阅读排行榜

推荐排行榜

最新评论

1. Re:ESA2GJK1DH1K基础篇: 阿里云物联网平台: 使用阿里云物联网平台提供的自定义Topic通信控制(ESP8266,TCP透传指令)
代码在哪里下载？

--题哦咯

串口1

```
void uart_init(u32 bound1){
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|

    //USART1_TX   GPIOA.9
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    //USART1_RX   GPIOA.10初始化
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    //USART 初始化设置
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;//字长
    USART_InitStructure.USART_StopBits = USART_StopBits_1;//一个停止位
    USART_InitStructure.USART_Parity = USART_Parity_No;//无奇偶校验位
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlo
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx; /

    USART_InitStructure.USART_BaudRate = bound1;//串口波特率
    USART_Init(USART1, &USART_InitStructure); //初始化串口1

    /*中断优先级配置*/
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=3;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure); //根据指定的参数初始化VIC寄存器
```

2. Re:ESP8266 SDK开发: 综合篇-C#上位机串口通信控制ESP8266
认真拜阅读您的程序 收获很大 ;但我发现一处问题 就是您在串口接收函数中 没有将空闲标志清零 导致程序最终并没有通过空闲中断来处理 而是每隔10ms处理一次

--觉代疯骚

```
    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);//开启串口接受中断

    USART_Cmd(USART1, ENABLE);                    //使能串口
}



//串口中断服务程序
__attribute__((interrupt("WCH-Interrupt-fast")))
void USART1_IRQHandler(void)
{
    u8 Res;
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
    {
        Res =USART_ReceiveData(USART1); //读取接收到的数据
    }
}
```

串口发送数据

```
void usart_send_bytes(USART_TypeDef *USARTx, char *c,uint32_t cnt)
{
    while(cnt--)
    {
        USART_SendData(USARTx, *c++);
        while(USART_GetFlagStatus(USARTx, USART_FLAG_TXE) == RESET );
    }
}


usart_send_bytes(USART1, "11223344", 4);//发送数据
```

串口1,2,3

```
void uart_init(u32 bound1,u32 bound2,u32 bound3){
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2|RCC_APB1Periph_USART
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|

    //串口引脚
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
```

```
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    //串口引脚
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    //串口引脚
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    //USART 初始化设置
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;//字长
    USART_InitStructure.USART_StopBits = USART_StopBits_1;//一个停止位
    USART_InitStructure.USART_Parity = USART_Parity_No;//无奇偶校验位
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlo
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx; /

    USART_InitStructure.USART_BaudRate = bound1;//串口波特率
    USART_Init(USART1, &USART_InitStructure); //初始化串口1

    USART_InitStructure.USART_BaudRate = bound2;//串口波特率
    USART_Init(USART2, &USART_InitStructure); //初始化串口2

    USART_InitStructure.USART_BaudRate = bound3;//串口波特率
    USART_Init(USART3, &USART_InitStructure); //初始化串口3

    /*串口--1*/
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=3;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure); //根据指定的参数初始化VIC寄存器

    /*串口--2*/
    NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure); //根据指定的参数初始化VIC寄存器

    /*串口--3*/
    NVIC_InitStructure.NVIC_IRQChannel = USART3_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);//开启串口接受中断
    USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);//开启串口接受中断
    USART_ITConfig(USART3, USART_IT_RXNE, ENABLE);//开启串口接受中断
```

```
    USART_Cmd(USART1, ENABLE);                    //使能串口
    USART_Cmd(USART2, ENABLE);                    //使能串口
    USART_Cmd(USART3, ENABLE);                    //使能串口
}


//串口中断服务程序
__attribute__((interrupt("WCH-Interrupt-fast")))
void USART1_IRQHandler(void)
{
    u8 Res;
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
    {
        Res =USART_ReceiveData(USART1); //读取接收到的数据
    }
}

//串口中断服务程序
__attribute__((interrupt("WCH-Interrupt-fast")))
void USART2_IRQHandler(void)
{
    u8 Res;
    if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
    {
        Res =USART_ReceiveData(USART2); //读取接收到的数据
    }
}

//串口中断服务程序
__attribute__((interrupt("WCH-Interrupt-fast")))
void USART3_IRQHandler(void)
{
    u8 Res;
    if(USART_GetITStatus(USART3, USART_IT_RXNE) != RESET)
    {
        Res =USART_ReceiveData(USART3); //读取接收到的数据
    }
}
```

