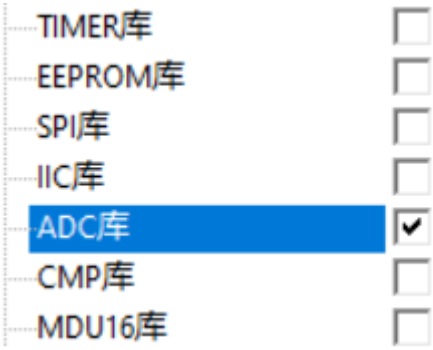
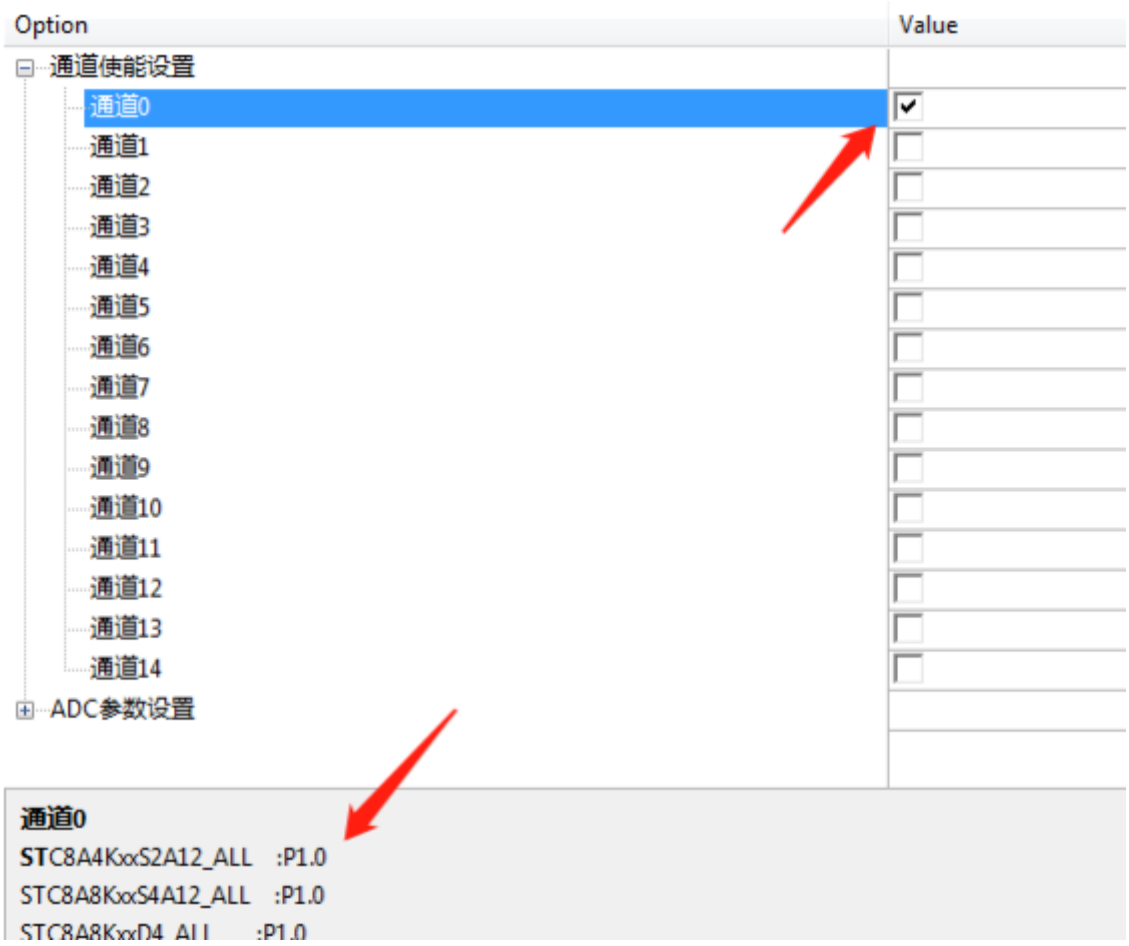
ADC库

ADC就是Analog to Digital Converter，是一种能把模拟的电压信号转换成数字信号的器件。ADC库就是关于单片机模数转换的操作库。在使用本库之前先到ecbm_core.h里使能。双击打开ecbm_core.h文件，然后进入图形化配置界面，使能ADC库。



ADC是单片机获取环境模拟量的重要输入外设，通常都会有多个通道。所以在使用之前还需要在adc.h的图形配置界面使能所需要的通道才能正常读取。



如图，在下面的说明里还能看到该通道在不同型号单片机下对应的引脚。

API

adc_init

函数原型：void adc_init(void);

