STM32F10X 重封装库使用示例说明

——望穿秋水

目录

(一)整体说明

1.重封装库基于 STD 标准库，使用时请添加相应的 STD 库。
2.IO 口的使用请查看 STM32F10x_C51Mode.h。 (双击 xx.h 右键->Open document xx.h)
3.stm32 外设的使用请查看 STM32F10xPeripMode.h。
外设硬件及软件资源包括:
(1)16 路外部引脚中断 Px0 - Px15 共 16 个引脚,x=A,B,C,D,E,F,G(引脚组号不能相同)
(2)4 个定时器(定时器 2、3、4 为通用定时器，定时器 1 为高级定时器)
(3)1 路和 2 路互补带死区 PWM 脉宽调制，使用定时器 1，引脚输出-PA8 出或 PA8/PB13
(4)4 路输入通道频率计,输入引脚 CH1-CH4:PA0、PA1、PA2、PA3
(5)3 个串口(printf 输出使用串口 1)，接收中断，1 位停止位，8 位数据位，无奇偶校验
(6)12 位精度 ADC(转换时间 1us)，共 16 路输入:PA0-PA7,PB0,PB1,PC0-PC5
(7)12 位精度 DAC，共 2 路输出:PA4,PA5
(8)内部硬件开门狗。
(9)内部 flash 读写(可用作掉电保存数据,充当 eeprom)
(10)掉电检测中断(掉电检测电压值可设置)
(11)软件层协议 ModBus

(12)软件数据处理算法

(13)外接模块驱动：DS18B20、DS3231、I2C、。

说明：

(1)在 STM32F10xPeripMode.h 中，每个模块的第一句均为一个宏定义，该值为 0 表示不使用该模块，为 1 表示使用该模块，如用到相关模块请将对应模块的宏值改为 1。

(二)示例代码

(0)IO 的使用

IO 操作部分在 STM32F10x_C51Mode.h 中,工程需包含 STM32F10x_C51Mode.h 才能使用。

```
//单个 IO 口操作，点亮 LED
    #define LED0    PA2
    #define LED1    PA3
    void Task1(void)
    {
        PA2_OUT; PA3_OUT; //LED 引脚设置为输出模式
        while(1)
         {
            LED0=1; //LED0 亮
            LED1=1; //LED1 亮
         }
    }
  //多 IO 口操作 （流水灯）
    void Task1(void)
    {
        uint8_t    i=0;
        PA_OUT;   //初使化 PA     16 个 IO 为输出模式
        PA=0x0000;//初使化 PA 16 个 IO 口输出低(LED 高电平亮)
        while(1)
         {
            PA=1<<i;
            i++;
            if(i>15) i=0;
            OS_delayMs(300); //延时 300ms(流水速度)
         }
    }
  //IO 用作独立按键
    #define KEY1          PA0in
    #define KEY2          PA1in
```

```
#define LED             PB5
#define KEY1_IN       PA0_IN
#define KEY2_IN       PA1_IN
#define LED_OUT     PB5_OUT
void Task1(void)
{
   KEY1_IN;   //初使化 KEY1 引脚为输入模式，KEY
   KEY2_IN;   //初使化 KEY2 引脚为输入模式，KEY
   LED_OUT //初使化 LED 引脚为输出模式,，LED
    while(1)
     {
        //按下就动作方式
        if(KEY1==0)          //按键被按下
         {
              delay_ms(40); //延时消抖
              if(KEY1==0)   //消抖后按键还是按下状态说明是真的被按下了
              {
                   LED=1;   //LED 亮
              }
         }
        if(KEY2==0)
         {
              delay_ms(40);
              if(KEY2==0)
              {
                   LED=0;   //LED 灭
                   //LED=~LED;   //取反，每按一次改变一次 LED 状态
              }
         }
        //松手才动作方式
        if(KEY1==0)          //按键被按下
         {
              delay_ms(40); //延时消抖
              if(KEY1==0)   //消抖后按键还是按下状态说明是真的被按下了
              {
                   while(KEY1==0); //等待按键被释放
                   LED=1;   //LED 亮
              }
         }
     }
}
//IO 口和其他芯片通信
 #define   ADD          0x05   //定义地址
 #define   CMD_01       0x09   //定义指令
```

```
#define    CMD_02       0xA4    //定义指令
#define    CLK          PA0     //PA0 用作时钟线
#define    CLK_OUT      PA0_OUT   //时钟线输出模式
#define    SDA          PA1       //PA1 用作数据线输出
#define    SDAin        PA1in     //PA1 用作数据线输入
#define    SDA_OUT      PA1_OUT   //数据线输出模式
#define    SDA_IN       PA1_IN    //数据线输入模式
void Task1(void)
{
    uint8_t    i;
    uint8_t    ReadDat;
    CLK_OUT;     //时钟线初使化为输出模式
    SDA_OUT;    //数据线初使化为输出模式
    while(1)
     {
       CLK=0;   //拉低时钟线，准备发送指令
       for(i=0;i<8;i++)
       {
            SDA=ADD & ( 1<<i ); //发送器件地址(8bit)先发低位    //此处是输出
       }
       SDA_IN; //配置数据线成输入模式
       for(i=0;i<8;i++)
       {
           ReadDat=ReadDat<<1;|//数据移位(必须在前)
           if(SDAin==1)    ReadDat |=0x01;     //此处是读取数据，要用输入
       }
       SDA_OUT; //读取完闭，恢复数据线成输出模式
     }
}
```