串口4

```
void uart_init(u32 bound4){
    //GPIO端口设置
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2|RCC_APB1Periph_USART
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|

    //串口引脚
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
```

```
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOC, &GPIO_InitStructure);


    //USART 初始化设置
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;//字长
    USART_InitStructure.USART_StopBits = USART_StopBits_1;//一个停止位
    USART_InitStructure.USART_Parity = USART_Parity_No;//无奇偶校验位
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlo
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx; /

    USART_InitStructure.USART_BaudRate = bound4;//串口波特率
    USART_Init(UART4, &USART_InitStructure); //初始化串口

    /*串口--4*/
    NVIC_InitStructure.NVIC_IRQChannel = UART4_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 2;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    USART_ITConfig(UART4, USART_IT_RXNE, ENABLE);//开启串口接受中断

    USART_Cmd(UART4, ENABLE);                    //使能串口
}



//串口中断服务程序
__attribute__((interrupt("WCH-Interrupt-fast")))
void UART4_IRQHandler(void)
{
    u8 Res;
    if(USART_GetITStatus(UART4, USART_IT_RXNE) != RESET)
    {
        Res =USART_ReceiveData(UART4); //读取接收到的数据
        USART_SendData(UART4, Res);//返回接收的数据
    }
}
```

串口5

```
void uart_init(u32 bound5){
    //GPIO端口设置
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2|RCC_APB1Periph_USART
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|

    //串口引脚
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
```

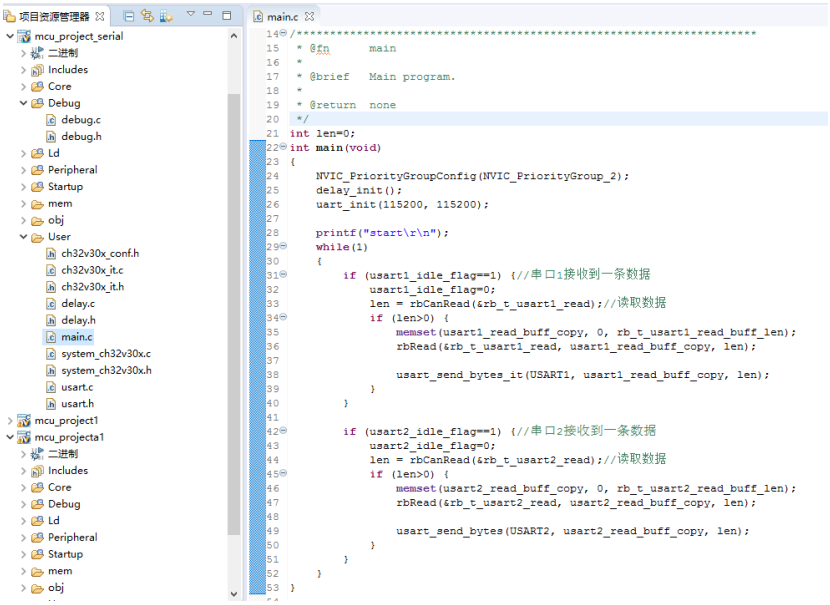
```
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOD, &GPIO_InitStructure);


    //USART 初始化设置
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;//字长
    USART_InitStructure.USART_StopBits = USART_StopBits_1;//一个停止位
    USART_InitStructure.USART_Parity = USART_Parity_No;//无奇偶校验位
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlo
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx; /

    USART_InitStructure.USART_BaudRate = bound5;//串口波特率
    USART_Init(UART5, &USART_InitStructure); //初始化串口

    /*串口--5*/
    NVIC_InitStructure.NVIC_IRQChannel = UART5_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    USART_ITConfig(UART5, USART_IT_RXNE, ENABLE);//开启串口接受中断

    USART_Cmd(UART5, ENABLE);                    //使能串口
}


//串口中断服务程序
__attribute__((interrupt("WCH-Interrupt-fast")))
void UART5_IRQHandler(void)
{
    u8 Res;
    if(USART_GetITStatus(UART5, USART_IT_RXNE) != RESET)
    {
        Res =USART_ReceiveData(UART5); //读取接收到的数据
        USART_SendData(UART5, Res);
    }
}
```