描述

ADC初始化函数。

输入

无

输出

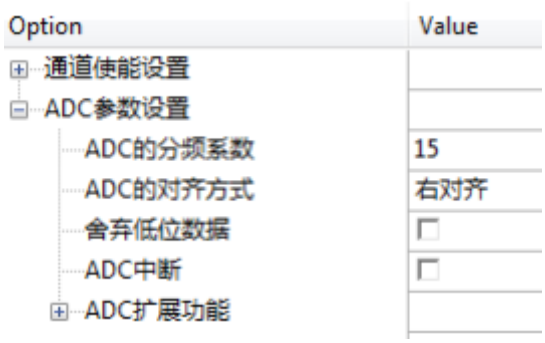
无

返回值

无

参数配置

在图形化配置界面设置的参数，将会在执行本函数的时候写到ADC寄存器中。因此要保证选项选择正确无误。



设置说明如下：

- ADC的分频系数：这个参数决定了ADC的转换速度，但是在应用中发现分频数在6以下的时候，数据会跳动得比较厉害。因此推荐输入7~15。
- ADC的对齐方式：在大于8位小于16位的ADC中，会需要两个寄存器来存放转换好的AD值。不足16位的部分将会补0。以12位为例，若是左对齐，则低4位全为0。若是右对齐，则高4位全为0。他们的效果如下图所示。

寄存器		ADC_RES								ADC_RESL								对应数值
左对齐	分布	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0	
	最小值	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	最大值	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	65520
	步进值	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	16
右对齐	分布	0	0	0	0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
	最小值	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	最大值	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	4095
	步进值	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

- 舍弃低位数据：从上面的设置中，可以看出右对齐才符合我们正常的使用习惯。那么左对齐的用法我猜测是用来滤波的。因为在正常情况下，ADC的数值跳动都集中在低几位中。如果设置了左对齐再舍弃掉ADC_RESL寄存器的值，那么跳动的那几位数据就会被舍弃掉，于是剩下比较平稳的高8位数据。
- ADC中断：使能之后将会打开ADC的中断使能，但是不推荐使用ADC中断。
- ADC扩展功能：目前还没有正式测试，不推荐使用这部分功能。因为不是每一个型号都有这个，有这功能的型号还缺货。

调用例程

```
#include "ecbm_core.h"//加载库函数的头文件。
void main(){//main函数，必须的。
    system_init();//系统初始化函数，也是必须的。
    adc_init();//初始化ADC
    while(1){
    }
}
```

注意事项

1. 本函数将将ADC已使能的通道的对应引脚设置为高阻态。

adc_read

函数原型：u16 adc_read(u8 ch);

描述

读取AD值函数。

输入

- ch：要读取AD值的通道编号，从0开始。

输出

无

返回值

- 该通道的AD值。

调用例程

串口获取单通道：

```
#include "ecbm_core.h"//加载库函数的头文件。
void main(){//main函数，必须的。
    system_init();//系统初始化函数，也是必须的。
    adc_init();//先初始ADC。
    while(1){
        delay_ms(1000);//每秒发送一次AD值到串口。
        debug("%u\r\n",adc_read(0));//发送通道0的AD值。
    }
}
```

串口获取多通道：

```
#include "ecbm_core.h"//加载库函数的头文件。
void main(){//main函数，必须的。
    system_init();//系统初始化函数，也是必须的。
    adc_init();//先初始ADC。
    while(1){
        delay_ms(1000);//每秒发送一次AD值到串口。
        debug("[0]=%u\r\n",adc_read(0));//发送通道0的AD值。
        debug("[2]=%u\r\n",adc_read(2));//发送通道2的AD值。
    }
}
```

注意事项

1. 使用本函数之前，要先初始化ADC。
2. 本函数用到哪个通道，就得先在adc.h里使能哪个通道。否则会读取失败。

adc_voltage

函数原型：float adc_voltage(u8 ch,float vref);

描述

读取电压函数。

输入

- ch：要读取的通道编号，从0开始。
- vref：ADC的Vref引脚电压，单位为伏。

输出

无

返回值

- 该通道的电压值，单位为伏。

调用例程

```
#include "ecbm_core.h"//加载库函数的头文件。
void main(){//main函数，必须的。
    system_init();//系统初始化函数，也是必须的。
    adc_init();//先初始ADC。
    while(1){
        delay_ms(1000);//每秒发送一次AD值到串口。
        debug("%f\r\n",adc_voltage(0,3.10f));//发送通道0的电压值，3.10为Vref引脚的实测电压值，即3.10V。
    }
}
```

注意事项

无

adc_read_vref

函数原型：float adc_read_vref(void);

