**STM32的ADC配置问题**

**问题介绍**

最近要使用STM32F103C8T6来做个数字万用表，于是开始学习STM32，要用到32内部的12位ADC

等于是刚刚接触STM32，一切从零开始，现在分享下如何简单的使用ADC

**预备知识**

**RCC：**

这个是用来设置时钟的，比如我可以设置我的系统时钟频率等

**TIM：**

顾名思义，是timer的缩写，是定时计数器.

**RCC 和 TIM的区别：**

RCC用来设置我32的系统时钟频率或者是一些其他硬件的时钟频率

而TIM是在某个时钟频率下工作的一个计数器，这个频率可以来自RCC的设置，也可以来自外部

注意，RCC设置频率的来源也可以是外部或者内部（内部不准确，我们一般不用，这也是为什么要外接8MHz晶振的原因），而后产生一个内部时钟频率送给TIM

为什么要说时钟呢？因为我要使ADC的采样率达到最大，也就是1MHz的采样率，而达到这样的采样率就需要设置ADC的时钟频率，ADC最大时钟频率是14MHz。这两者什么关系呢？

后面介绍，总之先知道要达到最大的采样率就需要设置我们的时钟

**以下正文**

那么就让我开始来配置RCC吧！

**No.1**

首先我们应该用外部接的8MHz晶振来做时钟源。外部高速晶振：HSE；内部高速晶振：HSI

用 void RCC\_HSEConfig(u32 RCC\_HSE) 这个函数来启动，内部参数设置 RCC\_HSE\_ON

RCC\_HSEConfig(RCC\_HSE\_ON);//开启8MHz外部晶振

1

然后检测外部高速晶振是否正常启动

用 RCC\_WaitForHSEStartUp(); 这个函数， 这个函数返回 SUCCESS 这个参数则代表正常启动

ErrorStatus HSEStartUpStatus;//设置标志位

HSEStartUpStatus = RCC\_WaitForHSEStartUp();

if(HSEStartUpStatus == SUCCESS)//若外部晶振正常启动

{

/\* Add here PLL ans system clock config \*/

}

这时候我们要通过PLL锁相环来使 外部接的晶振作为输入，输出另一个稳定频率的时钟信号

即我们要用PLL来进行倍频

这里我们设置PLL输出 = 8MHz \* 7 = 56MHz （那就是要进行7倍频）

//RCC\_PLLSource\_HSE\_Div1 意思是 PLL的输入时钟 = HSE时钟频率

//RCC\_PLLMul\_7 表示 7倍频

RCC\_PLLConfig(RCC\_PLLSource\_HSE\_Div1, RCC\_PLLMul\_7);

然后输出

RCC\_PLLCmd(ENABLE);//PLL输出使能

while(RCC\_GetFlagStatus(RCC\_FLAG\_PLLRDY) == RESET);//等待PLL输出

到目前为止我们得到一个56MHz的时钟频率

然后我们要利用PLL输出的这个频率作为我们STM32的系统时钟

RCC\_SYSCLKConfig(RCC\_SYSCLKSource\_PLLCLK);//设置系统时钟为56MHz

while(0x08 != RCC\_GetSYSCLKSource());//等待系统时钟被正确设置

1

2

这样系统时钟就被我们设置好了

接下来的任务就是要设置AHB时钟了

what？AHB时钟又是个啥？

AHB(Advanced High performance Bus)高级高性能总线 或者我们叫它 系统总线

就是我们要设置STM32内部总线的时钟频率

而且AHB又有高低速之分，也就是说我们要设置两个时钟分别给高速AHB和低速AHB

AHB掌管着DMA时钟，SRAM时钟和FLITF时钟，它并不直接管ADC的时钟，那我们为什么还要设置它呢？先别急，往下看

RCC\_HCLKConfig(RCC\_SYSCLK\_Div1);//设置AHB时钟（HCLK）

RCC\_PCLK2Config(RCC\_HCLK\_Div1);//设置高速AHB时钟 PLCK2为56MHz (最大72MHz)

RCC\_PCLK1Config(RCC\_HCLK\_Div2);//设置低速AHB时钟 PLCK1为28MHz (最大36MHz)

