (BtnCnt[1] > 9)  BtnCnt[1] = 0;  }  }  else  btn2 = 0;  }  int **main**(void)  {  int task=0;  SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  INTDisableInterrupts();  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  SpiInitDevice();  BtnInit();  Timer1Init();  INTEnableInterrupts();  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;  }  return 1;  } |

# 示例三 开关量输出LED灯显示例程

**功能描述及适用范围**

本节描述了在微芯PIC32MX220F032B型芯片上的数字引脚控制输出示例代码。代码中，实现了对5个led灯的跑马灯控制。

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。

### 硬件配置

表10-4输出引脚硬件配置表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | LED编号（PCB上的印刷号） | IO口 | 引脚号 | 端口复用选择指定功能 |
| 1 | D10 | PORTB\_7 | 16 | PORTSetPinsDigitalOut(IOPORT\_B, BIT\_7) |
| 2 | D12 | PORTB\_8 | 17 | PORTSetPinsDigitalOut(IOPORT\_B, BIT\_8) |
| 3 | D13 | PORTB\_9 | 18 | PORTSetPinsDigitalOut(IOPORT\_B, BIT\_9) |
| 4 | D4 | PORTB\_13 | 24 | PORTSetPinsDigitalOut(IOPORT\_B, BIT\_13) |

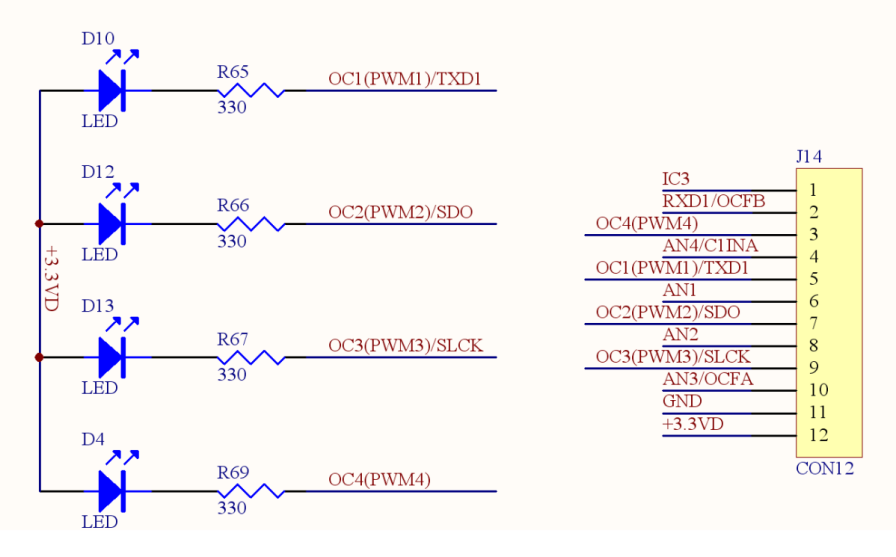


图10-10：开关量输出LED灯显示电路图

**开关量输出LED显示模块：**

LED发光二极管通常可以用来指示系统的工作状态，例如LED点亮可以表示电源模块正常工作。此外，在PIC32MX的I/O引脚外接LED显示，可以方便的显示各I/O端口的控制状态。采用端口下拉的显示电路，当芯片输出引脚为低电平时点亮发光二极管LED，反之，则熄灭。如图10-4所示，4个LED分别接到PIC32MX的4个I/O引脚上。

LED显示需要考虑所采用的发光二极管的工作参数及PIC32MX芯片引脚的驱动电流，芯片输出低电平时，允许外部器件向芯片引脚内灌入电流，这个电流称为“灌电流”；单片机输出高电平时，则允许外部器件从芯片的引脚拉出电流，这个电流称为“拉电流”。例如，采用红色贴片LED，其正常工作状态下的压降为1.4V左右，工作电流为3-8mA。由于工作电压为3.3V，设工作电流为5mA时，接入电阻最大应为欧姆，当工作电流为8mA时，接入电阻最小应为欧姆，可选限流电阻为330欧姆。因此，为了保护发光二极管避免电流过大损坏，同时考虑芯片引脚灌电流在承受范围内，可在LED支路串入一个330欧姆的电阻。

**程序算法框图**



图10-11： 主函数流程框图

### 各部分代码及算法流程框图

**1、主函数例程（程序流程框图见10-11所示）**

|  |
| --- |
| int **main**(void)  {  int task=0;  SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  INTDisableInterrupts();  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  LightInit();  Timer1Init();  INTEnableInterrupts();  while(1)  {  switch(task)  {  case 0:  if(light\_flag > 0)  {  light\_flag = 0;  Light();  }  break;  default:  break;  }  task ++;  if(task > 0) task = 0;  }  return 1;  } |



图10-12：跑马灯函数流程框图

**2、跑马灯函数例程（程序流程框图见10-12所示）**

|  |
| --- |
| void **Light**()  {  static int light = 0;  //将相关IO口置高电平  PORTSetBits(IOPORT\_B, BIT\_7 | BIT\_8 | BIT\_9 | BIT\_13);  switch (light) { //light status  case 0: //将PORTB.9置低电平  PORTClearBits(IOPORT\_B, BIT\_9);  break;  case 1: //将PORTB.8置低电平  PORTClearBits(IOPORT\_B, BIT\_8);  break;  case 2: //将PORTB.7置低电平  PORTClearBits(IOPORT\_B, BIT\_7);  break;  case 3: //将PORTB.13置低电平  PORTClearBits(IOPORT\_B, BIT\_13);  break;  case 4: //将PORTB.7置低电平  PORTClearBits(IOPORT\_B, BIT\_7);  break;  case 5: //将PORTB.8置低电平  PORTClearBits(IOPORT\_B, BIT\_8);  break;  default:  break;  }  light++;  if (light > 5) light = 0;  } |



图10-13：定时器中断函数流程框图

**3、定时器中断函数流程（程序流程框图见10-13所示）**

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

### 附件：代码

|  |
| --- |
| /\* File: ioports.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 = HS, FNOSC = FRCPLL, FPBDIV = DIV\_1  // Disable JTAG to release PORTB  #pragma config FUSBIDIO = OFF //FUSBIDIO为端口控制  #pragma config JTAGEN = OFF //JTAG disable  // 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 light\_cnt=0,light\_flag=1;  void **LightInit**()  {  PORTSetPinsDigitalOut(IOPORT\_B, BIT\_7 | BIT\_8 | BIT\_9 | BIT\_13 );  }  void **Light**()  {  static int light = 0;  //将相关IO口置高电平  PORTSetBits(IOPORT\_B, BIT\_7 | BIT\_8 | BIT\_9 | BIT\_13);  switch (light) { //light status  case 0: //将PORTB.9置低电平  PORTClearBits(IOPORT\_B, BIT\_9);  break;  case 1: //将PORTB.8置低电平  PORTClearBits(IOPORT\_B, BIT\_8);  break;  case 2: //将PORTB.7置低电平  PORTClearBits(IOPORT\_B, BIT\_7);  break;  case 3: //将PORTB.13置低电平  PORTClearBits(IOPORT\_B, BIT\_13);  break;  case 4: //将PORTB.7置低电平  PORTClearBits(IOPORT\_B, BIT\_7);  break;  case 5: //将PORTB.8置低电平  PORTClearBits(IOPORT\_B, BIT\_8);  break;  default:  break;  }  light++;  if (light > 5) light = 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);  light\_cnt++;  if(light\_cnt > 100) //100ms  {  light\_cnt = 0;  light\_flag = 1;  }  }  int **main**(void)  {  int task=0;  SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  INTDisableInterrupts();  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  LightInit();  Timer1Init();  INTEnableInterrupts();  while(1)  {  switch(task)  {  case 0:  if(light\_flag > 0)  {  light\_flag = 0;  Light();  }  break;  default:  break;  }  task ++;  if(task > 0) task = 0;  }  return 1;  } |

# 示例四 定时器定时应用例程

### 功能描述及适用范围

本节描述了在微芯PIC32MX220F032B型芯片上的定时器的示例代码。代码中，实现了秒表功能，通过两个按钮控制秒表的启动\停止、复位，通过SPI主控的LED数码管显示秒表值，秒表计时范围0~999.9秒。

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。

### 硬件配置

表11-1 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 |
| 4 | RA0 | 2 | ANSELAbits.ANSA0 = 0 | PA.0，按钮：启动\暂停 |
| 5 | RA1 | 3 | ANSELAbits.ANSA1 = 0 | PA.1，按钮：复位 |

开关量输入按键电路电位变化中断编程：在硬件系统中，通过按键开通或闭合来产生高或低电平，从而实现控制信号的键入。如图11-4所示，采用10千欧的电阻与K1～K4开关串联的方案设计按键模块，当某按键断开的时候，其相应的输出信号呈现出高电平；当按键闭合时，呈现出低电平。其输出信号直接与PIC32MX输入输出端口相连，提供相应的控制信息。