以下部分的使用请在工程中包含 STM32F10xPeripMode.h。

(1)外部中断

```
//所有 IO 口均可设置为中断引脚
在 stm32peripmode.h 中把  #define USE_EXTI    0  值改为 1
void Task1(void)
{
    EXTI_Config(GPIOB, GPIO_Pin_2, 0);//初使化 PB2 为外部中断引脚，0-下降沿触发
    while(1)
     {

     }
```

```
}
void EXTI2_IRQHandler(void)      //外部中断函数     引脚 2
{
  if(EXTI_GetITStatus(EXTI_Line2)==SET) //检查中断标志位是否置 1
  {
      EXTI_ClearITPendingBit(EXTI_Line2);//清除 EXTI 线路挂起位
      // 用户添加引脚 2 的中断代码
      /*************************/

        printf("外部中断触发成功~ \r\n ");

      /*************************/
  }
}
```

(2)定时器

```
//TIMx= TIM1 定时器 1，TIM2 定时器 2，TIM3 定时器 3 ，TIM4 定时器 可选

例一：计时或定时
在 stm32peripmode.h 中把 #define USE_TIMER   0 值改为 1
uint8_t   timecount=0,Second=0;   //定义全局变量
void Task1(void)
{
   //方式一：每次计数 1us
   TIMER_Config(TIM2,50000);//初使化定时器 2，定时 50000us=50ms(最大定时 65535us)

   //方式二：每次计数 = 第二个参数 / 72 单位(us)，第二、三参数最大值均为 65535
   TIMER__Config(TIM2,72,50000);//对 72M 主频 72 分频，供给定时器。定时器计数 50000 次。

   TIMER_ON(TIM2);   //打开定时器 2
   while(1)
    {
        if(timecount>=20) //每次中断用时 50ms，20 次中断为 1s
        {
            timecount=0; //计数清 0
            Second++;     //秒变量+1
            printf("秒计时：%d   \r\n",Second); //1 秒更新显示一次秒计时
        }
        if(Second==59)   TIMER_OFF(TIM2);//关闭定时器 2
        // TIMER_ON(TIM2);   //打开定时器 2
    }
}
```

```
void TIM2_IRQHandler(void)    //定时器 2 中断函数 (通用定时器)  //定时时间到，产生中断
 {                                                    //自动重装，无需像 51 单片机要重赋初值
      TIM_ClearITPendingBit(TIM2,TIM_IT_Update);
      // 用户添加定时器 2 的中断代码
      /************************/

       timecount++;  //每次定时器中断让 timecount+1，

      /************************/
}
```

例二：改变 IO 输出频率

```
TIMER_Reload
void Task1(void)
    {
      PB15_OUT; //使用 PB15 输出电平
      TIMER_Config(TIM3,50000);//初使化定时器 3，定时 50000us=50ms(最大定时 65535us)
      While(1)
      {
        TIMER_Reload(TIM3,1000); //改变定时值为 1000us
        OS_delayMs(1000); //延时 1000ms ,1 秒改变一次重装值
      }
   void TIM3_IRQHandler(void)     //定时器 3 中断函数 (通用定时器)  //定时时间到，产生中断
    {                                                    //自动重装，无需像 51 单片机要重赋初值
       TIM_ClearITPendingBit(TIM3,TIM_IT_Update);
       // 用户添加定时器 3 的中断代码
       /************************/

        PB15=~PB15; //IO 翻转(翻转速度由定时器重装值决定)

       /************************/
      }
```