串口6,7,8根据串口4,5修改就可以

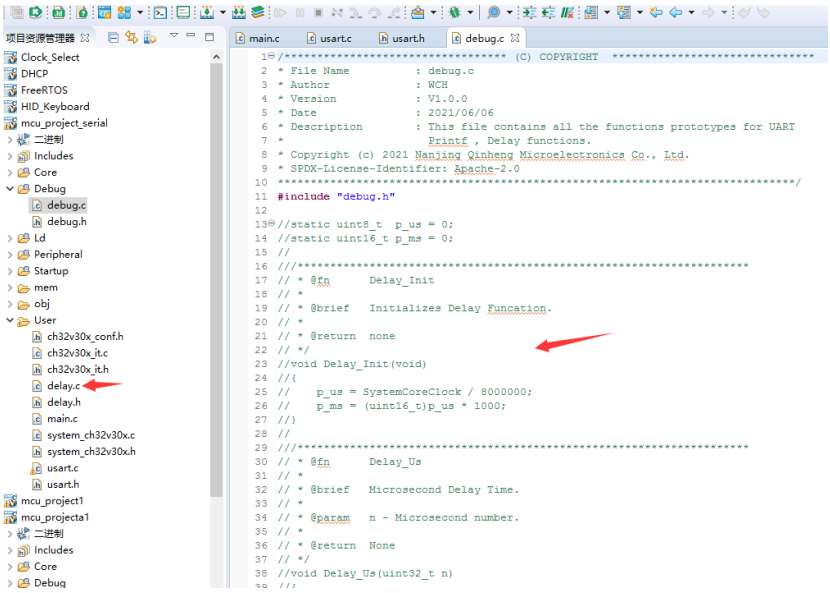
TX , RX
串口6(PC0, PC1)
串口7(PC2, PC3)
串口8(PC4, PC5)

我提供了一套标准的数据处理方案

1,提供的例程是串口1和串口2接收到什么数据就返回什么数据



2,我这边自己实现了延时函数,所以把官方的屏蔽了



main.c | usart.c | usart.h | debug.c | delay.c

```c
#include "usart.h"

volatile int32_t SysTickCntMs=0;

void delay_init(void)
{
    /*配置中断优先级*/
    NVIC_InitTypeDef NVIC_InitStructure = {0};
    NVIC_InitStructure.NVIC_IRQChannel = SysTick_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;//抢占式优先级
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;//响应式优先级
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;//使能
    NVIC_Init(&NVIC_InitStructure);

    /*配置定时器*/
    SysTick->CTLR= 0;
    SysTick->SR  = 0;
    SysTick->CNT = 0;
    SysTick->CMP = SystemCoreClock/1000;//后面的1000代表1000Hz(那就是1ms进一次中断)
    SysTick->CTLR= 0xf;
}

void delay_us(int32_t us)
{
    volatile int64_t ticks;
    ticks = SystemCoreClock / 1000000;
    ticks = ticks * us / 20;
    while(ticks>0)ticks = ticks -1;
}

void delay_ms(int32_t ms)
{
    SysTickCntMs = 0;
    while(SysTickCntMs<ms);
}

__attribute__((interrupt("WCH-Interrupt-fast")))
void SysTick_Handler(void)
{
    SysTick->SR=0;//清除中断
    SysTickCntMs++;

    usart2_idle_loop(50);
}
```

3,printf我这边也改了(中断发送,不会阻塞)

main.c | usart.c | usart.h | debug.c | delay.c

```
235         USART_ClearITPendingBit(USART1,USART_IT_TC);
236         USART_ITConfig(USART1, USART_IT_TC, DISABLE);
237     }
238 }
239
240
241
242 //串口中断服务程序
243 __attribute__((interrupt("WCH-Interrupt-fast")))
244 void USART2_IRQHandler(void)
245 {
246     char data;
247     char socket_id;
248     char Res;
249     if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
250     {
251         Res =USART_ReceiveData(USART2); //读取接收到的数据
252
253         PutData(&rb_t_usart2_read,&Res,1);
254
255         usart2_read_count++;
256
257         /*使用串口1打印串口2接收的数据*/
258         USART_SendData(USART1, Res);
259     }
260 }
261
262 //printf
263 __attribute__((used)) int _write(int fd, char *buf, int size)
264 {
265     usart_send_bytes_it(USART1, buf, size);
266     return size;
267 }
268
```