本示例中使用了K1和K2两个按钮，其中K1为秒表的“启动\暂停”，K2为秒表的“复位”。

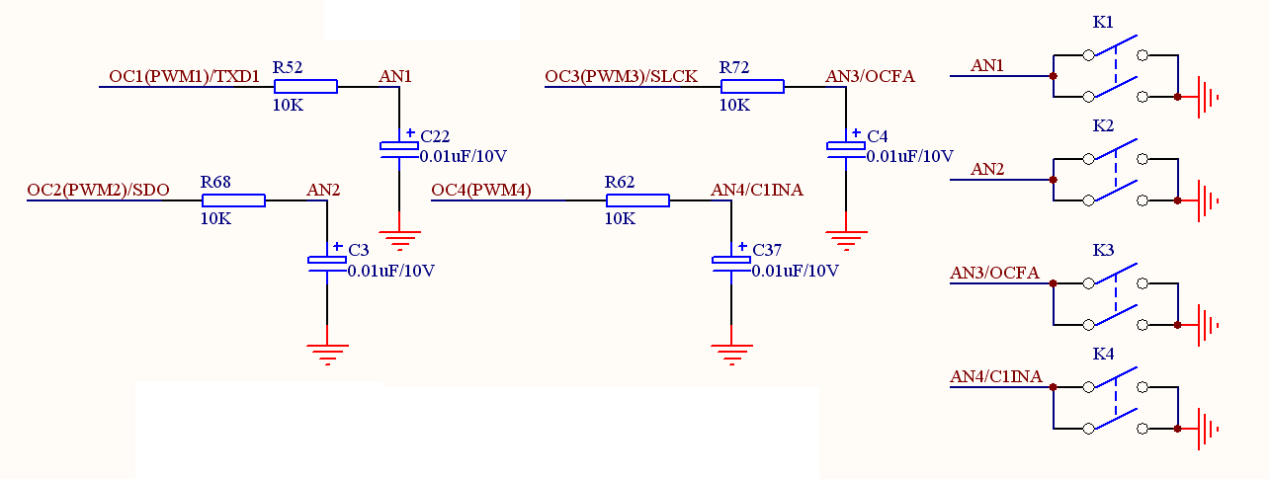


图11-4： 按键模块

七段数码管显示模块如图11-5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。

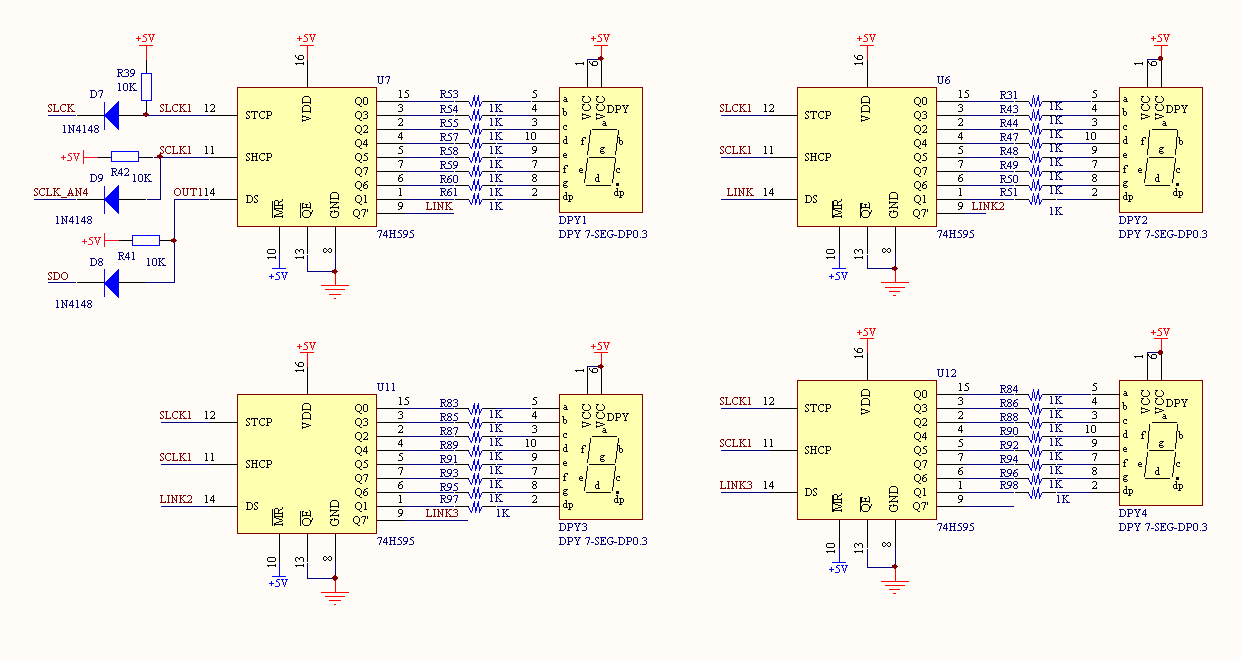


图11-5：3.3V输出电平转换到5V输入电平的转换电路及LED七段数码管驱动电路

### 程序算法框图



图11-6：主函数流程框图

### 各部分代码及算法流程框图

**1、主函数例程（程序流程框图见11-6所示）**

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



图11-7：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11-7所示）**

|  |
| --- |
| void Led()  {  unsigned char ledBuff[4] = {0x00, 0x00, 0x00, 0x00};  static unsigned char ledcnt[4]={0x00, 0x0A, 0x00, 0x00};  int i;  switch(op)  {  case null:  case reset:  for(i=0;i<4;i++)  {  ledcnt[i] = 0;  }  ledcnt[1] = 10;  break;  case start:  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;  }  }  }  }  break;  case pause:  break;  default:  break;  }  for (i = 0; i < 4; i++)  ledBuff[i] = Led\_lib[ledcnt[i]];  SpiDoBurst(ledBuff, 4);  } |



图11-8：定时器中断函数流程框图

**3、定时器中断函数例程（程序流程框图见11-8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;  }  } |

### 附件：代码

|  |
| --- |
| /\*  \* File: Timer.c  \* 定时器上实现的秒表程序：计时范围0~999.9秒  \* 按钮功能定义：  \* Button1：启动/暂停  \* Button2：复位  \*/  #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 BTN\_DELAY 5 //1\*5=5ms  #define SYS\_FREQ (48000000L)  typedef enum \_OP{  null,  start,  pause,  reset  }OP;  unsigned int led\_cnt=0,btn\_cnt=0,led\_flag=1,btn\_flag=0;  OP op=null;  //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;  switch(op)  {  case null:  case reset:  for(i=0;i<4;i++)  {  ledcnt[i] = 0;  }  ledcnt[1] = 10;  break;  case start:  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;  }  }  }  }  break;  case pause:  break;  default:  break;  }  for (i = 0; i < 4; i++)  ledBuff[i] = Led\_lib[ledcnt[i]];  SpiDoBurst(ledBuff, 4);  }  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;  }  btn\_cnt++;  if(btn\_cnt > 5) //5ms  {  btn\_cnt = 0;  btn\_flag = 1;  }  }  void BtnInit()  {  ANSELAbits.ANSA0 = 0; //Button1  ANSELAbits.ANSA1 = 0; //Button2  }  void ButtonScan(void)  {  static int btn1=0,btn2=0,btn3=0;  if(PORTAbits.RA0 == 0)  {  btn1 ++;  if(btn1 == BTN\_DELAY) //Button1 Pressed  {  switch(op)  {  case null:  case pause:  case reset:  op = start;  break;  case start:  op = pause;  break;  default:  break;  }  }  }  else  btn1 = 0;  if(PORTAbits.RA1 == 0) //Button2 Pressed  {  btn2++;  if(btn2 == BTN\_DELAY)  {  op = reset;  }  }  else  btn2 = 0;  }  int **main**(void)  {  int task=0;  SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  INTDisableInterrupts();  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  SpiInitDevice();  BtnInit();  Timer1Init();  INTEnableInterrupts();  while(1)  {  switch(task)  {  case 0:  if(led\_flag > 0)  {  led\_flag = 0;  Led();  }  break;  case 1:  if(btn\_flag > 0)  {  btn\_flag = 0;  ButtonScan();  }  default:  break;  }  task ++;  if(task > 1) task = 0;  }  return 1;  } |

# 示例五 PWM输出方波的例程

**功能描述及适用范围**

本节描述了在微芯PIC32MX220F032B型芯片上的PWM输出示例。通过4个PWM输出通道输出循环变化占空比的PWM方波，调整4个LED灯的亮度，达到“呼吸灯”的效果。

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。

### 硬件配置

表13-1 PWM输出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需代码 | 说明 |
| 1 | RPB7 | 16 | RPB7Rbits.RPB7R = 0b0101 | 复用引脚RPB7，配置为OC1输出 |
| 2 | RPB8 | 17 | RPB8Rbits.RPB8R = 0b0101 | 复用引脚RPB8，配置为OC2输出 |
| 3 | RPB9 | 18 | RPB9Rbits.RPB9R = 0b0101 | 复用引脚RPB9，配置为OC3输出 |
| 4 | RPB13 | 24 | RPB13Rbits.RPB13R = 0b0101 | 复用引脚RPB13，配置为OC4输出 |

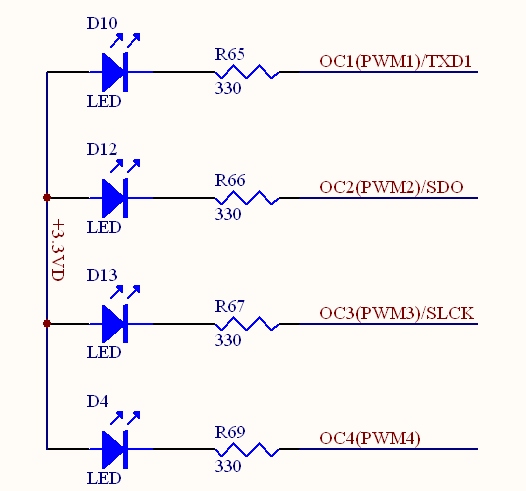


图13-2：PWM输出硬件接口示意图

### 程序算法框图

图13-3：主函数流程框图 图13-4：PWM初始化函数流程框图 图13-5：定时器中断函数流程框图

### 各部分代码

**1、主函数例程（程序流程框图见图13-3）**

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

**2、PWM初始化函数例程（程序流程框图见图13-4）**

|  |
| --- |
| void **PWMinit**()  {  //PWM引脚关联  RPB7Rbits.RPB7R = 0b0101; //PWM1  RPB8Rbits.RPB8R = 0b0101; //PWM2  RPB9Rbits.RPB9R = 0b0101; //PWM3  RPB13Rbits.RPB13R = 0b0101; //PWM4  //PWM1初始化  OC1CON = 0x0000; // Turn off OC1 while doing setup.  OC1RS = pwm1; // Initialize secondary Compare Register  OC1CON = 0x0006; // Configure for PWM mode  //PWM2初始化  OC2CON = 0x0000; // Turn off OC1 while doing setup.  OC2RS = pwm1; // Initialize secondary Compare Register  OC2CON = 0x0006; // Configure for PWM mode  //PWM3初始化  OC3CON = 0x0000; // Turn off OC1 while doing setup.  OC3RS = pwm1; // Initialize secondary Compare Register  OC3CON = 0x0006; // Configure for PWM mode  //PWM4初始化  OC4CON = 0x0000; // Turn off OC1 while doing setup.  OC4RS = pwm1; // Initialize secondary Compare Register  OC4CON = 0x0006; // Configure for PWM mode  //定时器2周期设定+开启  PR2 = PWM\_PR; // Set period  T2CONSET = 0x8000; // 使能 Timer2  //PWM1~4开启  OC1CONSET = 0x8000; // 使能 OC1  OC2CONSET = 0x8000; // 使能 OC2  OC3CONSET = 0x8000; // 使能 OC3  OC4CONSET = 0x8000; // 使能 OC4  } |

**3、定时器中断函数例程（程序流程框图见图13-5）**

|  |
| --- |
| void \_\_ISR(\_TIMER\_1\_VECTOR, ipl2) **Timer1Handler**(void)  {  // Clear the interrupt flag  INTClearFlag(INT\_T1);  if(pwm1\_d == 0)  {  pwm1 ++;  if(pwm1 > PWM\_PR)  {  pwm1 = PWM\_PR;  pwm1\_d = 1;  }  }  else  {  if(pwm1 == 0)  {  pwm1 = 0;  pwm1\_d = 0;  }  else  pwm1 --;  }  OC1RS = pwm1;  OC2RS = pwm1;  OC3RS = pwm1;  OC4RS = pwm1;  } |