(3)PWM 脉宽调制

```
//单路为 PA8 引脚 ,双路为 PA8 和 PB13 引脚
//3 个参数取值范围均为:0-65535
参数 1：分频系数 1(默认主频为 72000000MHz)，对主频分频后送给定时器
参数 2：分频系数 2，对定时器频率分频后输出
参数 3：占空比
频率范围：72000000/参数 1/65535 --- 72000000/参数 1/1
频率：      72000000/参数 1/参数 2
占空比：    参数 3/参数 2 * 100%
```

在 stm32peripmode.h 中把 #define USE_PWM 0 值改为 1

```
void Task1(void)
{
    PWM_TIM1_Config(72,1000,500);//PWM 频率范围:0.015Hz - 1KHz
                                 //PWM 频率：72MHz/72/1000=1kHz
                                 //PWM 占空比: 500/1000=50%
    PWM_TIM1_ON(); //打开 PWM 输出
    while(1)
     {
        OS_delayMs(5000); //延时 5000ms
        PWM_TIM1_OFF();   //关闭 PWM 输出  //PWM 开关示例,无实际意义
        OS_delayMs(5000); //延时 5000ms
        PWM_TIM1_ON(); //打开 PWM 输出  //PWM 开关示例,无实际意义
     }
}
```

//双路互补带死区输出 (死区计算器在说明-工具文件夹里)

```
 void Task1(void)
 {
    PWM_TIM1_OC_Config(72,1000,500,0x90);//PWM 频率范围:15Hz - 1000KHz
                                         //PWM 频率：72MHz/72/1000=1KHz
                                         //PWM 占空比: 500/1000=50%
                                         //死区时间：通过死区计算器算出
    PWM_TIM1_OC_ON(); //打开 PWM 输出
    While(1)
    {

    }
 }
```

(4)频率计的使用

<1> 输入通道数量值：

通道数量=1 输入引脚=PA0

通道数量=2 输入引脚=PA0 PA1

通道数量=3 输入引脚=PA0 PA1 PA2

通道数量=4 输入引脚=PA0 PA1 PA2 PA3

在 stm32peripmode.h 中把 #define USE_FREQMETER 0 值改为 1

```
void Task1(void)   //任务 1
{
  FreqMeter_TIM2(1,4);   //对 72MHz 主频进行 1 分频，使用 4 个测量输入通道。
  FreqMeter_ON();//打开频率计
  while(1)
  {
  }
```

```
}
```

<2> 对主频分频后的频率供给定时器 2 使用，测量对象频率要远小于 72MHz;
在 STM32PeripMode.h 中可对频率计是否使用串口打印信息进行配置。使用频率计时，定时器 2 不能用作普通定时器。

