STM32F407 UART4串口使用DMA接收不定长数据和DMA中断发送

一、前言

使用DMA通信的好处是，不占用单片机资源（不像普通串口中断，发送一个字节触发一次中断，发送100个字节触发100次中断；接收一个字节触发一次中断，接收200个字节触发20断 ；发送数据完毕触发一次DMA中断。

下图是STM32F407 单片机DMA通道关系图。

表 35. DMA1 请求映射

外设请求	数据流 0	数据流 1	数据流 2	数据流 3	数据流 4	数据流 5	数据流 6	数据流 7
通道 0	SPI3_RX		SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX		SPI3_TX
通道 1	I2C1_RX		TIM7_UP		TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
通道 2	TIM4_CH1		I2S3_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S3_EXT_TX	TIM4_UP	TIM4_CH3
通道 3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
通道 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
通道 5	UART8_TX(1)	UART7_TX(1)	TIM3_CH4 TIM3_UP	UART7_RX(1)	TIM3_CH1 TIM3_TRIG	TIM3_CH2	UART8_RX(1)	TIM3_CH3
通道 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2		TIM5_UP	
通道 7		TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX

1. 这些请求仅在 STM32F42xxx 和 STM32F43xxx 上可用。

表 36. DMA2 请求映射

外设请求	数据流 0	数据流 1	数据流 2	数据流 3	数据流 4	数据流 5	数据流 6	数据流 7
通道 0	ADC1		TIM8_CH1 TIM8_CH2 TIM8_CH3		ADC1		TIM1_CH1 TIM1_CH2 TIM1_CH3	
通道 1		DCMI	ADC2	ADC2		SPI6_TX(1)	SPI6_RX(1)	DCMI
通道 2	ADC3	ADC3		SPI5_RX(1)	SPI5_TX(1)	CRYP_OUT	CRYP_IN	HASH_IN
通道 3	SPI1_RX		SPI1_RX	SPI1_TX		SPI1_TX		
通道 4	SPI4_RX(1)	SPI4_TX(1)	USART1_RX	SDIO		USART1_RX	SDIO	USART1_TX
通道 5		USART6_RX	USART6_RX	SPI4_RX(1)	SPI4_TX(1)		USART6_TX	USART6_TX
通道 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
通道 7		TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	SPI5_RX(1)	SPI5_TX(1)	TIM8_CH4 TIM8_TRIG TIM8_COM

1. 这些请求在 STM32F42xxx 和 STM32F43xxx 上可用。


```
#define UART4_DMA_RX_BUFFER_MAX_LENGTH      (255)
#define UART4_DMA_TX_BUFFER_MAX_LENGTH      (255)
uint8_t UART4_DMA_RX_Buffer[UART4_DMA_RX_BUFFER_MAX_LENGTH];
uint8_t UART4_DMA_TX_Buffer[UART4_DMA_TX_BUFFER_MAX_LENGTH];
```

1、UART4 TX DMA初始化程序

```
void UART4_DMA_Tx_Configuration(void)
{
    DMA_InitTypeDef  DMA_InitStructure;


    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1 , ENABLE);                    //DMA1时钟使能
    DMA_DeInit(DMA1_Stream4);
    while (DMA_GetCmdStatus(DMA1_Stream4) != DISABLE);                       //等待DMA可配置
    DMA_InitStructure.DMA_Channel = DMA_Channel_4;                           //DMA通道配置
    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&UART4->DR;         //DMA外设地址
    DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)UART4_DMA_TX_Buffer;   //发送缓存指针
    DMA_InitStructure.DMA_DIR = DMA_DIR_MemoryToPeripheral;                  //DMA传输方向：内存--->外设
    DMA_InitStructure.DMA_BufferSize = UART4_DMA_TX_BUFFER_MAX_LENGTH;       //数据传输字节数量
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;         //外设非增量模式
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;                  //存储器增量模式
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte; //外设数据长度:8位
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;          //存储器数据长度:8位
    DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;                            //使用普通模式
    DMA_InitStructure.DMA_Priority = DMA_Priority_Medium;                    //中等优先级 DMA_Priority_High
    DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable;
    DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_Full;
    DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;              //存储器突发单次传输
    DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;      //外设突发单次传输
    DMA_Init(DMA1_Stream4, &DMA_InitStructure);                              //初始化DMA Stream
```

```
    DMA_Cmd(DMA1_Stream4, DISABLE);                                 //开启DMA传输
}
```

2、UART4 RX DMA初始化程序

```
void UART4_DMA_Rx_Configuration(void)
{
    DMA_InitTypeDef  DMA_InitStructure;


    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1 , ENABLE);                   //DMA1时钟使能
    DMA_DeInit(DMA1_Stream2);
    while (DMA_GetCmdStatus(DMA1_Stream2) != DISABLE);                      //等待DMA可配置
    DMA_InitStructure.DMA_Channel = DMA_Channel_4;                          //通道选择
    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&UART4->DR;        //DMA外设地址
    DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)UART4_DMA_RX_Buffer;  //接收缓存指针
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory ;                //DMA传输方向：外设到存储器模式：外设--->内存
    DMA_InitStructure.DMA_BufferSize = UART4_DMA_RX_BUFFER_MAX_LENGTH;      //缓冲大小
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;        //外设非增量模式
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;                 //存储器增量模式
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte; //外设数据长度:8位
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;         //存储器数据长度:8位
    DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;                           //使用普通模式
    DMA_InitStructure.DMA_Priority = DMA_Priority_Medium;                   //中等优先级 DMA_Priority_VeryHigh
    DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable;
    DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_Full;
    DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;             //存储器突发单次传输
    DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;     //外设突发单次传输
    DMA_Init(DMA1_Stream2 , &DMA_InitStructure);                            //初始化DMA_Stream
    DMA_Cmd(DMA1_Stream2, ENABLE);                                          //开启DMA传输
}
```