### 附件：代码

|  |
| --- |
| /\* File: PWMExample.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, FSOSCEN = OFF  #pragma config FUSBIDIO = OFF //FUSBIDIO为端口控制  #pragma config JTAGEN = OFF //JTAG disable  // Period needed for timer 1 to trigger an interrupt every 0.1 second  // (48MHz PBCLK / 1 = 48000000KHz Timer 1 clock)  #define PERIOD 9600 //48000/48000000 = 0.001s = 0.2ms  #define BTN\_DELAY 2 //2\*5=10ms  #define SYS\_FREQ (48000000L)  #define PWM\_PR 0x0FFF //PWM周期  UINT16 pwm1=0,pwm1\_d=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);  if(pwm1\_d == 0)  {  pwm1 ++;  if(pwm1 > PWM\_PR)  {  pwm1 = PWM\_PR;  pwm1\_d = 1;  }  }  else  {  if(pwm1 == 0)  {  pwm1 = 0;  pwm1\_d = 0;  }  else  pwm1 --;  }  OC1RS = pwm1;  OC2RS = pwm1;  OC3RS = pwm1;  OC4RS = pwm1;  }  void **PWMinit**()  {  //PWM引脚关联  RPB7Rbits.RPB7R = 0b0101; //PWM1  RPB8Rbits.RPB8R = 0b0101; //PWM2  RPB9Rbits.RPB9R = 0b0101; //PWM3  RPB13Rbits.RPB13R = 0b0101; //PWM4  //PWM1初始化  OC1CON = 0x0000; // Turn off OC1 while doing setup.  OC1RS = pwm1; // Initialize secondary Compare Register  OC1CON = 0x0006; // Configure for PWM mode  //PWM2初始化  OC2CON = 0x0000; // Turn off OC1 while doing setup.  OC2RS = pwm1; // Initialize secondary Compare Register  OC2CON = 0x0006; // Configure for PWM mode  //PWM3初始化  OC3CON = 0x0000; // Turn off OC1 while doing setup.  OC3RS = pwm1; // Initialize secondary Compare Register  OC3CON = 0x0006; // Configure for PWM mode  //PWM4初始化  OC4CON = 0x0000; // Turn off OC1 while doing setup.  OC4RS = pwm1; // Initialize secondary Compare Register  OC4CON = 0x0006; // Configure for PWM mode  //定时器2周期设定+开启  PR2 = PWM\_PR; // Set period  T2CONSET = 0x8000; // 使能 Timer2  //PWM1~5开启  OC1CONSET = 0x8000; // 使能 OC1  OC2CONSET = 0x8000; // 使能 OC2  OC3CONSET = 0x8000; // 使能 OC3  OC4CONSET = 0x8000; // 使能 OC4  }  int **main**(void)  {  SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  INTDisableInterrupts();  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  PWMinit();  Timer1Init();  INTEnableInterrupts();  while(1)  ;  return 0;  } |

# 示例六 由PWM输出构成 DA模拟量输出和将其采样的AD例程

### 功能描述及适用范围

本节描述了在微芯PIC32MX220F032B型芯片上的DA-AD综合示例。通过PWM输出占空比从0~100%渐变循环变化的数字信号，该信号通过RC滤波后，接入10位AD接口AN1～AN4，并将AD采样结果通过SPI接口输出给8段数码管显示（0～1023）。

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。

### 硬件配置

表13-2 SPI引脚和A/D引脚选择硬件配置表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 功能描述 | 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 | RPB7 | 16 | RPB7Rbits.RPB7R = 0b0101 | 复用引脚RPB7，配置为OC1输出 |
| 5 | AN0 | 2 | ANSELAbits.ANSA0 = 1 | PORTA.0，使能为模拟通道0 |

基于PWM的D/A变换电路：采用PIC32MX220F032B型芯片所具有的脉宽调制PWM通道输出可变占空比的PWM波形，将PWM波形通过RC滤波电路将其形成大小可调的模拟量，即可实现D/A转换。如图13-6所示。

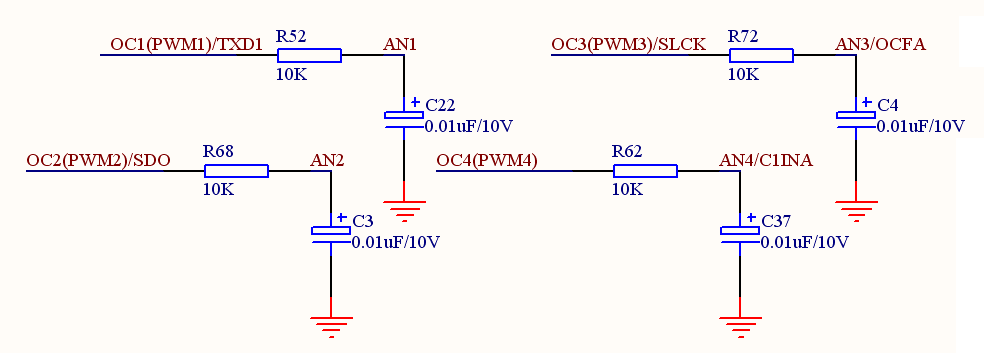
图13-6：PWM滤波电路

图13-6中所示的为较简单的RC电路构成的一阶滤波电路，电阻的阻值采用10千欧，电容的大小采用0.01微法/10V，滤波电路的输出引脚AN1～AN4与芯片的模拟输入引脚连接，可以用万用表直流电压档检测D/A输出的电压值，也可以用A/D转换模块将其电压值转换成数字量送到LED七段数码显示器显示出转换值。

**程序算法框图**



图13-7：主函数流程框图

### 各部分代码及算法流程框图

**1、主函数例程（程序流程框图见图13-7所示）**

|  |
| --- |
| int **main**(void)  {  int i,ads;  SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  INTDisableInterrupts();  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  SpiInitDevice();  AD10init();  PWMinit();  Timer1Init();  INTEnableInterrupts();  while(1)  {  if(ADS\_flag > 0)  {  ADS\_flag = 0;  adrst[adptr] = AD10Sample();  adptr++;  if(adptr > 15)  {  adptr = 0;  ads = 0;  for(i=0;i<16;i++)  ads += adrst[i];  ads = ads >> 4;  AD10DispRst(ads);  }  }  }  return 0;  } |

图13-8：PWMinit函数流程框图 图13-9：定时器中断函数流程框图 图13-10：AD采样函数流程框图

**2、PWMinit函数例程（程序流程框图见图13-8所示）**

|  |
| --- |
| void **PWMinit**()  {  //PWM引脚关联  RPB7Rbits.RPB7R = 0b0101; //PWM1  //PWM1初始化  OC1CON = 0x0000; // Turn off OC1 while doing setup.  OC1RS = pwm1; // Initialize secondary Compare Register  OC1CON = 0x0006; // Configure for PWM mode  //定时器2周期设定+开启  PR2 = PWM\_PR; // Set period  T2CONSET = 0x8000; // Enable Timer2  //PWM1开启  OC1CONSET = 0x8000; // Enable OC1  } |

**3、定时器中断函数例程（程序流程框图见图13-9所示）**

|  |
| --- |
| // Configure the Timer 1 interrupt handler  void \_\_ISR(\_TIMER\_1\_VECTOR, ipl2) **Timer1Handler**(void)  {  // Clear the interrupt flag  INTClearFlag(INT\_T1);    ADS\_cnt++;  if(ADS\_cnt > 10) //0.01s  {  ADS\_cnt = 0;  ADS\_flag = 1;  }  pwm\_cnt++;  if(pwm\_cnt > 1)  {  pwm\_cnt = 0;  if(pwm1\_d == 0)  {  pwm1 ++;  if(pwm1 > DUTYMAX )  {  pwm1 = DUTYMAX;  pwm1\_d = 1;  }  }  else  {  if(pwm1 == 0)  {  pwm1 = 0;  pwm1\_d = 0;  }  else  pwm1 --;  }  OC1RS = pwm1;  }  } |

**4、AD采样函数例程（程序流程框图见图13-10所示）**

|  |
| --- |
| UINT16 **AD10Sample**(void)  {  AD1CON1bits.ASAM = 1; // 自动采样：31个Tad后自动转换  while (!AD1CON1bits.DONE); // 等待转换完成  AD1CON1bits.ASAM = 0; // 结束本次采样/转换操作  return ADC1BUF0; //返回采样结果  } |

### 附件：代码

|  |
| --- |
| /\*  \* File: DAAD\_Example.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 = HS, FNOSC = FRCPLL, FPBDIV = DIV\_1  #pragma config FUSBIDIO = OFF //FUSBIDIO为端口控制  #pragma config JTAGEN = OFF //JTAG disable  // 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 = 0.2ms  #define BTN\_DELAY 2 //2\*5=10ms  #define SYS\_FREQ (48000000L)  #define PWM\_PR 0x0FFF  #define DUTYMAX 0x0DFF  UINT16 pwm1=0,pwm1\_d=0,pwm\_cnt=0;  unsigned int ADS\_cnt=0,ADS\_flag=1;  unsigned int adrst[16]={0},adptr=0;  //数码管显示数据 分别代表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  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 **AD10init**(void)  {  ANSELAbits.ANSA0 = 1; // PORTA.0选择为AN0模拟输入  AD1CON1 = 0x00E0; // 自动采样  AD1CHS = 0x00000000; // CH0和CH1均使用AN0通道  AD1CSSL = 0;  AD1CON3 = 0x0203; // 采样时间 = 2Tad  AD1CON2 = 0x6004; // 选择 VREF+ 和 VREF- 作为参考  // 采样2次后产生中断信号  AD1CON1bits.ADON = 1; // 开启AD  }  UINT16 **AD10Sample**(void)  {  AD1CON1bits.ASAM = 1; // 自动采样：31个Tad后自动转换  while (!AD1CON1bits.DONE); // 等待转换完成  AD1CON1bits.ASAM = 0; // 结束本次采样/转换操作  return ADC1BUF0; //返回采样结果  }  void **AD10DispRst**(UINT16 rst)  {  static BYTE spibuff[4];  spibuff[2] = Led\_lib[rst % 10]; //个位  spibuff[1] = Led\_lib[(rst / 10) % 10]; //十位  spibuff[0] = Led\_lib[(rst / 100) % 10]; //百位  spibuff[3] = Led\_lib[rst / 1000]; //千位  SpiDoBurst(spibuff,4);  }  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);  }  // Configure the Timer 1 interrupt handler  void \_\_ISR(\_TIMER\_1\_VECTOR, ipl2) **Timer1Handler**(void)  {  // Clear the interrupt flag  INTClearFlag(INT\_T1);  ADS\_cnt++;  if(ADS\_cnt > 10) //0.01s  {  ADS\_cnt = 0;  ADS\_flag = 1;  }  pwm\_cnt++;  if(pwm\_cnt > 1)  {  pwm\_cnt = 0;  if(pwm1\_d == 0)  {  pwm1 ++;  if(pwm1 > DUTYMAX )  {  pwm1 = DUTYMAX;  pwm1\_d = 1;  }  }  else  {  if(pwm1 == 0)  {  pwm1 = 0;  pwm1\_d = 0;  }  else  pwm1 --;  }  OC1RS = pwm1;  }  }  void **PWMinit**()  {  //PWM引脚关联  RPB7Rbits.RPB7R = 0b0101; //PWM1  //PWM1初始化  OC1CON = 0x0000; // Turn off OC1 while doing setup.  OC1RS = pwm1; // Initialize secondary Compare Register  OC1CON = 0x0006; // Configure for PWM mode  //定时器2周期设定+开启  PR2 = PWM\_PR; // Set period  T2CONSET = 0x8000; // Enable Timer2  //PWM1开启  OC1CONSET = 0x8000; // Enable OC1  }  int **main**(void)  {  int i,ads;  SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  INTDisableInterrupts();  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  SpiInitDevice();  AD10init();  PWMinit();  Timer1Init();  INTEnableInterrupts();  while(1)  {  if(ADS\_flag > 0)  {  ADS\_flag = 0;  adrst[adptr] = AD10Sample();  adptr++;  if(adptr > 15)  {  adptr = 0;  ads = 0;  for(i=0;i<16;i++)  ads += adrst[i];  ads = ads >> 4;  AD10DispRst(ads);  }  }  }  return 0;  } |