4,串口1和串口2接收数据都是使用环形队列接收

main.c | usart.c | usart.h | debug.c | delay.c

```
197
198
199 //串口中断服务程序
200 __attribute__((interrupt("WCH-Interrupt-fast")))
201 void USART1_IRQHandler(void)
202 {
203     u8 Res;
204     if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
205     {
206         Res =USART_ReceiveData(USART1); //读取接收到的数据
207
208         PutData(&rb_t_usart1_read,&Res,1);
209         usart1_read_count++;
210     }
211     else if(USART_GetITStatus(USART1,USART_IT_IDLE) == SET)//空闲中断
212     {
213         USART_ReceiveData(USART1);
```

main.c | usart.c | usart.h | debug.c | delay.c

```
242 //串口中断服务程序
243 __attribute__((interrupt("WCH-Interrupt-fast")))
244 void USART2_IRQHandler(void)
245 {
246     char Res;
247     if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
248     {
249         Res =USART_ReceiveData(USART2); //读取接收到的数据
250
251         PutData(&rb_t_usart2_read,&Res,1);
252
253         usart2_read_count++;
254
255         /*使用串口1打印串口2接收的数据*/
256         USART_SendData(USART1, Res);
257     }
258 }
259
```

4,串口1判断接收完一条数据使用的是自带的空闲中断

main.c | usart.c | usart.h | debug.c | delay.c

```
197
198
199 //串口中断服务程序
200 __attribute__((interrupt("WCH-Interrupt-fast")))
201 void USART1_IRQHandler(void)
202 {
203     u8 Res;
204     if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
205     {
206         Res =USART_ReceiveData(USART1); //读取接收到的数据
207
208         PutData(&rb_t_usart1_read,&Res,1);
209         usart1_read_count++;
210     }
211     else if(USART_GetITStatus(USART1,USART_IT_IDLE) == SET)//空闲中断
212     {
213         USART_ReceiveData(USART1);//清除中断
214         usart1_idle_flag = 1;
215     }
216
217     if(USART_GetITStatus(USART1, USART_IT_TXE) != RESET)
218     {
219         if(rbCanRead(&rb_t_usart1_send)>0)//如果里面的数据个数大于0
220         {
221             rbRead(&rb_t_usart1_send,&rb_t_usart1_send_byte,1);
222             USART_SendData(USART1, rb_t_usart1_send_byte);
223         }
224         else
225         {
226             //发送字节结束
227             USART_ClearITPendingBit(USART1,USART_IT_TXE);
228             USART_ITConfig(USART1, USART_IT_TXE, DISABLE);
229             USART_ITConfig(USART1, USART_IT_TC, ENABLE);
230         }
231     }
232     //发送完成
233     if (USART_GetITStatus(USART1, USART_IT_TC) != RESET)
234     {
235         USART_ClearITPendingBit(USART1,USART_IT_TC);
236         USART_ITConfig(USART1, USART_IT_TC, DISABLE);
237     }
238 }
```

5,串口2判断接收完一条数据是使用定时器自定义的空闲时间

main.c | usart.c | usart.h | debug.c | delay.c

```c

/**
 * @brief  串口2自定义空闲中断检测(放到1ms定时器)
 * @param  value: 空闲时间
 * @param  None
 * @param  None
 * @retval None
 * @example
 **/
void usart2_idle_loop(int value){
    if(usart2_read_count!=0){//串口接收到数据
        if(usart2_read_count_copy != usart2_read_count){
            usart2_read_count_copy = usart2_read_count;
            usart2_read_idle_count=0;
        }
        else{
            usart2_read_idle_count ++;
            if(usart2_read_idle_count>value){
                usart2_read_idle_count=0;

                usart2_read_count_copy = usart2_read_count;
                usart2_read_count = 0;
                usart2_idle_flag = 1;//空闲标志
            }
        }
    }
}