//注：这里面Div1表示一倍分频，也就是不分频。 PLCK2 = HCLK = 56MHz

//Div2表示2倍分频 PLCK1 = HCLK / 2 = 28MHz

1

2

3

4

5

总线设置好了我们就终于可以开始设置ADC的时钟频率了

但这里又要提到一个名词APB

APB(Advanced Peripheral Bus)外围总线

这个才是直接管ADC时钟的总线，APB又分APB1 和 APB2

APB1管TIMx (x = 2， 3， 4 ……) WWDG，SPI2， USART2， USAT3， I2C， CAN的时钟

APB2管TIM1， GPIOx， ADC1， ADC2， SPI1， USART1的时钟

很明显我们只需要对APB2的ADC功能进行设置就行

//使能ADC & GPIOA

//这里我用ADC1采样，PA0端口，具体看各个芯片的数据手册

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA | RCC\_APB2Periph\_ADC1, ENABLE);

重点来了

现在来设置ADC了！！！

来上函数 void ADC\_ADCCLKConfig(u32 RCC\_ADCCLKSource)

看看这个参数 RCC\_ADCCLKSource: 定义ADCCLK，该时钟源自APB2时钟（PCLK2）

懂了吧，为啥非要对AHB进行设置，ADC的时钟来源于AHB的PLCK2

//RCC\_PCLK2\_Div4 意思是 ADC时钟 = PCLK2 / 4 = 14MHz

RCC\_ADCCLKConfig(RCC\_PCLK2\_Div4);//ADC最大时钟频率是14MHz

至此，我们就对ADC时钟设置完了，一般来说，我们的STM32系统时钟都是设置的是72MHz，但这里我们为什么非要费那么老劲来设置RCC呢？还是上面这个函数，它的参数一共就4个

Div2 Div4 Div6 Div8 也就是2 4 6 8 分频。

72MHz并不能通过这四个分频得到14MHz最大时钟，所以我们特地设置56MHz，通过4分频，产生14MHz的ADC最大时钟频率。72MHz最大能做到 72 / 6 = 12MHz，64/6=10.66666MHz

来吧，上一份完整的RCC代码

static void RCC\_ConfigInitail()

{

ErrorStatus HSEStartUpStatus;

FlagStatus Status;

//RCC配置

RCC\_DeInit();//重置

RCC\_HSEConfig(RCC\_HSE\_ON); //外部8MHz晶振启动！

HSEStartUpStatus = RCC\_WaitForHSEStartUp();

if(SUCCESS == HSEStartUpStatus)//若启动成功

{

RCC\_PLLConfig(RCC\_PLLSource\_HSE\_Div1, RCC\_PLLMul\_7);//56MHz PLL输出

RCC\_PLLCmd(ENABLE);//PLL输出使能

while(RCC\_GetFlagStatus(RCC\_FLAG\_PLLRDY) == RESET);//等待PLL输出成功

//设置系统时钟56MHz

RCC\_SYSCLKConfig(RCC\_SYSCLKSource\_PLLCLK);

while(0x08 != RCC\_GetSYSCLKSource());//等待设置成功

RCC\_HCLKConfig(RCC\_SYSCLK\_Div1);

RCC\_PCLK2Config(RCC\_HCLK\_Div1);//PLCK2 56MHz

RCC\_PCLK1Config(RCC\_HCLK\_Div2);//PLCK1 28MHz

//使能APB2外设时钟 ADC & GPIOA

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA | RCC\_APB2Periph\_ADC1, ENABLE);

RCC\_ADCCLKConfig(RCC\_PCLK2\_Div4);//ADC1时钟频率 14MHz

}

}

1

等有时间再更新后面的

2018 / 4 / 4 更新……

好了，我们现在来说一说ADC的配置吧

ADC的配置就没什么值得说的，它没有定时器难理解

直接上代码，注释就是解释

ADC\_InitTypeDef ADC\_InitStructure;

GPIO\_InitTypeDef GPIO\_InitStructure;

//GPIO配置

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_0;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AIN;//GPIO采用模拟输入

GPIO\_Init(GPIOA, &GPIO\_InitStructure);//对PA0初始化

//因为我使用的是ADC1\_IN0就是通道0，而这对应STM32C8T6的PA0口

//对于其它型号的要具体看芯片手册

//ADC配置

ADC\_DeInit(ADC1);//重置

ADC\_InitStructure.ADC\_Mode = ADC\_Mode\_Independent;//ADC1和ADC2单独工作，互不影响

ADC\_InitStructure.ADC\_ContinuousConvMode = DISABLE;//ADC单次采样，即采样一次就停止

ADC\_InitStructure.ADC\_ScanConvMode = DISABLE; //ADC单通道采样(ENABLE是多通道扫描)