# 示例七 SPI例程

**功能描述**

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。

**硬件配置**

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。

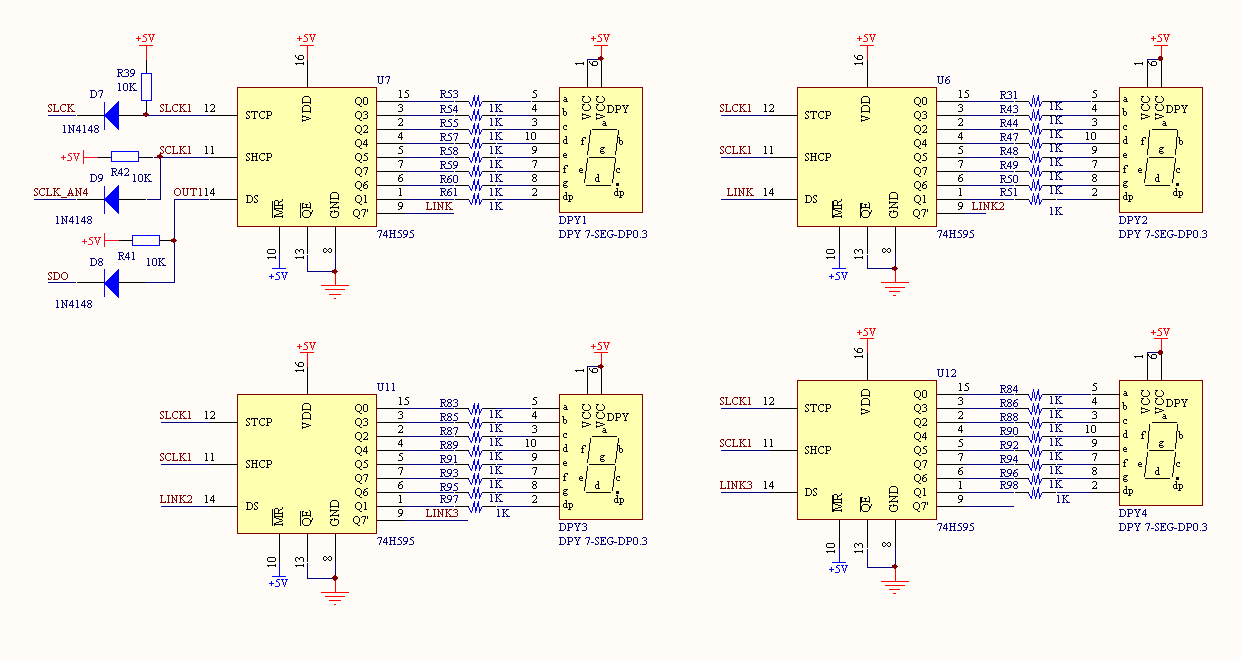


图14-2：3.3V输出电平转换到5V输入电平的转换电路及LED七段数码管驱动电路

**74HC595简介及用法**

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。

**程序算法框图**

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

### 各部分代码

**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;  } |

# 示例八 RS-232例程

**功能描述及适用范围**

本节描述了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。

**硬件配置**

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" \t "_blank)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" \t "_blank)上常用的[通讯接口](http://baike.baidu.com/view/135196.htm" \t "_blank)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" \t "_blank)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" \t "_blank)、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。

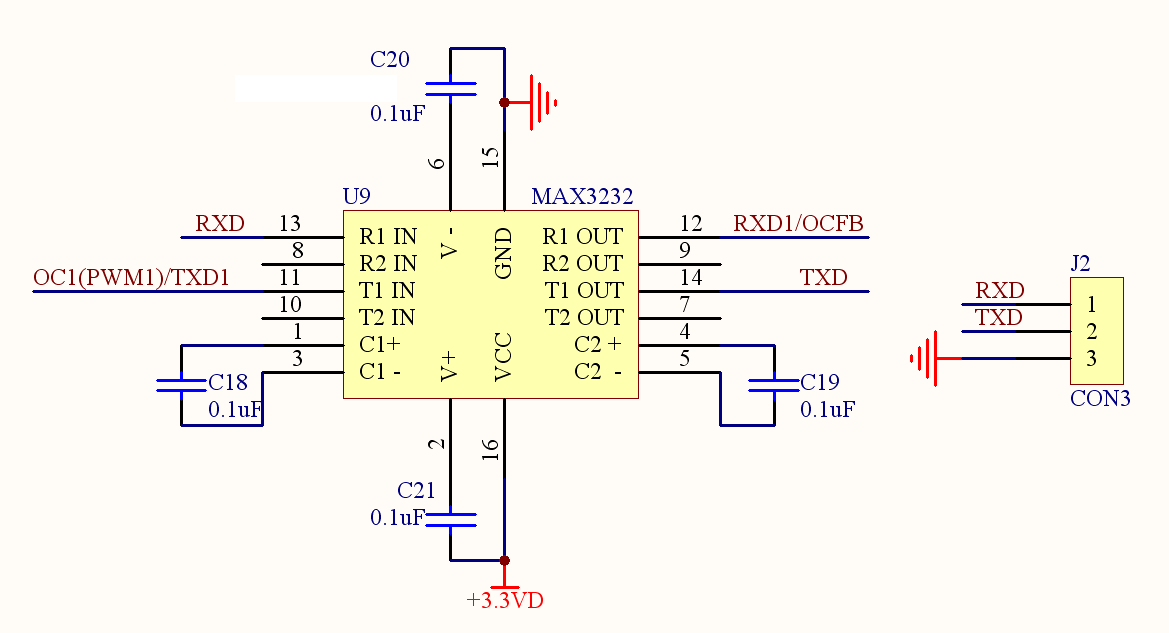


图16-4：RS-232接口电路图

### 程序算法框图



图16-5：主函数流程框图

### 各部分代码及算法流程框图

**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;  }  } |

# 示例九 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。

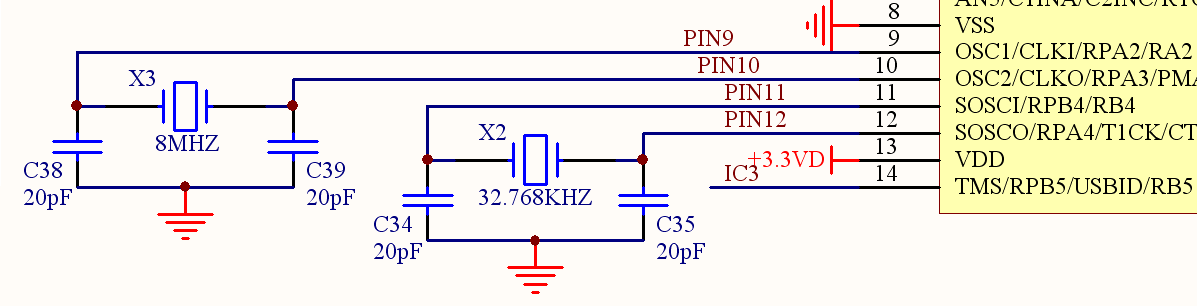


图18-2： 日历时钟晶体振荡器电路

### 功能描述及适用范围

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。

### 硬件配置

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。

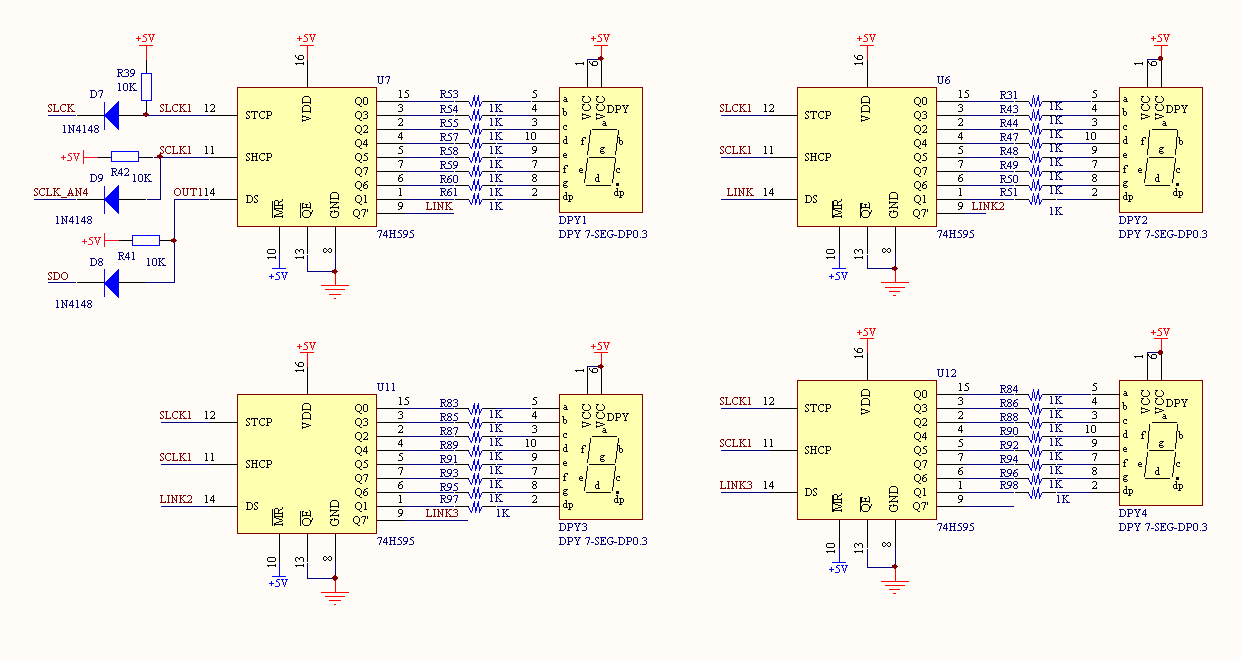


图18-3：3.3V输出电平转换到5V输入电平的转换电路及LED七段数码管驱动电路

**程序算法框图**



图18-4 主函数流程框图

### 各部分代码及算法流程框图

**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;  } |

# 示例十 A/D例程

### **功能描述及适用范围**

本节描述了在微芯PIC32MX220F032B型芯片上的AD输入程序示例。示例中将AD采样的原始值通过SPI控制的7段LED数码显示器显示出来，显示范围0～1023。