```

main.c | usart.c | usart.h | debug.c | delay.c

```c
#include "usart.h"

volatile int32_t SysTickCntMs=0;

void delay_init(void)
{
    /*配置中断优先级*/
    NVIC_InitTypeDef NVIC_InitStructure = {0};
    NVIC_InitStructure.NVIC_IRQChannel = SysTick_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;//抢占式优先级
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;//响应式优先级
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;//使能
    NVIC_Init(&NVIC_InitStructure);

    /*配置定时器*/
    SysTick->CTLR= 0;
    SysTick->SR  = 0;
    SysTick->CNT = 0;
    SysTick->CMP = SystemCoreClock/1000;//后面的1000代表1000Hz(那就是1ms进一次中断)
    SysTick->CTLR= 0xf;
}

void delay_us(int32_t us)
{
    volatile int64_t ticks;
    ticks = SystemCoreClock / 1000000;
    ticks = ticks * us / 20;
    while(ticks>0)ticks = ticks -1;
}

void delay_ms(int32_t ms)
{
    SysTickCntMs = 0;
    while(SysTickCntMs<ms);
}

__attribute__((interrupt("WCH-Interrupt-fast")))
void SysTick_Handler(void)
{
    SysTick->SR=0;//清除中断
    SysTickCntMs++;

    usart2_idle_loop(50);
}
```

6,串口1发送数据可以使用中断方式

main.c | usart.c | usart.h | debug.c | delay.c

```c

/**
 * @brief  串口中断发送数据
 * @param  c:数据的首地址 cnt:发送的数据个数
 * @param  None
 * @param  None
 * @retval None
 * @example
 **/
void usart_send_bytes_it(USART_TypeDef *USARTx, char *c, uint32_t cnt)
{
    if (USARTx == USART1)
    {
        PutData(&rb_t_usart1_send, c, cnt);
        USART_ITConfig(USARTx, USART_IT_TXE, ENABLE);
    }
}


/**
 * @brief  串口发送数据
 * @param  *c:发送的数据指针 cnt:数据个数
 * @param  None
 * @param  None
 * @retval None
 * @example
 **/
void usart_send_bytes(USART_TypeDef *USARTx, char *c,uint32_t cnt)
{
    if (USARTx != USART1) {
        usart_send_bytes_it(USART1, c, cnt);//打印日志
    }

    while(cnt--)
    {
        USART_SendData(USARTx, *c++);
        while(USART_GetFlagStatus(USARTx, USART_FLAG_TXE) == RESET );
    }
}
```

分类: CH32V307(WCH单片机)学习开发

杨奉武
关注 - 1
粉丝 - 779

0 0

« 上一篇： 102-CH32V307(WCH单片机)学习开发-系统滴答定时器

posted on 2022-05-19 17:24 杨奉武 阅读(0) 评论(0) 编辑 收藏 举报

【推荐】大学生技术公益开发训练营，让你的应用为公益发光发热
【推荐】阿里云数据库训练营，云数据库 MySQL 从入门到高阶

编辑推荐：
· 由C# dynamic是否装箱引发的思考
· 闲置树莓派：种朵花然后做延时摄影吧
· 理解 ASP.NET Core - 发送 Http 请求(HttpClient)
· 探索 ABP 基础架构
· 万字长文深度剖析 RocketMQ 设计原理

最新新闻：
· 美的回应裁员传闻：鉴于对内外部环境的判断 有序收缩非核心业务
· 科学家破解卵子起源糖尿病代际遗传之谜
· 谁拥有爱因斯坦的肖像权？
· Gitee 开源库将先审再上线
· 研究显示 CEO 级别高管的密码选择很随意
» 更多新闻...

历史上的今天：
2020-05-19 阿里云物联网平台: 使用阿里云物联网平台提供的物理模型Topic通信控制...
2020-05-19 阿里云物联网平台: 使用阿里云物联网平台提供的自定义Topic通信控制(A...
2017-05-19 1-LPC1778建立工程

单片机,物联网,上位机,…

扫一扫二维码，加入群聊。