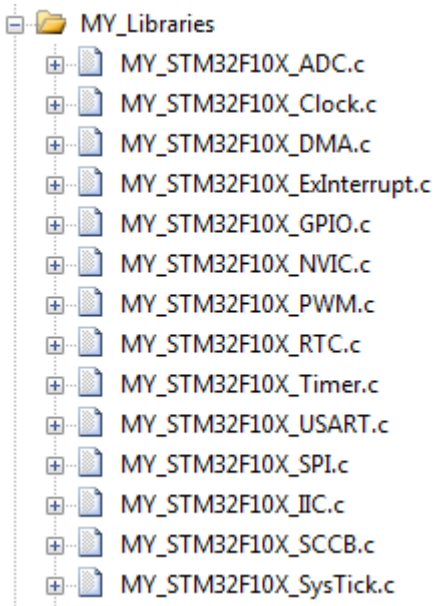
简述：在使用 STM32 的几年时间里，用寄存器为 STM32 编写的库，包括：时钟配置、GPIO 配置、串口配置、外部中断配置、PWM 配置、ADC 配置、DMA 配置，SPI、IIC 的使用等等，几乎包含了所有常用功能。大家可以直接用，新手也可以用来学习，里面的注释很详细。

2016.02.21

我大概从大二上学期，也就是 12 年，开始学 STM32，但是因为整块的时间不多，学习比较分散，进展很缓慢。大概是 13 年寒假我利用寒假时间，结合李想老师的视频开始系统学习 STM32，因为学习时间集中，学习比较系统，所以那段时间进展比较快。

我学完东西比较喜欢做笔记总结，以免以后忘记，所以在学 STM32 的时候做了一些笔记，也顺便编程练一下。后来感觉这样做的效率不是太高，那些笔记以后用到的时候再看并重新编程有点儿麻烦，就索性根据我常用到的 STM32 的功能，用寄存器操作模式为 STM32 编写了一个库，以后用到的时候直接调用，比较方便。

从 13 年寒假开始，我用到 STM32 的什么比较常用的功能后就把那些功能整合到我写的库里面，比如串口配置、外部中断配置、PWM 配置、ADC 配置等等。积累到现在，就成了这个库：



图一

压缩包里有以下两个内容：

图二

“STM32F10X 库文件”文件夹中包含了库的头文件和图一所示的库的源文件，库中每个源文件的功能函数在库的公共头文件“MY_STM32F10X_Conf.h”中都有详细介绍。压缩包中“STM32F10X_Library”文件夹是使用库的示例工程。库的使用很简单，只要把头文件 “MY_STM32F10X_Conf.h”和“stm32f10x.h” 包含到工程里面，再根据需要的功能在工程中添加图一中的源文件，然后在用库的地方，include 库的公共头文件“MY_STM32F10X_Conf.h”，就可以调用库中的函数了。上面提到的文件都可以在“STM32F10X 库文件”文件夹中找到。库中有个别地方会用到 SYSTEM 文件夹中的函数功能，把 SYSTEM 文件夹中的内容也加到工程中就行，SYSTEM 文件夹可以直接从示例工程中拷贝。关于库的使用，可以参考压缩包中的示例工程。

这个库完全是用寄存器编写的，注释很详细，方便对寄存器编程感兴趣的新手参考。对于熟悉 STM32 的老手们，这个库中有很多常用功能可以直接调用，也可以减轻不少编程工作。