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。

**硬件配置**

表19-1 A/D引脚和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 | AN10 | 25 | ANSELBbits.ANSB14 = 1 | PORTB.14，使能为模拟输入 |
| 2 | SCK2 | 26 | 由SPI模块自动选择(SCK2只能选这个引脚) | SPI数据时钟 |
| 3 | SDO2 | 17 | PPSOutput(2, RPB8, SDO2) | SPI数据输出 |
| 4 | SLCK | 18 | PORTSetPinsDigitalOut(IOPORT\_B, BIT\_9) | 外部移位寄存器数据锁存 |

在便携式实验板上A/D输入通道设计了4路，即AN1～AN4，可以从J14接口从板外输入模拟量，见图19-5所示，也可以由OC1～OC4（即PWM1～PWM4脉宽调制）输出可变占空比的PWM波形，PWM波形经过RC滤波后形成可变的模拟信号，由A/D采样即可得到模拟信号的幅值，通过SPI接口送到LED数码管上显示出来。同时AN1～AN4引脚也可以作为开关量的通用输入输出引脚使用，这种使用方式可以将K1～K4按键开关读入到单片机中，见图19-4所示。

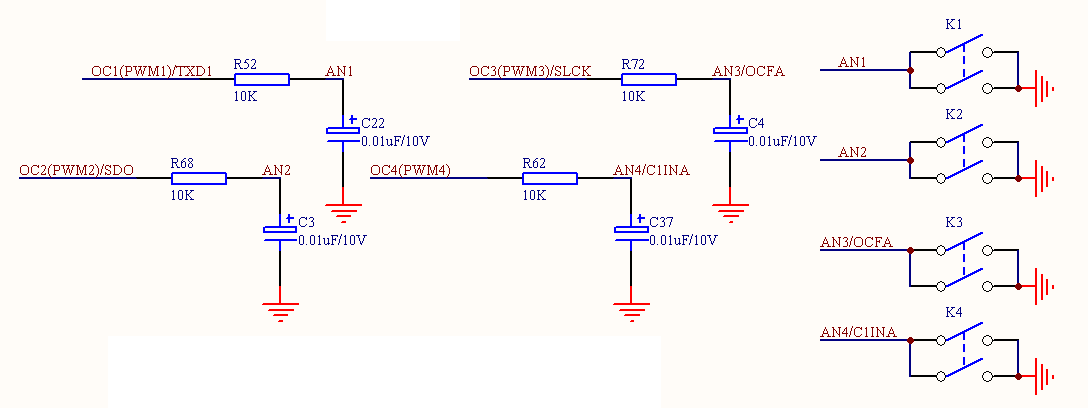


图19-4： A/D输入与按键接口复用电路图

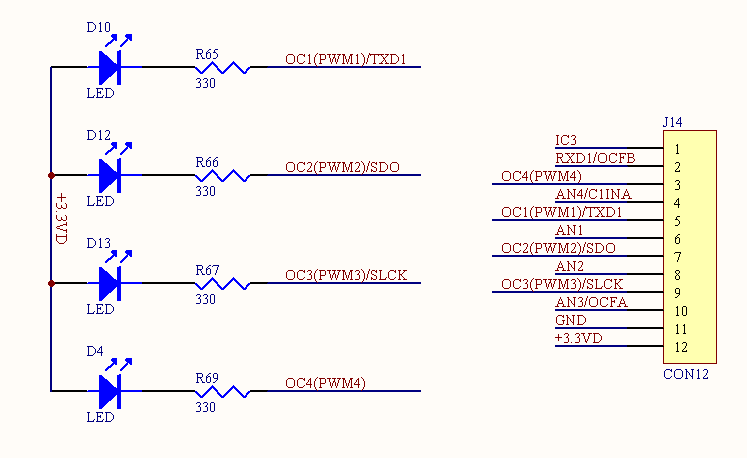


图19-5： A/D输入外接电路图

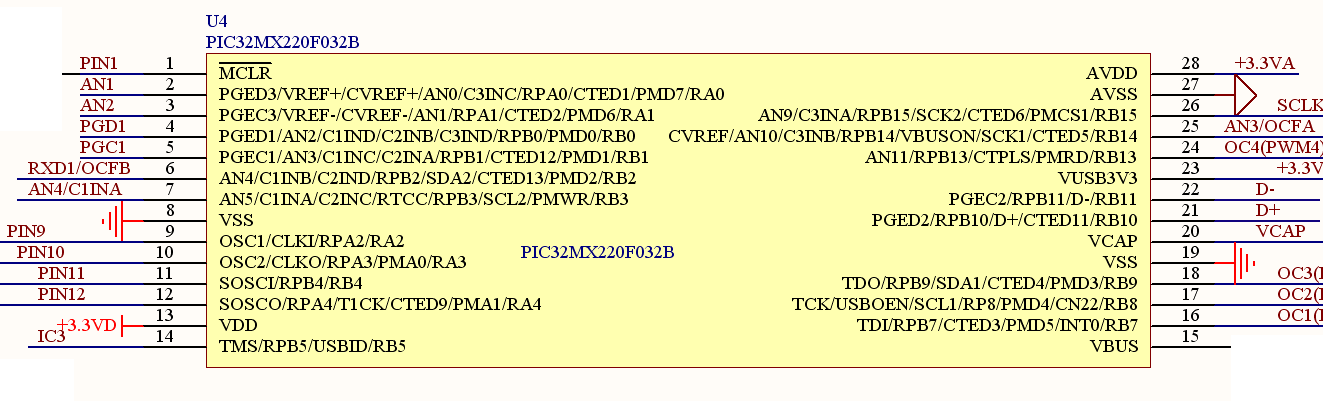


图19-6：A/D输入到芯片的引脚图

**程序算法框图**

图19-7：主函数流程框图 图19-8：AD采样函数流程框图 图19-9：定时器中断函数流程框图

### 各部分代码及算法流程框图

**1、主函数例程（程序流程框图见图19-7所示）**

|  |
| --- |
| int **main**(void)  {  SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  SpiInitDevice();  AD10init();  INTDisableInterrupts();  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  Timer1Init();  INTEnableInterrupts();  while(1)  {  if(ADS\_flag > 0)  {  ADS\_flag = 0;  AD10DispRst(AD10Sample());  }  }  return 1;  } |

**2、AD采样函数例程（程序流程框图见图19-8所示）**

|  |
| --- |
| UINT16 **AD10Sample**(void)  {  AD1CON1bits.ASAM = 1; // 自动采样：31个Tad后自动转换  while (!AD1CON1bits.DONE); // 等待转换完成  AD1CON1bits.ASAM = 0; // 结束本次采样/转换操作  return ADC1BUF0; //返回采样结果  } |

**3、定时器中断函数例程（程序流程框图见图19-9所示）**

|  |
| --- |
| void \_\_ISR(\_TIMER\_1\_VECTOR, ipl2) **Timer1Handler**(void)  {  // Clear the interrupt flag  INTClearFlag(INT\_T1);  ADS\_cnt++;  if(ADS\_cnt > 50) //0.05s  {  ADS\_cnt = 0;  ADS\_flag = 1;  }  } |

### 附件：代码

|  |
| --- |
| /\*  \* File: main.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 = HS, 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)  //数码管显示数据 分别代表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}; //LED字库  /\*-------LED段码分布图------  ---0---  | |  7 3  | |  ---6---  | |  5 2  | |  ---4--- 1  ----------------------------\*/  unsigned int ADS\_cnt=0,ADS\_flag=1;  unsigned int adrst=0;  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 **AD10init**(void)  {  ANSELBbits.ANSB14 = 1; // PORTB.14选择为AN10模拟输入  AD1CON1 = 0x00E0; // 自动采样  AD1CHS = 0xA0A0000; // CH0和CH1均使用AN10通道  AD1CSSL = 0;  AD1CON3 = 0x0203; // 采样时间 = 2Tad  AD1CON2 = 0x6004; // 选择 VREF+ 和 VREF- 作为参考  // 采样2次后产生中断信号  AD1CON1bits.ADON = 1; // 开启AD  }  UINT16 **AD10Sample**(void)  {  AD1CON1bits.ASAM = 1; // 自动采样：31个Tad后自动转换  while (!AD1CON1bits.DONE); // 等待转换完成  AD1CON1bits.ASAM = 0; // 结束本次采样/转换操作  return ADC1BUF0; //返回采样结果  }  void **AD10DispRst**(UINT16 rst)  {  static BYTE spibuff[4];  spibuff[2] = Led\_lib[rst % 10]; //个位  spibuff[1] = Led\_lib[(rst / 10) % 10]; //十位  spibuff[0] = Led\_lib[(rst / 100) % 10]; //百位  spibuff[3] = Led\_lib[rst / 1000]; //千位  SpiDoBurst(spibuff,4);  }  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);  }  // Configure the Timer 1 interrupt handler  void \_\_ISR(\_TIMER\_1\_VECTOR, ipl2) **Timer1Handler**(void)  {  // Clear the interrupt flag  INTClearFlag(INT\_T1);  ADS\_cnt++;  if(ADS\_cnt > 50) //0.05s  {  ADS\_cnt = 0;  ADS\_flag = 1;  }  }  int **main**(void)  {  SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  SpiInitDevice();  AD10init();  INTDisableInterrupts();  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  Timer1Init();  INTEnableInterrupts();  while(1)  {  if(ADS\_flag > 0)  {  ADS\_flag = 0;  AD10DispRst(AD10Sample());  }  }  return 1;  } |

# 示例十一 USB电路连接图和编程示例

当实际应用USB接口时，必须使用频率准确稳定的主振荡器，将8MHZ的晶体振荡器连接到芯片的OSC1和OSC2引脚上。见图21-2所示。

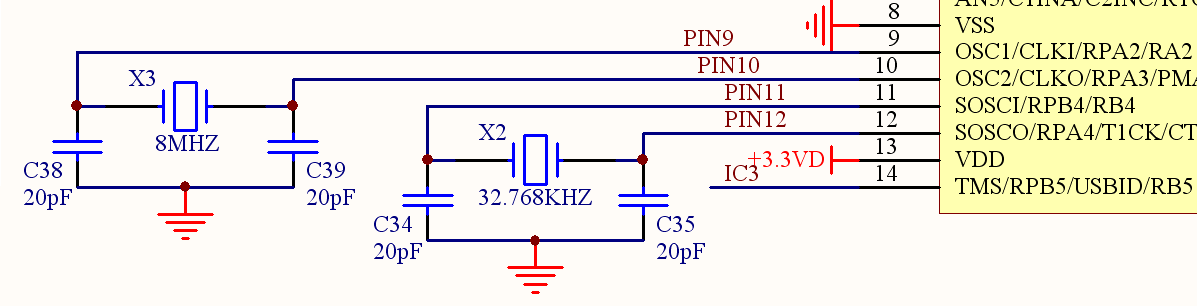


图21-2： USB必须的晶体振荡器电路

**功能描述及使用范围**

本节描述了在微芯PIC32MX220F032B型芯片上的USB\_HID程序示例。该例程实现了该单片机通过USB\_HID方式与计算机通信，示例包括单片机和PC两部分程序。

示例功能描述：PC与单片机之间通过USB通信，在PC端程序面板上勾选或取消“Light”复选框，来点亮便携式开发板上的LED1指示灯；在便携式开发板上按下或弹起按键1，则PC端程序面板上的“Button”复选框用勾选或取消的方式指示按键状态。