ADC\_InitStructure.ADC\_ExternalTrigConv = ADC\_ExternalTrigConv\_None;//软件触发ADC

ADC\_InitStructure.ADC\_DataAlign = ADC\_DataAlign\_Right;//数据右对齐

ADC\_InitStructure.ADC\_NbrOfChannel = 1;//ADC通道转换数目，我们只用一个ADC，那么就是1

//对于多通道采集才使这个值 >= 2， 取值范围是1~16

ADC\_Init(ADC1,&ADC\_InitStructure);//初始化

ADC\_Cmd(ADC1, ENABLE);//使能

//ADC校准

ADC\_ResetCalibration(ADC1);//重置ADC校准器

while(ADC\_GetResetCalibrationStatus(ADC1));//等待重置结束

ADC\_StartCalibration(ADC1);//开始校准

while(ADC\_GetCalibrationStatus(ADC1));//等待校准完成

ADC\_RegularChannelConfig(ADC1, ADC\_Channel\_0, 1, ADC\_SampleTime\_1Cycles5);

最后一句话是重点

这句话是重点，看函数名就知道是对ADC通道配置的

这里我们就用一个通道，所以就只写一句，如果有多通道，那么就多写几句，参数变一下即可

要知道ADC转换周期是12.5个ADC时钟周期 而ADC\_SampleTime\_1Cycles5就是ADC采样时间为 1.5个周期，所以一共12.5 + 1.5 = 14个周期

14MHz / 14 = 1MHz，这样我们就能得到1MHz最大的采样率了

除了1.5以为，STM固件库还给出了其它的一些稀奇古怪的周期

例如：

7.5

13.5

28.5

41.5

55.5

71.5

239.5

呐，就这，ADC最基本的应用就配置完成了

剩下的就是些不痛不痒的代码了

static u16 Get\_ADC\_Value()

{

ADC\_SoftwareStartConvCmd(ADC1, ENABLE);//软件启动，ADC开始转换

while(ADC\_GetSoftwareStartConvStatus(ADC1));//等待转换完成

return ADC\_GetConversionValue(ADC1);//返回得到的ADC值

}

6

到目前为止ADC的最简单的配置就完成了

下面是我个人的情况，可以选择性阅读

下面是我实力分析为什么在高采样率的情况下，上面这些普通的代码不适合测AC交流

这么说吧，这是我一开始写的代码，目的是为了采集交流信号（这不废话么）

要求是测得10HZ ~ 100KHZ的交流信号的有效值

于是我想采用计算真.有效值的方法进行计算

真.有效值的计算公式是

exp( 1/T \* ∫ f(x)² dt ) (积分区间是0 ~ T)

将之变成离散的信号处理，那么公式就又积分变到求和

exp( 1/n \* ∑ f(x)² )（求和的话，那就是从0 ~ n）

那么也就是说我需要采样，一个周期采样n个点，n越多，计算得到的真有效值就越接近真值，跟做个示波器差不多了

如此一来我写了如下的程序

//times表示一个周期采样的个数，即上面说的n

float Get\_ADC\_EffectiveValue(u8 times)

{

float sum = 0.0;//∑ f(x)²

float temp = 0.0;//得到返回的ADC值

u8 t = 0;

for(t = 0; t < times; t++)

{

//12位采样(0xFFF = 4096)，STM32是3.3V供电

//所以 3.3 / 0xFFF ≈ 0.000805664

temp = (Get\_ADC\_Value() - 2048) \* 0.000805664f;

sum += temp \* temp;

}

return (float)sqrt(sum / times);

}

这里我是通过电路将AC信号控制在0 ~ 3.3V，也就是原本的AC信号加上一个DC直流偏置

直流偏置 = VCC / 2，所以我要Get\_ADC\_Value()之后减去直流偏置，得到原来真实的量

这里再补充一个小细节，再0.000805664后加上个f，意思是告诉CPU这是一个float型常量，别给我弄成个double型了，因为STM32是32位CPU，那么也就是说在32位以下的数据计算速度都差不多。

float占4个字节（32位），double占8个字节（64位），很明显处理float要比double快多了，以后建议能用float就尽量用float

那么好了，我的具体方案是这样的，通过TIM的输入捕获功能来捕捉到信号的上升沿，在通过计算差值得到周期T，我一共捕捉11个上升沿，这样我就得到了10个周期，求平均值后使误差减小

然后ADC开始采样，根据得到的周期来确定一个周期我要采样几个点，就这样

然而后面我发现一系列问题，导致我现在否定了这个方案。

来，我们来分析分析

while(1)

{

反复分析;//2333333

}

首先我们的定时计数器是工作在56MHz下的，我设置的可计数值为65535，不分频

那么有可能这个周期比较长，我计数器在从0计数到65535后，还没捕捉到第二个上升沿，那么我的这个计数器就会溢出。不要紧，我设置一个全局变量u8 OverFlow = 0;//记录溢出次数