希望对大家有用^_^

下面贴出库的公共头文件的截图，里面有对库中的功能函数的详细介绍：

```
/*******************************************库函数使用说明*********************************************
/*
    1，此库适用于STM32F10X系列单片机；
    2，若要使用此库中的某个功能函数，首先要包含此头文件，然后在工程中包含要使用的功能函数
       对应的.c文件；
    3，每个模块使用细节，请看对应使用说明
    4，若要使用此库，请注明作者：hongge
*/

#ifndef wang_MY_STM32F10X_Conf_hongge
#define wang_MY_STM32F10X_Conf_hongge


#include <stm32f10x.h>
#include "sys.h"                                        //用于位段操作，用法见头文件

#define ui unsigned int
#define uc unsigned char
```

```
/*****************************************时钟配置*************************************************
/*功能函数使用说明：
  1，STM32_ClockInit用于系统及APB外设时钟初始化，其中HSE不分频作为PLL输入，PLL输出系统时钟，AHB
    不分频，APB1二分频，APB2不分频。PllMultiply为PLL倍频数。SYSCLK=PLL输入时钟*PllMultiply，
    APB1CLK=SYSCLK/2，APB2CLK=SYSCLK。
  2，STM32_ClockEnable用于使能外设时钟。AXB为要使能时钟的外设所属外设区；RCC_AXBPeriph为要使能
    时钟的外设对应的时钟位宏定义（如：RCC_APB2Periph_AFIO）。
*/
typedef enum
{
  AHB=0,APB1,APB2
}AXB_TypeDef;
void STM32_ClockInit(u8 PllMultiply);
void STM32_ClockEnable(AXB_TypeDef AXB,uint32_t RCC_AXBPeriph);



/*****************************************GPIO操作*************************************************/
/*功能函数使用说明：
  1，STM32_GPIOInit用来初始化GPIO引脚。GPIOx:要初始化的引脚所属GPIO口；GPIOInit_pst:用于初始化引脚的结构
    体（详见stm32f10x_gpio.h）。
  2，STM32_GPIOWriteBit用来对IO口进行按位写操作。GPIOx:要进行位操作的引脚所属GPIO口；GPIO_Pin:要操作的引
    脚（支持或操作）；BitVal:要把引脚设置为的值（Bit_SET或Bit_RESET）。
  3，STM32_GPIOReadBit用来对IO口进行按位读操作。GPIOx:要进行位操作的引脚所属GPIO口；GPIO_Pin:要操作的引脚
    返回值：要读取引脚的电平值（Bit_SET或Bit_RESET）。
  4，STM32_GPIOWriteData用来对IO口进行整体写操作。GPIOx:要进行整体写操作的GPIO口；Data:要写入IO口的值。
  5，STM32_GPIOReadData用来对IO口进行整体读操作。GPIOx:要进行整体读操作的GPIO口。返回值：要读取的GPIO口的
    引脚电平值（16位数）。
*/
void STM32_GPIOInit(GPIO_TypeDef* GPIOx,GPIO_InitTypeDef* GPIOInit_pst);
void STM32_GPIOWriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal);
BitAction STM32_GPIOReadBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void STM32_GPIOWriteData(GPIO_TypeDef* GPIOx,u16 Data);
u16 STM32_GPIOReadData(GPIO_TypeDef* GPIOx);



/*****************************************SysTick*************************************************/
/*功能函数使用说明：
  1，STM32_SysTickInit用于SysTick模块初始化。SystemClk为当前系统时钟，单位为Hz；HeartBeatFreq为SysTick中断频率，
    单位为Hz，取值必须大于0.536441802978515625，取零表示无效，不作处理。
  2，STM32_Delay_us为us延时函数，nus为需要延时的us数。在USE_SYSTICK_INT模式下，延时最大可达477204025us（受中间值的
    数据类型u32位数限制，2^32/9=477204025）；在另一模式下，延时最大可达1864135.111us（受计数器有效位数24限制）。
  3，STM32_Delay_ms为ms延时函数，nms为需要延时的ms数。在USE_SYSTICK_INT模式下，延时最大可达477204.025ms（受中间值的