(5)USART 串口发送和接收

在 stm32peripmode.h 中把 #define USE_USART 0 值改为 1

```
void Task1(void)
{
  uint16_t   Temp; //16 位无符变量 Temp，在被动接收中使用
  char strdat[]={"你好啊~朋友!"}; //示例字符串数组
  uint8_t   Arr[10]={0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09};//示例数组
  USART2_Config(9600); //波特率初使化为 9600
   while(1)
    {
      //发送部分
        USART2_SendByte(0x0F); //使用串口 2 发送一个字节数据
        USART2_SendStr(strdat);//使用串口 2 发送数组字符串数据
        USART2_SendStr("大吉大利，今晚吃鸡~");//使用串口 2 直接发送字符串
        USART2_Send(Arr,8); //使用串口 2 发送 Arr 数组中的前 8 个数据

      //系统有提供发送缓存 Usart2.TX_BUFF[ ]数组，最大 256，使用如下
        Usart2.TX_BUFF[0]=0x24;    //数据可为任意数据
        Usart2.TX_BUFF[1]=0xA2;
        Usart2.TX_BUFF[2]=0xF5;
        Usart2.TX_BUFF[3]=0xE1;
        Usart2.TX_BUFF[4]=0x3B;
        USART2_Send(Usart2.TX_BUFF, 5);

      //主动接收处理
        //假设 0x01 是读数据指令，本设备发送 0x01 给其他设备，会返回 8 个字节的数据
        Usart2.ByteCnt=0;//清空接收缓存,准备接收数据
        USART2_SendByte(0x01);    //发送一个读取数据指令
        while(Usart2.ByteCnt<8); //等待接收 8 个字节完成
        if(Usart2.RX_BUFF[0]==0x02) { Fun1(); } //数据处理   Fun()为功能函数
        if(Usart2.RX_BUFF[1]==0x07) { Fun2(); } //数据处理
          ......
        if(Usart2.RX_BUFF[7]==0xF3) {Fun8(); } //数据处理
       //或用 switch()实现功能跳转
       switch(Usart2.RX_BUFF[2]) //一般中间字节为目标数据，其他为辅助字节
       {
          case 0x00:    Fun1();              break;
```

```
            case 0x01:      Fun2();              break;
              ......
            case 0xFF:      Fun256();          break;
          }

        //被动接收处理 (假如其他设备不定时发来数据，有起始码 8bit+数据码 16bit
                          //+校验码 16bit+结束码 8bit)
        if(Usart2.ByteCnt!=0) //如果有接收
        {
            if(Usart2.RX_BUFF[0]==0x23) //判断起始码(假设起始码为 0x23)
            {
                {
                    //两个 8bit 转换为 16bit
                    Temp=(Usart2.RX_BUFF[1]<<8)|Usart2.RX_BUFF[2];
                    switch(Temp)
                    {
                          case 0x0000:    Fun1();    break;
                          case 0x0001:    Fun2();    break;
                            .....
                          case 0x00FF:    Fun256();    break;
                    }
                    Usart2.ByteCnt=0;   //清除接收，准备下一次接收
                }
            }
            else    Usart2.ByteCnt=0; //第一个数据非起始码，则清除缓存,重新接收
          }
       }
}
void USART2_IRQHandler(void)    //串口 2 中断函数
{
   if(USART_GetFlagStatus(USART2, USART_FLAG_PE) != RESET)
    {
         USART_ReceiveData(USART2);
         USART_ClearFlag(USART2, USART_FLAG_PE);
   }
   if(USART_GetFlagStatus(USART2, USART_FLAG_ORE) != RESET)
   {
        USART_ReceiveData(USART2);
        USART_ClearFlag(USART2, USART_FLAG_ORE);
   }
 if(USART_GetFlagStatus(USART2, USART_FLAG_FE) != RESET)
 {
   USART_ReceiveData(USART2);
   USART_ClearFlag(USART2, USART_FLAG_FE);
```

```
    }
    if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
    {
        USART_ClearFlag(USART2, USART_FLAG_RXNE);     //清除接收中断标志
        USART_ClearITPendingBit(USART2, USART_IT_RXNE);
       //以上代码为固定格式，无需理会
      //串口字节接收：
        Usart2.RX_BUFF[Usart2.ByteCnt++]=USART2_ReadByte();
      //  用户添加串口 2 接收中断代码
      /*************************/
      //接收的数据已保存在 Usart2.RX_BUFF[ ]缓存数据中，一般此处用户不用写，
      //只需从缓存数组中读取数据即可
     /*************************/
    }
}
```

(6)ADC 的使用

```
// ADC 输入引脚有：PA0-PA7,PB0,PB1,PC0-PC5 共 16 个
在 stm32peripmode.h 中把  #define USE_ADC      0  值改为 1
void Task1(void)
{
    float a,b;
    uint16_t c;
    ADC_Config(GPIOA,GPIO_Pin_0); //使用 PA0 作为 ADC 输入引脚
    ADC_Config(GPIOA,GPIO_Pin_4); //使用 PA4 作为 ADC 输入引脚
    while(1)
     {
          a=ReadADC_Volt(GPIOA,GPIO_Pin_0); //读取 PA0 的电压
          b=ReadADC_Volt(GPIOA,GPIO_Pin_4);//读取 PA4 的电压
          c=ReadADC_Hex(GPIOA,GPIO_Pin_4);//读取 PA4 的电压转换值(整型)
          printf("ad1 电压  = %0.4f \r\n",a);    //串口显示电压
          printf("ad2 电压  = %0.4f \r\n",b); //浮点电压值范围：0.0000-3.3000
          printf("ad2 转换值  = %d \r\n",c); //整型值范围:0-4095
          printf(" \r\n");
          OS_delayMs(1000); //延时 1000ms
     }
}
```