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。

USB引脚和USB插头电路见图21-3所示。

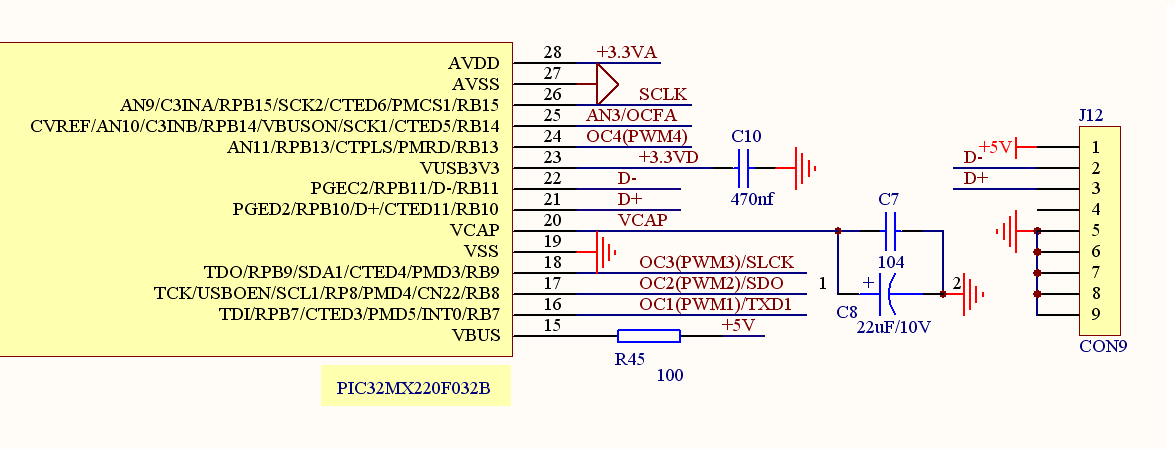


图21-3： USB引脚和USB插头电路图

本示例中用到了2个指示灯和1个按键。其中LED1（D10）用来指示PC端软件面板的“Light”状态，LED4（D4）在运行过程中以1Hz左右的频率闪烁，指示程序正在运行；按键K1的状态会在PC端软件面板上实时显示。

### 硬件配置

表21-1 USB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 | D+ | 21 | 由USB模块指定 | USB数据+ |
| 2 | D- | 22 | 由USB模块指定 | USB数据- |
| 3 | RB7 | 16 | PORTSetPinsDigitalOut(IOPORT\_B, BIT\_7) | Light指示 |
| 4 | RB13 | 24 | PORTSetPinsDigitalOut(IOPORT\_B, BIT\_13) | 程序工作状态指示灯 |
| 5 | RA0 | 2 | ANSELAbits.ANSA0 = 0 | 按键1（K1）输入 |

指示灯及按钮输入的电路见图21-4所示。

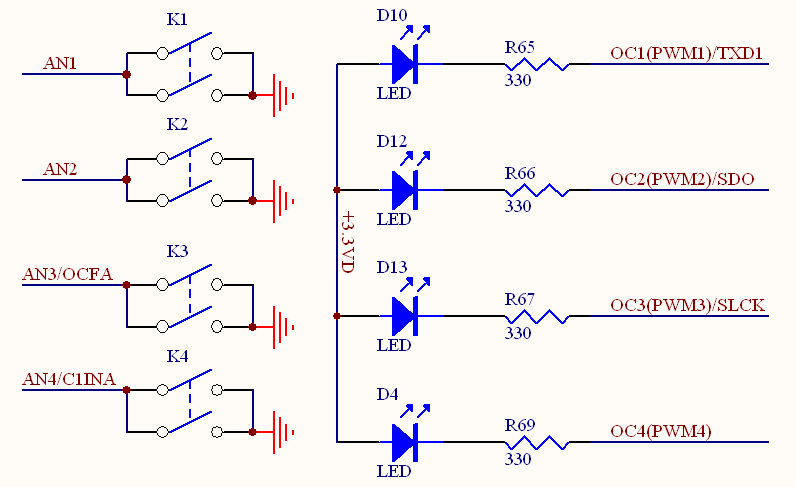


图21-4： 指示灯及按键输入电路图

**程序算法框图**



图21-5：主函数流程框图

### 各部分代码及算法流程框图

**1、主函数例程（程序流程框图见21-5所示）**

|  |
| --- |
| int **main**(void)  {  UINT pbClk;  int task=0;  // Setup configuration  pbClk = SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  InitLED();  INTDisableInterrupts();  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  BtnInit();  Timer1Init();  TRANS\_LAYER\_Init(pbClk);    INTEnableInterrupts();  // Enter firmware upgrade mode if there is a trigger or if the application is not valid  while (1)  {  switch(task)  {  case 0:  TRANS\_LAYER\_Task();  FRAMEWORK\_FrameWorkTask();  BlinkLED();  break;  case 1:  if(btn\_flag > 0)  {  btn\_flag = 0;  ButtonScan();  }  break;  case 2:  if(update\_flag > 0)  {  update\_flag = 0;  UpdateData(Btn\_Status);  }  break;  default:  break;  }  task ++;  if(task > 2) task = 0;  }  return 0;  } |



图21-6：按键扫描函数流程框图

**2、按键扫描函数例程（程序流程框图见21-6所示）**

|  |
| --- |
| void **ButtonScan**(void)  {  static int btn1=0;  if(PORTAbits.RA0 == 0)  {  btn1 ++;  if(btn1 == BTN\_DELAY) //Button1 Pressed  {  ButtonPress(1,TRUE);  }  }  else if(btn1 > 0)  {  btn1 = 0;  ButtonPress(1,FALSE);  }  } |



图21-7：灯Light更新函数流程框图

**3、灯Light状态更新函数例程（程序流程框图见21-7所示）**

|  |
| --- |
| void **Light**(int light,BOOL on)  {  if(on)  PORTClearBits(IOPORT\_B,BIT\_7);  else  PORTSetBits(IOPORT\_B,BIT\_7);  } |

### 附件1：代码

|  |
| --- |
| /\*  \* USB\_HID\_Example.c  \*/  #include "Include\GenericTypeDefs.h"  #include "Include\HardwareProfile\HardwareProfile.h"  #include "Include\USB\_HID\_Example.h"  #include "Include\FrameWork\Framework.h"  #include <stdlib.h>  #include <plib.h>  // Configuring the Device Configuration Registers  // 40Mhz Core/Periph, Pri Osc w/PLL, Write protect Boot Flash  #if defined(TRANSPORT\_LAYER\_USB) || defined(TRANSPORT\_LAYER\_USB\_HOST)  #pragma config UPLLEN = ON // USB PLL Enabled  #pragma config UPLLIDIV = DIV\_2 // USB PLL Input Divider = Divide by 2  #endif  #pragma config DEBUG = OFF // Background Debugger disabled  #pragma config FPLLMUL = MUL\_20 // PLL Multiplier: Multiply by 20  #pragma config FPLLIDIV = DIV\_2 // PLL Input Divider: Divide by 2  #pragma config FPLLODIV = DIV\_2 // PLL Output Divider: Divide by 2  #pragma config FWDTEN = OFF // WD timer: OFF  #pragma config POSCMOD = HS // Primary Oscillator Mode: High Speed xtal  #pragma config FNOSC = PRIPLL // Oscillator Selection: Primary oscillator w/ PLL  #pragma config FPBDIV = DIV\_1 // Peripheral Bus Clock: Divide by 1  #pragma config BWP = OFF // Boot write protect: OFF  #pragma config ICESEL = ICS\_PGx1 // ICE pins configured on PGx1 (PGx2 is multiplexed with USB D+ and D- pins).  #pragma config JTAGEN = OFF //JTAG disable  #define PERIOD 40000 //40000/40000000 = 0.001s = 1ms  #define BTN\_DELAY 5 //2\*1=2ms  #define BTN\_PUSH 200 //200ms  #define SWITCH\_PRESSED 0  unsigned int btn\_cnt=0,btn\_flag=0,update\_flag=1,counter=0;  UINT8 Btn\_Status = 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);  btn\_cnt++;  if(btn\_cnt > 5) //5ms  {  btn\_cnt = 0;  btn\_flag = 1;  counter++;  }  }  void **BtnInit**()  {  ANSELAbits.ANSA0 = 0; //Button  }  void **ButtonScan**(void)  {  static int btn1=0;  if(PORTAbits.RA0 == 0)  {  btn1 ++;  if(btn1 == BTN\_DELAY) //Button1 Pressed  {  ButtonPress(1,TRUE);  }  }  else if(btn1 > 0)  {  btn1 = 0;  ButtonPress(1,FALSE);  }  }  void **ButtonPress**(int button,BOOL press)  {  UINT8 set=0,clear=0xff;  switch(button)  {  case 1:  set = 0x01;  clear = 0xfe;  break;  default:  break;  }  if(press)  Btn\_Status |= set;  else  Btn\_Status &= clear;  update\_flag = 1;  }  void **Light**(int light,BOOL on)  {  if(on)  PORTClearBits(IOPORT\_B,BIT\_7);  else  PORTSetBits(IOPORT\_B,BIT\_7);  }  int **main**(void)  {  UINT pbClk;  int task=0;  // Setup configuration  pbClk = SYSTEMConfig(SYS\_FREQ, SYS\_CFG\_WAIT\_STATES | SYS\_CFG\_PCACHE);  InitLED();  INTDisableInterrupts();  INTConfigureSystem(INT\_SYSTEM\_CONFIG\_MULT\_VECTOR);  BtnInit();  Timer1Init();  TRANS\_LAYER\_Init(pbClk);    INTEnableInterrupts();  // Enter firmware upgrade mode if there is a trigger or if the application is not valid  while (1)  {  switch(task)  {  case 0:  TRANS\_LAYER\_Task();  FRAMEWORK\_FrameWorkTask();  BlinkLED();  break;  case 1:  if(btn\_flag > 0)  {  btn\_flag = 0;  ButtonScan();  }  break;  case 2:  if(update\_flag > 0)  {  update\_flag = 0;  UpdateData(Btn\_Status);  }  break;  default:  break;  }  task ++;  if(task > 2) task = 0;  }  return 0;  } |

### 附件2：示例工程的使用方法

**1、打开项目并选择编程器**

在MPLAB X IDE环境下，打开USB\_HID\_Example\Firmware\MPLAB\_X\_Workspace\USB\_HID\_Btl\_StarterKit.X工程，将USB线连接到便携式开发板的编程器一端，另一端与计算机相连。