    数据类型u32位数限制，2^32/9000=477204.025）；在另一模式下，延时最大可达1864.135111ms（受计数器有效位数24限制）。
  4，STM32_MeasureFuncExctTimeBySysTick为函数执行时间测量函数。Function为待测函数名；count为测试次数（取其中最大值）。
    返回值为被测函数的执行时间，单位为us，返回值最大477218588.444444us。经测试，用SysTick的测量方法比用定时器更精确。

注：
  1，使用SysTick中断的话，须处理中断函数，例：void SysTick_Handler(void){if(SysTick->CTRL&(1<<16)){~~~;}}
  2，STM32_MeasureFuncExctTimeBySysTick的使用条件：必须初始化中断，实现中断处理函数，并使FuncExctTimeMeasureIntCount在
    中断中加1，如下：
                //void SysTick_Handler(void)
                //{
                //  if(SysTick->CTRL&(1<<16))           //计数器溢出中断
                //  {
                //    FuncExctTimeMeasureIntCount++;    //不能删
                //  }
                //}
*/
#define USE_SYSTICK_INT                     //使用SysTick中断
//void STM32_DelayInit(u32 SystemClk);



/*****************************************NVIC通用配置*************************************************/
/*功能函数使用说明：
  1，STM32_NVICSetVectorTable用来设置向量表偏移地址。NVIC_VectTab:基址，NVIC_Offset:偏移量；
  2，STM32_NVICInit用来设置中断分组、优先级设置及中断使能。NVIC_Channel为中断通道（中断编号），
    NVIC_Group为中断分组，NVIC_PreemptionPriority为抢占优先级，NVIC_SubPriority为响应优先级；
*/
#define NVIC_GROUP 2
void STM32_NVICSetVectorTable(u32 NVIC_VectTab, u32 NVIC_Offset);     //还没懂，抽时间弄懂
void STM32_NVICInit(u8 NVIC_Channel,u8 NVIC_Group,u8 NVIC_PreemptionPriority,u8 NVIC_SubPriority);

/*****************************************串口通信*************************************************/
/*功能函数使用说明：
  1，STM32_USARTInit用来初始化串口。USARTx为需要初始化的串口(1~3)；PCLK为APB时钟；BaudRate为波特率
  2，Uart_PutChar用来发送字符数据。USARTx为需发送数据的串口；ch为要发送的字符数据
  3，Uart_PutString用来发送字符串数据。USARTx为需发送数据的串口；p为需发送的字符串的首地址；Length为
    需发送的字符串的长度

注意：须处理中断函数，例：void USART1_IRQHandler(void){if(USART1->SR&(1<<5)){~~~;}}
*/

void STM32_USARTInit(USART_TypeDef *USARTx,u32 PCLK,u32 BaudRate);
void STM32_USARTSendChar(USART_TypeDef *USARTx,u8 ch);              //串口x发送字符数据
void STM32_USARTSendString(USART_TypeDef *USARTx,char *p,u16 Length);     //串口x发送字符串数据
```

```
/*******************************************定时器*******************************************/
/*功能函数使用说明：
  1，STM32_TimeInit用来设置定时器中断，设置定时时间
    Timex为定时器选择（TIM2~TIM7），arr为计数器重装值，psc为定时器预分频值。
  2,STM32_TimeInitUs用来设置定时器中断，设置定时时间。Timex为定时器选择（TIM2~TIM7），us为要定时的us数。
  3，STM32_MeasureFuncExctTimeByTimerx用来测试待测函数的执行时间。TIMx:定时器，Function：要测试的函数，
    count:测试的次数（多次测试取执行时间最大值）。返回值为测试函数的执行时间，单位为us，最大
    为4294967296us（受返回值类型u32的位数限制）。经测试，用定时器的测量方法没有用SysTick精确。

注意：
  1，须处理中断函数，例：void TIM2_IRQHandler(void){if(TIM2->SR&0X0001){~~~;}TIM2->SR&=~(1<<0);}