每当我的计数器溢出的话，OverFlow ++;

我计数的Counter = f\_TIM / f\_IN

f\_TIM表示计数器频率即56MHz，f\_IN表示输入信号的频率

假如输入信号频率f\_IN = 100KHz，那么Counter = 560，嗯，没什么问题

假如f\_IN = 10Hz，那么Counter = 5.6 \* 1e6，这个可以么？不会太大么？

OverFlow 最大计数255，所以最大Counter = 255 \* 65536，这么看有点看不出来，不是很直观

反过来求一下5.6 \* 1e6 / 65536 后向下取整为85，也就是说OverFlow 最大只会计数到85，也不是问题

也有人会问，中断不管么？有时间误差啊！

管什么？最多85 + 1次中断，+1是因为捕获到上升沿。急什么，忽略这个时间误差

假如我只知道f\_IN，我想知道我的f\_TIM应该设置多少？

这样，我们假设Counter >= 100， 那么我计数器从0计数到1的时间间隔t = T\_IN / Counter

也就是最小误差(仪器误差)t <= T\_IN /100，嗯，1%不到的误差，可以忽略了。

结论一：即当Counter >= 100 时，可以忽略计算周期的误差

那么我就取Counter >= 100

所以f\_TIM >= 100 \* f\_IN，STM32最大72M时钟频率，所以理论上能精确计算的最大输入信号周期的为720KHz。当然，这不是绝对的，你可以通过输入捕获预分频，来扩大这个值，误差还是 1 / Counter这里我们不做过多讨论

由于我们的Counter最小为560，所以根据结论一 可知，一个周期下，输入捕获计算周期的误差忽略，再加上我还通过取平均来减小误差，使得计算得到的误差更小了

结论二：输入捕获计算周期的误差可以忽略不计，不需要管输入捕获了

接下来我们来分析ADC

(嘿嘿嘿！好玩的来了！)

我们得到了输入信号的周期T后，就可以计算一个周期下，ADC采样的次数了

设一个周期内，ADC采样次数为n

则有n = f\_ADC / f\_IN

f\_ADC为ADC的采样频率，f\_IN为信号输入频率

那么我们还采用刚才的思路看看，已知f\_IN，取n >= 100

则 f\_ADC >= 100 \* f\_IN

确实，前面说过，n越大，越接近积分得到的结果

但是！！！CPU计算也需要时间啊

看看这段代码

for(t = 0; t < times; t++)// 2n

{

temp = (Get\_ADC\_Value() - 2048) \* 0.000805664f;// 3n

sum += temp \* temp;// 3n

}

这时候我们就不得不计算程序的时间复杂度了

我就假设+ - \* / 都是一样的计算时间，都是一个执行周期(真正的情况下乘除要比加减慢，更不要说你根本不知道编译器翻译成汇编之后会有多少条指令，我这么算算是少的)

t < times 需要n次判断

t++需要n次计算

我就假设Get\_ADC\_Value()得到值后存入内存不要时间

Get\_ADC\_Value() - 2048需要n次计算

\* 0.000805664f需要n次计算

赋值到temp需要n次计算

temp \* temp需要n次计算

sum + (temp \* temp)需要n次计算

赋值到sum需要n次计算

呵呵，一共是8n的时间复杂度

ARM3官方给出的平均计算速度是1.25MIPS/MHz

意思是1MHz频率下，每秒执行1.25M条指令。我们是56MHz主频，那么就是每秒56M \* 1.25条指令

执行一条指令需要 1 / (56 \* 1.25 \* 1e6)

也就是说光是用在计算上的时间就需要

t = 8n / (56 \* 1.25 \* 1e6) ≈ 0.114n （单位us）

我们来看看n取100时候的情况

t = 11.4us， 而对于100KHz信号来说，100KHz = 10us，呵呵，光计算的时间就差了一个周期！！

对于10KHz的信号也会有10%的误差

只有1KHz以下的信号才不会收到影响…………

连采集到的信号都不对，谈什么求和逼近积分……做梦吧，梦里啥都有……

结论三：普通方法不能求得1KHz以上信号的准确 真有效值

正所谓鱼和熊掌不可兼得……

但天无绝人之路，我还有DMA！！！意不意外，惊不惊喜！没想到吧！(此处自动脑补表情包)

可是我现在还不会……等我学会了再回来更新，就酱，如果本文对你有收获，想收藏就收藏，想点赞就点赞

---------------------

作者：尧风之下嗅桂香

来源：CSDN

原文：https://blog.csdn.net/weixin\_40119511/article/details/79735003

版权声明：本文为博主原创文章，转载请附上博文链接！