在X IDE的项目浏览窗口，右键单击USB\_HID\_Example，并选择Properties，打开项目属性。如图1所示：

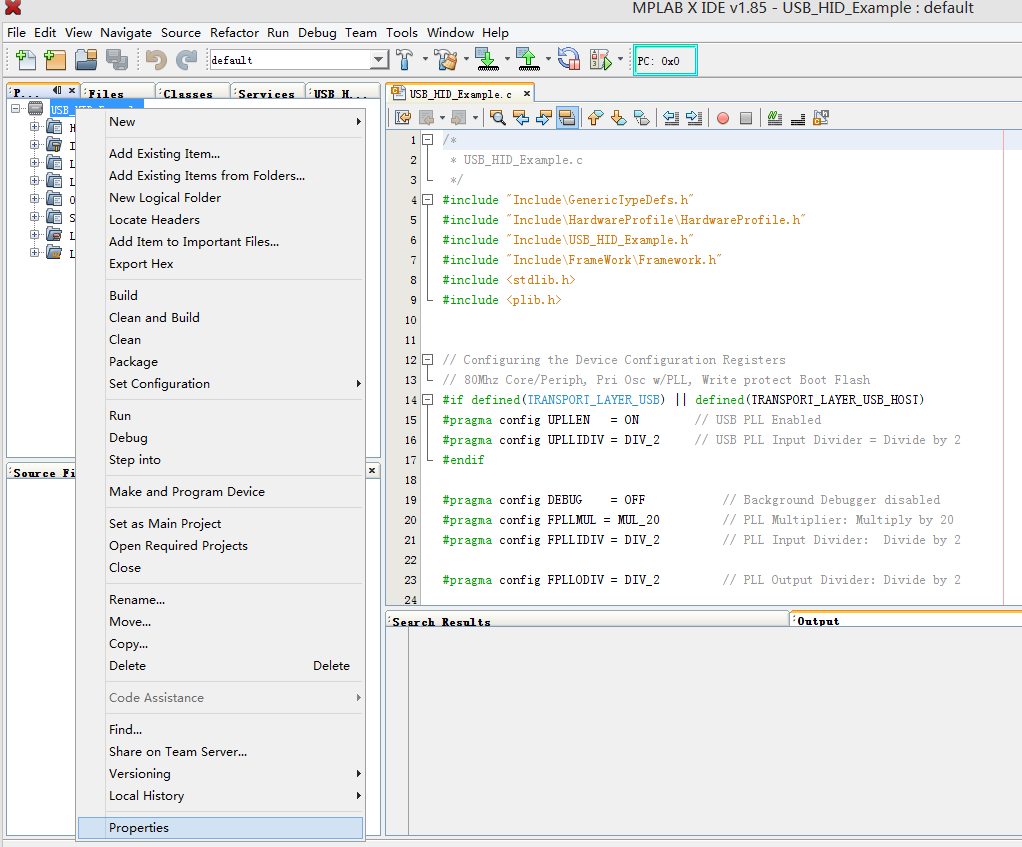


图1：在X IDE环境下打开项目属性

在项目属性界面，将编程器指定为便携式开发板，其默认路径在编程器下面的PICKit3->SN:Default\_PK3，如图2所示：

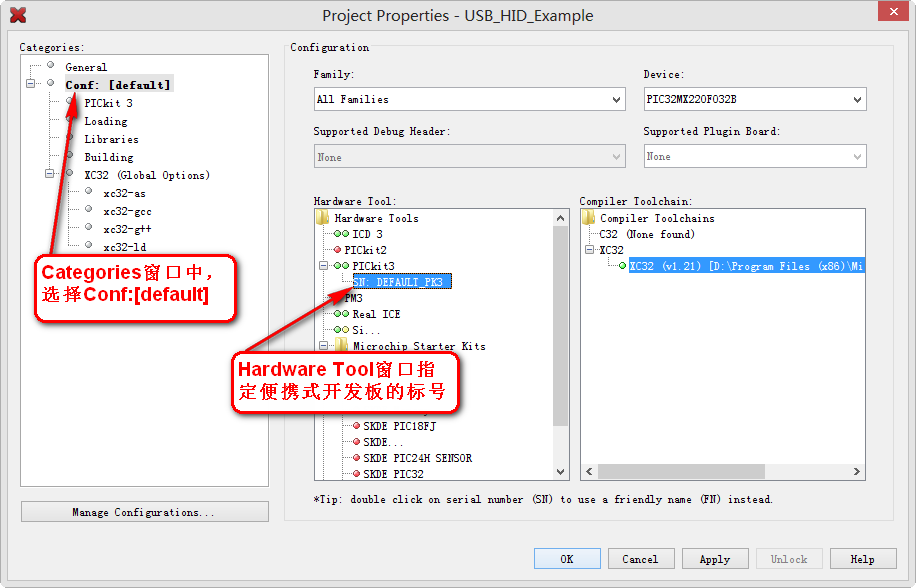


图2：在项目属性窗口指定编程器为便携式开发板

**2、编译并烧写程序代码到PIC**

要烧写代码到PIC便携开发板，必须具备以下硬件条件：

1）便携式开发板的编程端与计算机通过USB相连

2）便携式开发板上的编程跳针已连接（共5个，跨在开发板中间白色竖线上）

在MPLAB X IDE环境下，项目浏览窗口中选中USB\_HID\_Example，然后点击工具栏上的“编译并下载”按钮，开始编译并烧写代码到PIC，如图3所示：

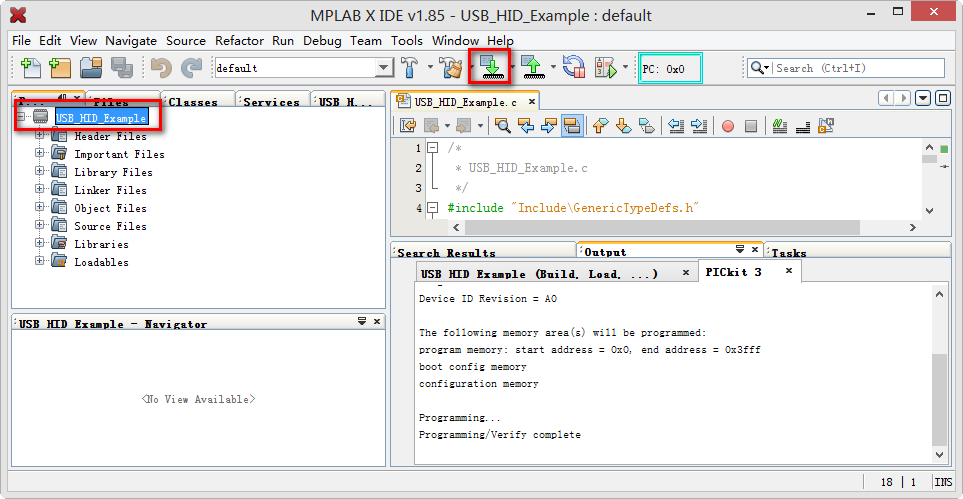


图3：编译并烧写代码

**3、通信测试**

将下载好代码的PIC便携式开发板的示例端USB口计算机连接，注意不要编程端与示例端的两个USB口同时连接到计算机上。

第一次连接时，计算机可能会弹出发现新设备对话框，选择自动安装驱动程序即可。

打开USB\_HID\_Example\PC application\USBHIDEXP.exe，初始界面如图4所示：

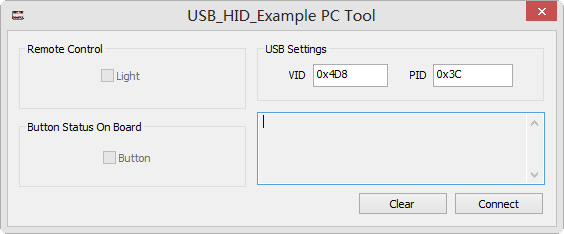


图4：PC端工具主界面

在PC工具主界面上点击“Connect”按钮连接便携式开发板。注意，便携式开发板的USB硬件ID默认为VID=0x4D8，PID=0x3C，如未改变，请在连接时保持该值。

连接成功后，在提示框里提示“Device Connected.”，图如5所示：

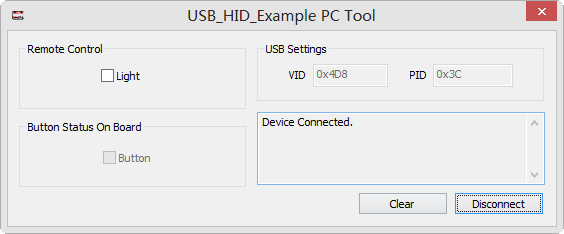


图5：连接成功

勾选或取消界面上的Light，PIC便携式开发板上的LED1会跟着点亮或熄灭。

按动便携式开发板上的按键1（K1），PC软件界面上的“Button”会勾选或取消，分别指示K1的按下与抬起动作。并在提示框中指示接收到的Button按键状态更新指令。如图6所示：

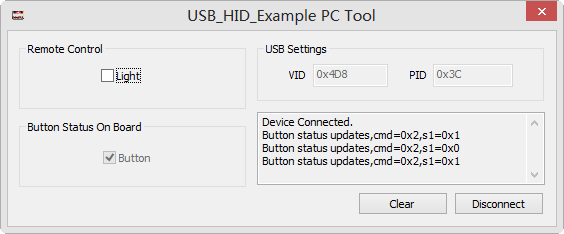


图6：通信测试界面

### 附件3：通信协议

PC主机应用程序与PIC便携式开发板之间使用通信协议进行交互。

**帧格式**

通信协议遵循例1所示的帧格式。帧格式双向（即，从主机应用程序到便携式开发板以及从便携式开发板到主机应用程序）相同。

例1：帧格式

|  |
| --- |
| [<SOH>…]<SOH>[<DATA>…]<CRCL><CRCH><EOT>  其中：  <...> 表示一个字节  [...] 表示可选或可变数量的字节 |

帧以一个控制字符（帧头开始（Start of Header，SOH））开始，以另一个控制字符（传输结束（End of Transmission， EOT））结束。帧的完整性由两个循环冗余校验（Cyclic Redundancy Check，CRC）-16 字节保护，分别表示为CRCL（低字节）和CRCH（高字节）。

**控制字符**

数据字段中的某些字节可能与控制字符SOH 和EOT 相似。数据链路转义（Data Link Escape， DLE）字符用于转义此类会被解释为控制字符的字节。自举程序总是接受<DLE>之后的字节作为数据，并且总是在发送任何控制字符之前先发送一个<DLE>。

表1：控制字符说明

|  |  |  |
| --- | --- | --- |
| 控制 | 十六进制值 | 说明 |
| <SOH> | 0x01 | 标识帧开始 |
| <EOT> | 0x04 | 标识帧结束 |
| <DLE> | 0x10 | 数据链路转义符 |

**命令**

通信中的命令如表2所示。数据字段中的第一个字节用于承载命令。

表2：命令说明

|  |  |  |
| --- | --- | --- |
| 十六进制值 | 传输方向 | 说明 |
| 0x01 | PC🡪便携式开发板 | 更新Light状态 |
| 0x02 | 便携式开发板🡪 PC | 更新Button状态 |

**示例**

更新Light状态由PC端发送至开发板。

例2：更新Light状态示例

|  |
| --- |
| [<SOH>…]<SOH>[<0x01><STATUS>]<CRCL><CRCH><EOT>  其中：  <STATUS> ——Light状态，=0x01点亮LED1，=0x00熄灭LED1 |

更新Button状态由开发板发送至PC端。

例3：更新Button状态示例

|  |
| --- |
| [<SOH>…]<SOH>[<0x02><STATUS>]<CRCL><CRCH><EOT>  其中：  <STATUS> ——Button（K1）状态，=0x01表示K1按下，=0x00表示K1弹起 |