  2，STM32_MeasureFuncExctTimeByTimerx的使用条件：必须实现相应定时器的中断处理函数，并使FunctionTimeMeasureIntCount在
      中断中加1，如下：
                    //void TIMx_IRQHandler(void)
                    //{
                    //  if(TIMx->SR&0X0001)      //溢出中断
                    //  {
                    //    FunctionTimeMeasureIntCount++;
                    //  }
                    //  TIMx->SR&=~(1<<0);       //清除中断标志位
                    //}
*/
void STM32_TimerInit(TIM_TypeDef *TIMx,u16 arr,u16 psc);
void STM32_TimerInitUs(TIM_TypeDef *TIMx,u32 us);
u32 STM32_MeasureFuncExctTimeByTimerx(TIM_TypeDef *TIMx,void(*Function)(void),u8 count);
```

```
/*******************************************AD转换*******************************************/
/*功能函数使用说明：
  1，STM32_ADC1Init用来初始化ADC转换器，ADCChannelx为需要使用AD转换的通道，SimpleTime为AD转换采样时间；
  2，STM32_ADC1WatchDogInit用来初始化看门狗中断。ADCChannelx为看门狗监测通道，HighThreshouldVolt为监测
     的电压上限（单位：v），LowThreshouldVolt为监测的电压下限（单位：v）；
  3，STM32_ADC1StartAConversion用来开始一次AD转换；
  4，STM32_ADC1GetVal用来获取ADCChannelx所对应的通道的电压转换值，返回float类型的电压值。DoNewSimple：0，
     不进行新的采样，直接用寄存器中的值转化为电压；1，进行新的采样，采样、转换结束后，再用寄存器中的值
     转化为电压；

注意：1，VCC为当前ADC参考电压。
      2，对于函数STM32_ADC1GetVal，如果程序中对执行时间要求较高：
        （1）不要使用采样后进行电压转换，等待时间较多（主要是采样+转换时间），可以利用转换结束中断，在中断中读取转换数值，
            以避免等待时间；
        （2）将采样时间降低，缩短此程序的等待时间（这个办法不知道能不能明显减少等待时间，还没测试）。
*/
#define VCC 3.3
typedef enum        //ADC通道定义：0~17，共18个通道
{
  ADCChannel0=0,ADCChannel1,ADCChannel2,ADCChannel3,ADCChannel4,ADCChannel5,ADCChannel6,ADCChannel7,ADCChannel8
  ,ADCChannel9,ADCChannel10,ADCChannel11,ADCChannel12,ADCChannel13,ADCChannel14,ADCChannel15
  ,ADCChannel16,ADCChannel17
}ADCChannelx_TypeDef;
typedef enum        //采样时间定义：1.5，7.5，13.5，28.5，41.5，55.5，71.5，239.5个周期
{
  SimpleTime_1_5=0,SimpleTime_7_5,SimpleTime_13_5,SimpleTime_28_5,SimpleTime_41_5
  ,SimpleTime_55_5,SimpleTime_71_5,SimpleTime_239_5
}SimpleTime_TypeDef;
void STM32_ADC1Init(ADCChannelx_TypeDef ADCChannelx,SimpleTime_TypeDef SimpleTime);   //此处以ADC1，PA口为例
void STM32_ADC1WatchDogInit(ADCChannelx_TypeDef ADCChannelx,float HighThreshouldVolt,float LowThreshouldVolt);
void STM32_ADC1StartAConversion(void);
float STM32_ADC1GetVal(u8 DoNewSimple);
```

```
/*******************************************DMA高速数据传送*******************************************/
/*功能函数使用说明：
  1，STM32_DMAInit用来初始化DMA的使用，DMA_CHx为要使用的DMA通道，PeripheralAddr为外设地址，MemoryAddr为
    存储器地址，DataLength为要传送的数据的长度
  2，STM32_DMAEnableOnce用来开启一次DMA传送，DMA_CHx为要使用的DMA通道

注意：
  1，记得使能外设时钟和外设DMA控制位（初始化好外设）；