描述

ADC读取Vref函数。

输入

无

输出

无

返回值

- 单片机Vref的电压值。

小科普

很多人被“基准电压”这个词糊弄到了，就觉得STC内部的1.19V基准电压是Vref，其实这是错误的想法。下面针对几种主流认知一一说明：

- “这个1.19V电压不随着VCC的变化而改变，且可以用来计算AD值，那么1.19V的作用和Vref一样。”这个观点主要是不理解Vref的作用，认为Vref只是一个用来校准电压的东西。实际上Vref的用处不止是校准电压，它还决定了ADC能测量的最大电压。我们可以把ADC想象成一个大型的比较器，被测电压进来先和Vref的256分之1（假设是8位ADC）比较，如果被测电压小于Vref的256分之1就返回0，于是AD值就是0；如果被测电压大于等于Vref的256分之1且小于Vref的256分之2就返回1，于是AD值就是1。如果被测电压大于Vref的256分之2，就按这个规律一直比较下去直到比较出结果。所以说如果1.19V是Vref的话，那基本1.19V以上的电压都测不到了。
- “官方手册都没提到Vref，和电压基准有关的就只有1.19V了，不是它还会是谁？”有这种观点的人，估计深受官方广告手册的毒害。直到STC8的手册出来之前，STC的手册基本和广告传单差不多。所以有些人就没理解STC单片机的结构。首先重要的一点就是ADC一定需要Vref的，但是STC为了简化引脚，会在单片机内部把Vref和AVCC都连接到VCC上。于是很多人就没见过有Vref的存在。不过现在STC8也有了把AVCC和Vref都引出引脚的型号了，比如STC8A8K64S4A12。
- “1.19V不是Vref，那搞这个1.19V多此一举干嘛？”有些网友充分理解Vref和1.19V的区别后就会有这个问题。从型号规划上来看，不是所有型号都有ADC，但是所有型号都有1.19V。再联想手册了有提到单片机内置有LDO，所以我断定这个1.19V主要是为了LDO服务的。连到ADC的15通道只是为了提供便利，可以为客户省下一个TL431芯片。

调用例程

```
#include "ecbm_core.h"//加载库函数的头文件。
float vref;
void main(){//main函数，必须的。
    system_init();//系统初始化函数，也是必须的。
    adc_init();//先初始ADC。
    vref=adc_read_vref();//读取Vref的电压值。
    while(1){
        delay_ms(1000);//每秒发送一次AD值到串口。
        debug("%f\r\n",adc_voltage(0,vref));//发送通道0的电压值。
    }
}
```

注意事项

1. 在使用本函数之前先初始化ADC，否则一定得不到正确的值。

adc_it_start

函数原型：void adc_it_start(void);

描述

开启ADC中断函数。

输入

无

输出

无

返回值

无

调用例程

```
if(key_flag==0){//如果按键按下，
    adc_it_stop();//先关闭ADC中断。
    ...//其他代码。
    adc_it_start();//再打开ADC中断。
}
```

注意事项

1. 要使用ADC的中断，必须先在adc.h的图形化配置界面使能ADC中断。
2. adc_init函数里会打开ADC中断，因此本函数实际上要和adc_it_stop搭配一起使用的。也就是说假如ADC中断没有被关闭过，就没必要再用这个函数，因为中断一直会开启着。

adc_it_stop

函数原型：void adc_it_stop(void);

描述

关闭ADC中断函数。

输入

无

输出

无

返回值

无

调用例程

```
if(key_flag==0){//如果按键按下，
    adc_it_stop();//先关闭ADC中断。
    ...//其他代码。
    adc_it_start();//再打开ADC中断。
}
```

注意事项

1. 要使用ADC的中断，必须先在adc.h的图形化配置界面使能ADC中断。
2. 本函数执行后会关闭ADC中断，使用adc_start函数可以再度打开ADC中断。

adc_read_start

函数原型：void adc_read_start(u8 ch);

描述

ADC转换开始函数。

输入

- ch：要读取AD值的通道号，从0开始。

输出

无

返回值

无

调用例程

```
adc_read_start(0);//准备读通道0的值。
```

注意事项

1. 本函数仅仅是开始一次转换，还不能马上得到AD值，当AD转换结束时会触发中断，在中断里才能读到本次测量的AD值。

adc_read_it和中断处理函数

函数原型：u16 adc_read_it(void);

描述

ADC读取AD值函数。

输入

无

输出

无

返回值

- 触发本次中断的通道的AD值。

调用例程

基本ADC

```
u16 adc_value;
...//其他代码。
adc_read_start(0);//准备读通道0的值。
...//其他代码。
void fun1(void)ADC_IT_NUM{//这是ADC的中断处理函数。
    adc_value=adc_read_it();//上次执行adc_read_start是读取通道0，所以这里读取到的是通道0的AD值。
}
```

注意事项

1. ADC中断只由AD转换完成标志位触发，也就是说adc_read_it函数经常会伴随着ADC中断处理函数一起出现。且adc_read_it只能放在中断处理函数中使用。两者是挂钩的。
2. 假如有多个通道的adc_read_start函数执行，那么可能会引发多次中断，顺序是adc_read_start函数执行的通道顺序。
3. 如果你在读这段话的时候感觉头晕，那请不要用中断法来读取AD值。直接用adc_read函数就行，简单快捷。我也觉得中断法没多大用处。

优化建议

去掉所有中断法，只使用查询法获取AD值。