3、UART4 启动DMA发送初始化程序

```
void UART4_DMA_Begin_Send(uint8_t *send_buffer , uint16_t nSendCount)
{
    GPIO_UART4_RS485_SEND_enable();
    if (nSendCount < UART4_DMA_TX_BUFFER_MAX_LENGTH)
    {
        memcpy(UART4_DMA_TX_Buffer , send_buffer , nSendCount);
        DMA_Cmd(DMA1_Stream4 , DISABLE);                    //关闭DMA传输
        while (DMA_GetCmdStatus(DMA1_Stream4) != DISABLE);  //确保DMA可以被设置
        DMA_SetCurrDataCounter(DMA1_Stream4 , nSendCount);  //数据传输量
        DMA_Cmd(DMA1_Stream4 , ENABLE);                     //开启DMA传输
    }
}
```

4、UART4 DMA方式端口初始化程序（包含DMA配置）

```
void UART4_Configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;


    USART_DeInit(UART4);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_UART4 , ENABLE);      //for USART2, USART3, UART4 or UART5.
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

    GPIO_PinAFConfig(GPIOA, GPIO_PinSource0, GPIO_AF_UART4);
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource1, GPIO_AF_UART4);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    USART_InitStructure.USART_BaudRate = 115200;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No ;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(UART4, &USART_InitStructure);
    USART_Cmd(UART4, ENABLE);

    USART_ClearFlag(UART4, USART_FLAG_TC); //清除发送完成标志
    while (USART_GetFlagStatus(UART4, USART_FLAG_TC) == RESET); //等待空闲帧发送完成后再清零发送完成标志（警告：增加这条在蓝牙的硬件框架下会导致死机，原因是前面只使能了USART
    USART_ClearFlag(UART4, USART_FLAG_TC); //清除发送完成标志
```

```
    USART_ITConfig(UART4, USART_IT_RXNE, DISABLE);              //禁止USART1接收不为空中断
    USART_ITConfig(UART4, USART_IT_TXE, DISABLE);               //禁止USART1发送空中断
    USART_ITConfig(UART4, USART_IT_IDLE, ENABLE);               //开启USART1空闲中断
    USART_ITConfig(UART4, USART_IT_TC, DISABLE);                //禁止USART1传输完成中断

    USART_DMACmd(UART4 ,   USART_DMAReq_Tx,DISABLE);            //禁止串口的DMA发送
    USART_DMACmd(UART4 ,   USART_DMAReq_Rx,ENABLE);             //使能串口的DMA接收
}
```

5、UART4 DMA中断接收和DMA中断发送

```

void UART4_IRQHandler(void)
{
    int16_t ch;


    if (USART_GetITStatus(UART4 , USART_IT_IDLE) != RESET)
    {
        USART_ClearITPendingBit(UART4 , USART_IT_IDLE); //必须先清除总线空闲中断标识，然后读一下数据寄存器，DMA接收才会正确（先读SR，然后读DR才能清除空闲中断标识）注意：这句
        ch =  USART_ReceiveData(UART4);                 //必须先清除总线空闲中断标识，然后读一下数据寄存器，DMA接收才会正确（先读SR，然后读DR才能清除空闲中断标识）注意：这句

        #ifdef __DEBUG_stm32f407__
            __DEBUG_UART4_IT_IDLE++;
        #endif

        DMA_Cmd(DMA1_Stream2, DISABLE);                     //关闭DMA,防止处理其间有数据
        DMA_ClearFlag(DMA1_Stream2 , DMA_FLAG_TCIF2 | DMA_FLAG_FEIF2 | DMA_FLAG_DMEIF2 | DMA_FLAG_TEIF2 | DMA_FLAG_HTIF2);//清零标志位
        ch = UART4_DMA_RX_BUFFER_MAX_LENGTH - DMA_GetCurrDataCounter(DMA1_Stream2);
        if (ch > 0)
        {
            //UART4_Outtime_mark = TRUE;
            UART4_receCount = ch;
            //memcpy(UART4_mscomm_buffer , UART4_DMA_RX_Buffer , UART4_receCount);
            WriteBufferTo_ringBuffer(ring , UART4_DMA_RX_Buffer , UART4_receCount);
        }
        DMA_SetCurrDataCounter(DMA1_Stream2 , UART4_DMA_RX_BUFFER_MAX_LENGTH);
        DMA_Cmd(DMA1_Stream2, ENABLE);
    }
    else if (USART_GetITStatus(UART4 , USART_IT_TC)!= RESET)
    {
        USART_ClearITPendingBit(UART4 , USART_IT_TC);

        #ifdef __DEBUG_stm32f407__
            __DEBUG_UART4_IT_TC++;
        #endif

        DMA_ClearFlag(DMA1_Stream4 , DMA_FLAG_TCIF4 | DMA_FLAG_FEIF4 | DMA_FLAG_DMEIF4 | DMA_FLAG_TEIF4 | DMA_FLAG_HTIF4);
        DMA_SetCurrDataCounter(DMA1_Stream4 , 0);   //清除数据长度
    }
}
```

6、主程序

```
void main(void)
{
    UART4_Configuration();
    UART4_DMA_Tx_Configuration();
    UART4_DMA_Rx_Configuration();

    while (1)
    {
          //在合适的时候调用UART4_DMA_Begin_Send(uint8_t *send_buffer , uint16_t nSendBytes)
          // 通过DMA中断方式将数据发送出去
    }
}
```