  2，开启一次DMA传送后可对数据是否发送完成进行检测：while(1){if(DMA1->ISR&(1<<13)){DMA1->IFCR|=1<<13;break;}}
*/
void STM32_DMAInit(DMA_Channel_TypeDef *DMA_CHx,u32 PeripheralAddr,u32 MemoryAddr,u32 DataLength);  //从存储器到外设
void STM32_DMAEnableOnce(DMA_Channel_TypeDef *DMA_CHx);


/*******************************************RTC实时时钟实验*******************************************/
/*功能函数使用说明：
  1，STM32_RTCInit用来初始化实时时钟

注意：须处理中断函数，例：void RTC_IRQHandler(void){if((RTC->CRL&1<<0)!=0){~~~;}RTC->CRL&=0xfffe;while((RTC->CRL&(1<<5))==0)
*/
u8 STM32_RTCInit(void);
```

```
/*******************************************PWM实验*******************************************/
/*功能函数使用说明：
  1，STM32_TIM4PWMInit用于初始化TIM4对应的PWM。TIMCLK为当前输入时钟频率；Freqence为输出频率，取值为
    1KHz~14MHz（测试所得）；PulseWidth为脉冲宽度，取值为0~100。
  2，STM32_TIMxPWMInit用于初始化TIMx对应的PWM。TIMCLK：当前输入时钟频率；TIMChannelx:定时器通道；
          Freqence:输出频率，取值为1Hz~3.4MHz（测试所得）；DutyRatio:PWM占空比，取值为0~1.0。
          注意：若把Freqence改为浮点类型的话，估计最低频率可以达到0.01Hz。
  3,STM32_TIMxSetPWMDuty用来改变TIMx一个通道的PWM的占空比。
*/
typedef enum
{
  PB6=6,PB7,PB8,PB9
}GPIOBPinx_TypeDef;
void STM32_TIM4PWMInit(u32 TIMCLK,GPIOBPinx_TypeDef GPIOBPinx,u32 Freqence,u8 PulseWidth);  //以TIM4，PB口输出PWM，无重映射为例

typedef enum
{
  TIMChannel1=0,TIMChannel2,TIMChannel3,TIMChannel4
}TIMChannelx_TypeDef;
void STM32_TIMxPWMInit(u32 TIMCLK,TIM_TypeDef* TIMx,TIMChannelx_TypeDef TIMChannelx,u32 Freqence,float DutyRatio);  //定时器输出PWM初始
void STM32_TIMxSetPWMDuty(TIM_TypeDef* TIMx,TIMChannelx_TypeDef TIMChannelx,float DutyRatio); //设置定时器输出的PWM的占空比
```

```
/*********************************************外部中断*********************************************/
/*功能函数使用说明：
  1，STM32_ExInterruptInit用于外部中断初始化。GPIOx为需要设置外部中断的端口号；GPIO_Pin为需要设置外部中断的
    引脚号；NVIC_Group为中断优先级分组，NVIC_PreemptionPriority为抢占优先级，NVIC_SubPriority为响应优先级，
    ExIntWorkMode为外部中断触发方式。

  注意：须处理中断函数，例：void EXTI15_10_IRQHandler(void){STM32_Delay_ms(20);~~~;EXTI->PR=1<<12;}

*/
typedef enum
{
  INTWORKMODE_RISING=1,INTWORKMODE_FALLING=2
}ExIntWorkMode_TypeDef;
void STM32_ExInterruptInit(GPIO_TypeDef *GPIOx,u16 GPIO_Pin,u8 NVIC_Group,u8 NVIC_PreemptionPriority,
        u8 NVIC_SubPriority,ExIntWorkMode_TypeDef ExIntWorkMode);



/*********************************************SPI通信实验*********************************************/
/*功能函数使用说明：
  1，STM32_SPIxInit用于初始化SPI口，SPIx为SPI口。
  2，STM32_SPIxSetSpeed设置SPI波特率分频，SPIx为SPI口，波特率=Fpclk/分频数。
  3，STM32_SPIxReadWriteByte用于通过SPI总线读写一个字节，SPIx为SPI口，TxData为要写到SPI总线上的数据，
    函数返回从SPI总线上读取的数据。
*/
// SPI总线波特率设置
typedef enum