(7)DAC 的使用

```
// DAC 输入引脚有：PA4，PA5 共 2 个
```

在 stm32peripmode.h 中把 #define USE_DAC 0 值改为 1

```
void Task1(void)
{
    float a=2.700,b=1.245;
    uint16_t c=1200;
    DAC_Config(GPIOA,GPIO_Pin_4); //使用 PA4 作为 DAC 输出引脚
    DAC_Config(GPIOA,GPIO_Pin_5); //使用 PA5 作为 DAC 输出引脚
    while(1)
     {
            WriteDAC_Volt (GPIOA,GPIO_Pin_4,a); //写入 PA4 要输出的电压 2.700V
            WriteDAC_Volt (GPIOA,GPIO_Pin_5,b);//写入 PA5 要输出的电压 1.245V
            WriteDAC_Hex (GPIOA,GPIO_Pin_5,c);// 写入 PA5 的电压值(整型) 1200
            printf("dac1 电压 = %0.4f \r\n",a);  //显示电压,范围：0.0000-3.3000
            printf("dac2 电压 = %0.4f \r\n",b); //显示电压
            printf("dac2 电压整型值 = %d \r\n",c); //整型值范围:0-4095
            printf(" \r\n");
            OS_delayMs(1000); //延时 1000ms
     }
}
```

(8)程序看门狗

在 stm32peripmode.h 中把 #define USE_IWDG 0 值改为 1

```
void Task1(void)   //任务 1
{
      printf("-------------MCU    Reset------------ \r\n");
      IWDG_Config(1000); //看门狗超时时间设为 1000ms
      while(1)
       {
          printf("Task1 is running\r\n");
          OS_delayMs(500);   //延时 500ms，当延时改为 2000ms 时，可以看到程序不断重启。
          IWDG_Feed();   //喂狗
       }
}
```

在实际工程中使用看门狗时，其初使化建议放在 main 中。

(9)ModBus 主站

//ModBus 的固定指令码需要自行查看相关手册说明

在 stm32peripmode.h 中把 #define USE_MODEBUS 0 值改为 1

modbusRTU SW01_ON=

```
 {
    .SlaveID=0x01, //从机站号
    .Function=0x05, //功能码                  (05 为单个线圈写操作)
    .Address=0x0001, //数据地址              (01 号继电器)
    .Word=0xFF00        //数据                (FF00 为吸合)
 };//CRC 校验会自动生成，无需写入。示例代码，使用时删除
modbusRTU SW01_OFF=
{
   .SlaveID=0x01,
   .Function=0x05,
   .Address=0x0001,                  //(01 号继电器)
   .Word=0x0000                      // (0000 为断开)
};//CRC 校验会自动生成，无需写入。示例代码，使用时删除

   void Task1(void)   //任务 1
   {
       ModBusRTU_Config(); //ModBus 初使化
       uint8_t Delay01_ON[]={0x01,0x05,0x00,0x01,0xFF,0x00}; //数组方式   (无需填入 CRC 校验值)
       uint8_t Delay01_OFF[]={0x01,0x05,0x00,0x01,0x00,0x00};//数组方式  (无需填入 CRC 校验值)
       //以上数组可以单独保存到另一个.c 文件或.h 文件中，使用时调用即可。
       while(1)
       {
           //发送命令方式 1     使用内部结构体  (此方式内存开销较大)
           ModBusRTU_Send(SW01_ON);   //发送 01 号继电器吸合指令
           OS_delayMs(2000); //延时 2 秒
           ModBusRTU_Send(SW01_OFF);   //发送 01 号继电器断开指令
           OS_delayMs(2000); //延时 2 秒

           //发送命令方式 2     使用自定义数组(建议使用此方式，会减小内存开销)
          ModBusRTU_SendCmd(Delay01_ON,6); //函数会自动计算和发送 CRC 校验值
          OS_delayMs(2000); //延时 2 秒
          ModBusRTU_SendCmd(Delay01_OFF,6);//函数会自动计算和发送 CRC 校验值
          OS_delayMs(2000); //延时 2 秒
       }
   }
```