# 示例十二 Complex例程

**功能描述及适用范围**

本节描述了在微芯PIC32MX220F032B型芯片上的综合示例。

本示例集成了SPI控制的LED八段数码管显示、PWM占空比设置、定时器中断、按钮扫描、UART模块的RS232通信（中断方式）、IO口输出等众多功能。

运行时，2个LED灯产生“呼吸灯”的效果；第一个LED数码管数值每秒自动加1，从0到9循环。将两块开发板的RS232通信端口接到一起，按动板卡A的按钮，板卡B上对应的LED数码管数字循环加1，按动板卡B的按钮，板卡A上对应的LED数码管数字循环加1。

适用范围：本文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。

### 硬件配置

表1 Complex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) |
| 9 | RPB13 | 24 | RPB13Rbits.RPB13R = 0b0101 | 将复用引脚RPB13配置为OC4输出 |

#### 七段数码管显示模块

七段数码管显示模块如图1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。

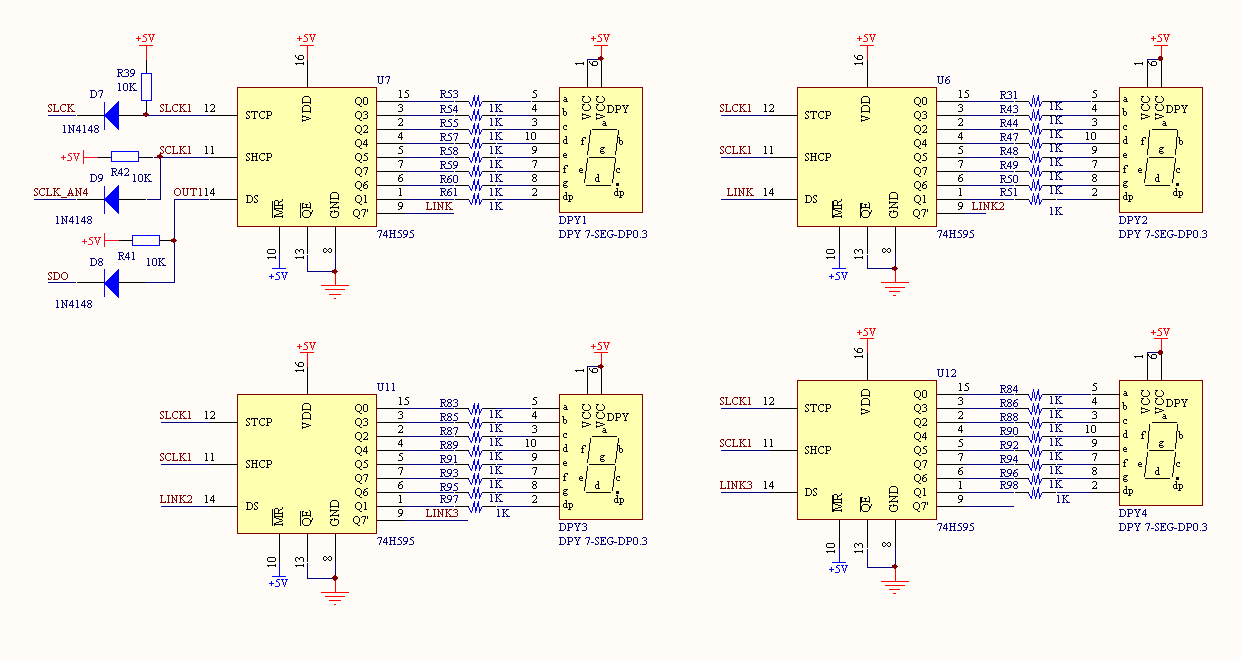


图1 七段数码显示

图1： LED七段数码管驱动电路

#### 开关量输入按键电路电位变化中断编程

在硬件系统中，通过按键开通或闭合来产生高或低电平，从而实现控制信号的键入。如图2所示，采用4.7千欧的电阻与SW-PB限位开关串联的方案设计按键模块，当某按键断开的时候，其相应的输出信号（S1、S2和S3）呈现出高电压；当按键闭合时，输出信号则呈现出低电压。其输出信号直接与PIC32MX输入输出端口相连，提供相应的控制信息。

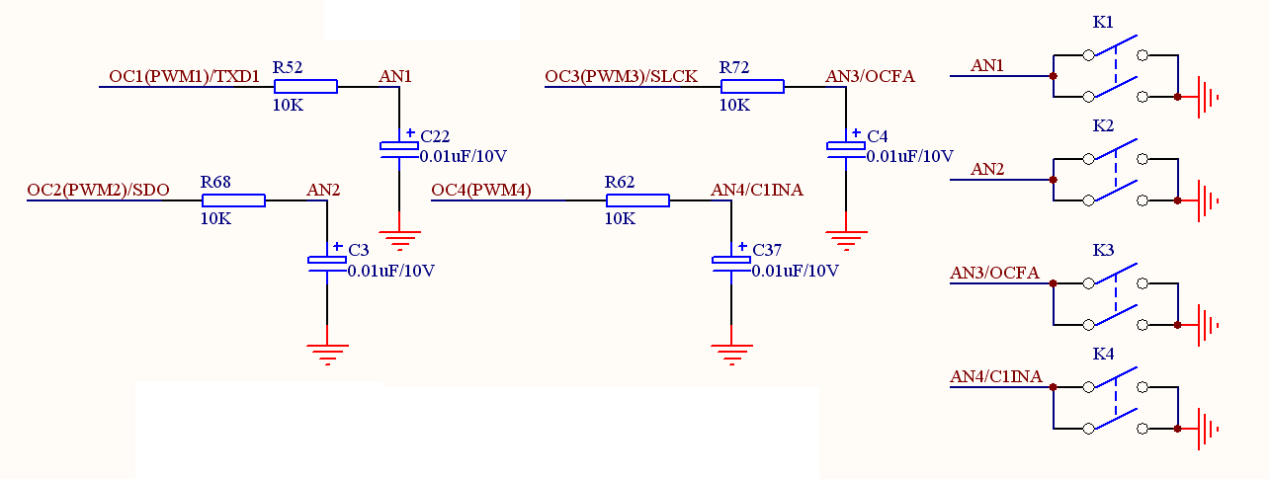


图2： 按键模块

#### RS-232通信模块

RS-232接口电路如图-3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。

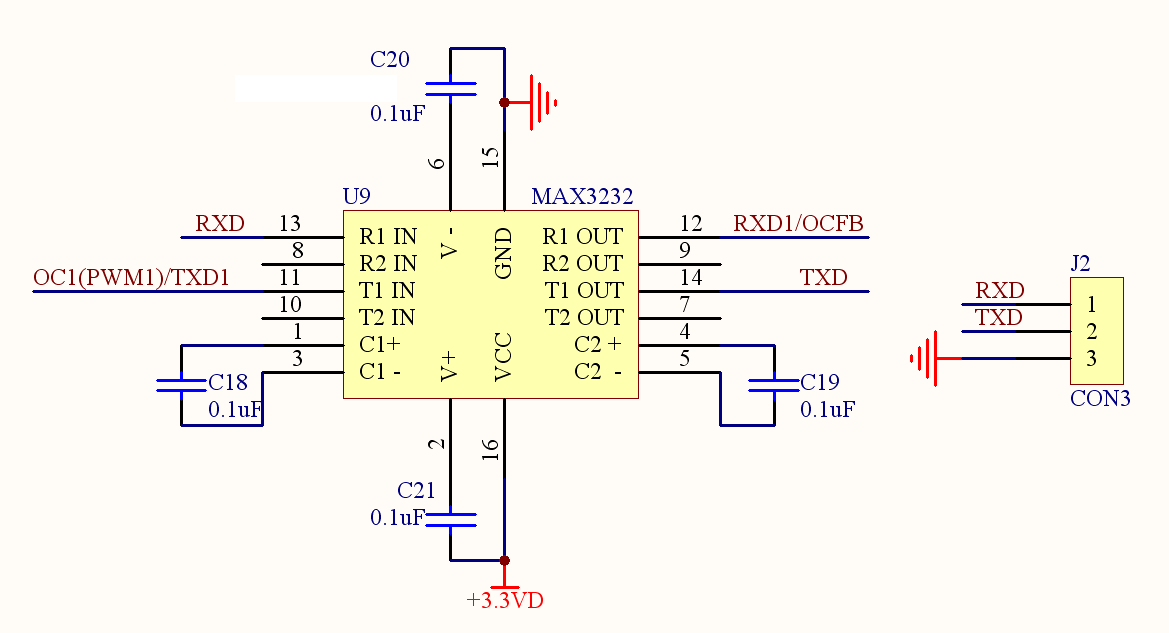


图3： RS-232接口电路图

**程序算法框图**



图4：主函数流程框图

### 各部分代码及算法流程框图

**1、主函数例程**

|  |
| --- |
| 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 5： LED函数流程框图

**2、LED函数例程**

|  |
| --- |
| 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6： Button函数流程框图

**3、Button函数例程**

|  |
| --- |
| 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7： 定时器中断函数流程框图

**4、定时器中断函数例程**

|  |
| --- |
| void \_\_ISR(\_TIMER\_1\_VECTOR, ipl2) **Timer1Handler**(void)  {  // Clear the interrupt flag  INTClearFlag(INT\_T1);  // LED数码管输出计时  led\_cnt++;  if(led\_cnt > 100) //0.1s  {  led\_cnt = 0;  led\_flag = 1;  }  // 按钮扫描计时  btn\_cnt++;  if(btn\_cnt > 5) //5ms  {  btn\_cnt = 0;  btn\_flag = 1;  }  // PWM占空比设置  if(pwm1\_d == 0)  {  pwm1 ++;  if(pwm1 > PWM\_PR)  {  pwm1 = PWM\_PR;  pwm1\_d = 1;  }  }  else  {  if(pwm1 == 0)  {  pwm1 = 0;  pwm1\_d = 0;  }  else  pwm1 --;  }  OC4RS = pwm1;  } |



图8： UART1中断函数流程框图

**5、UART1中断函数例程**

|  |
| --- |
| 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: ComplexExample.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  #define PWM\_PR 0x0FFF //PWM周期  // 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;  UINT16 pwm1=0,pwm1\_d=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  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 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,RPB2); // Assign RPB2 as input pin for U1RX  PPSOutput(1,RPB7,U1TX); // Set RB7 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);  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* PWM初始化  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void **PWMinit**()  {  //PWM引脚关联  RPB13Rbits.RPB13R = 0b0101; //PWM4  //PWM4初始化  OC4CON = 0x0000; // Turn off OC1 while doing setup.  OC4RS = pwm1; // Initialize secondary Compare Register  OC4CON = 0x0006; // Configure for PWM mode  //定时器2周期设定+开启  PR2 = PWM\_PR; // Set period  T2CONSET = 0x8000; // Enable Timer2  //PWM1开启  OC4CONSET = 0x8000; // Enable OC4  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 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();  PWMinit();  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;  }  } |