{
  SPIBaudFreqDiv_2=0,SPIBaudFreqDiv_4,SPIBaudFreqDiv_8,SPIBaudFreqDiv_16,
  SPIBaudFreqDiv_32,SPIBaudFreqDiv_64,SPIBaudFreqDiv_128,SPIBaudFreqDiv_256
}SPIBaudFreqDiv_TypeDef;

void STM32_SPIxInit(SPI_TypeDef *SPIx);                                                 //初始化SPI口
void STM32_SPIxSetBaudFreqDiv(SPI_TypeDef *SPIx,SPIBaudFreqDiv_TypeDef SPIBaudFreqDiv);   //设置SPI波特率分频，波特率=Fpclk/分频数
u8 STM32_SPIxReadWriteByte(SPI_TypeDef *SPIx,u8 TxData);                                //SPI总线读写一个字节



/*********************************************IIC通信实验*********************************************/
/*功能函数说明：
  1，STM32_IICInit用于IIC使用初始化。
  2，STM32_IICWriteOneByte用于向IIC从机的寄存器写入一个字节。IICSlaveAddr为IIC从机地址，RegAddr为寄存器地址，
     Byte为要写入的字节。
  3，STM32_IICWriteMultiByte用于向IIC从机的寄存器写入多个字节。IICSlaveAddr为IIC从机地址，RegAddr为寄存器地址，
     pByteWrite为要写入的多个字节的首地址，Count为要写入的字节个数。
  4，STM32_IICReadOneByte用于从IIC从机的寄存器读取一个字节。IICSlaveAddr为IIC从机地址，RegAddr为寄存器地址。返回值
     为读取的字节。
  5，STM32_IICReadMultiByte用于从IIC从机的寄存器读取多个字节。IICSlaveAddr为IIC从机地址，RegAddr为寄存器地址，
     Count为要读取的字节的个数，pByteRead为读取的多个字节的保存位置的首地址。

  注意：此处使用的是模拟IIC。
*/

//IO方向设置
#define SDA_IN()  {GPIOC->CRH&=0XFFFF0FFF;GPIOC->CRH|=8<<12;}
#define SDA_OUT() {GPIOC->CRH&=0XFFFF0FFF;GPIOC->CRH|=3<<12;}

//IO操作BitBand定义
#define IIC_SCL    PCout(12) //SCL
#define IIC_SDA    PCout(11) //SDA
#define READ_SDA   PCin(11)  //输入SDA

//IIC从机地址定义
#define IIC_MPU6050_ADDRESS 0x68    //MPU6050的IIC地址（AD0接地）
#define IIC_HMC5883L_ADDRESS 0x1e   //HMC5883L的IIC地址

void STM32_IICInit(void);
void STM32_IICWriteOneByte(u8 IICSlaveAddr,u8 RegAddr,u8 Byte);
void STM32_IICWriteMultiByte(u8 IICSlaveAddr,u8 RegAddr,u8* pByteWrite,u8 Count);
u8 STM32_IICReadOneByte(u8 IICSlaveAddr,u8 RegAddr);



/*********************************************SCCB通信实验*********************************************/
/*
  注意：此处使用模拟SCCB
*/
#define SCCB_SDA_IN()  {GPIOD->CRH&=0XFF0FFFFF;GPIOD->CRH|=0X00800000;}
#define SCCB_SDA_OUT() {GPIOD->CRH&=0XFF0FFFFF;GPIOD->CRH|=0X00300000;}

//IO操作函数
#define SCCB_SCL         PDout(3)       //SCL
#define SCCB_SDA         PDout(13)      //SDA

#define SCCB_READ_SDA    PDin(13)       //输入SDA
#define SCCB_ID          0X42           //OV7670的ID

//SCCB通信基础函数
void STM32_SCCBInit(void);
void STM32_SCCBStart(void);
void STM32_SCCBStop(void);
void STM32_SCCBNoAck(void);
u8 STM32_SCCBWriteByte(u8 dat);
u8 STM32_SCCBReadByte(void);
u8 STM32_SCCBWriteReg(u8 reg,u8 data);
u8 STM32_SCCBReadReg(u8 reg);
```

-

压缩包内容


STM32F10
X_Library
STM32F10
X库文件
示例工程
库文件