SDK应用_ADC模块配置及应用（一）

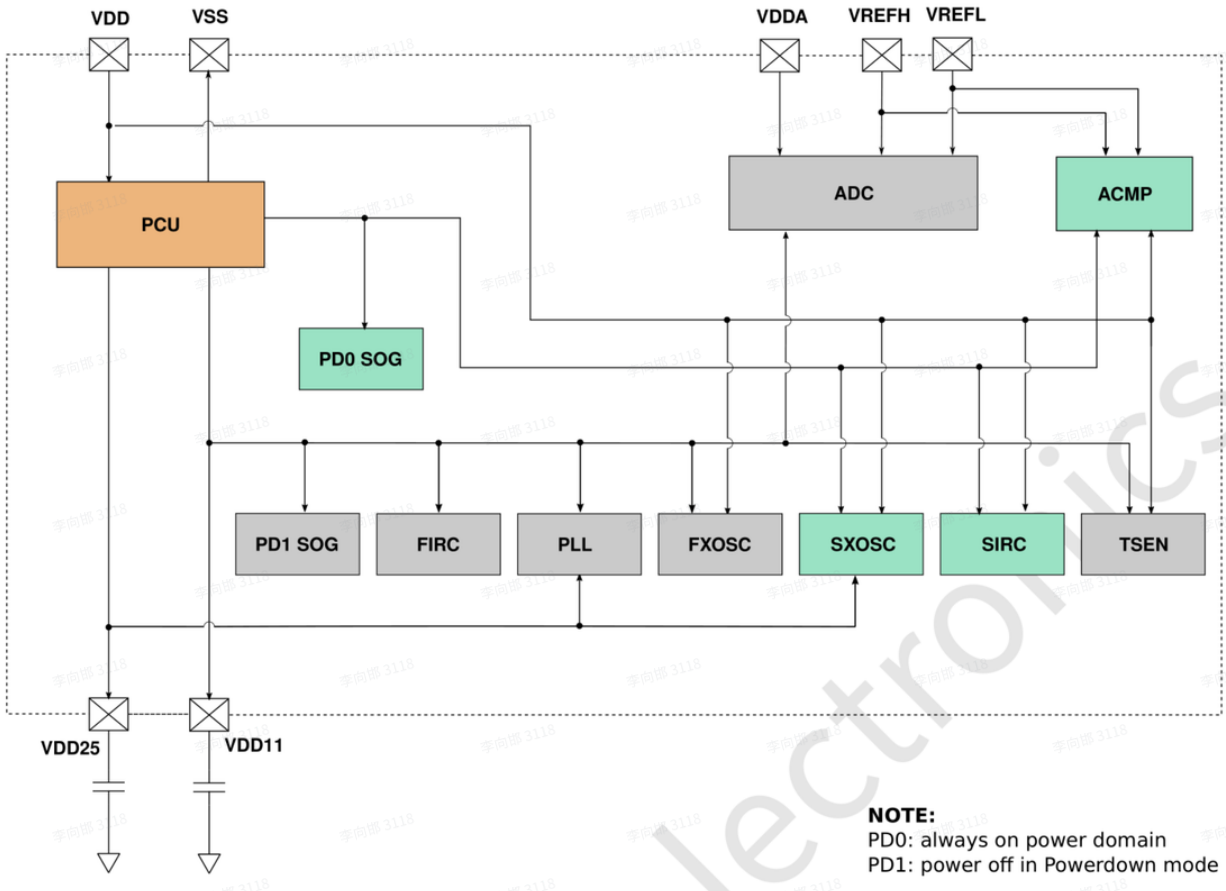
版本：

Config Tool Version：2.0.4

ME0 SDK Version：1.0.4

前言：

ADC采集信号由于其数字信号本身不具有实际意义，仅仅表示一个相对大小，故任何一个模拟数字转换器都需要一个参考电压作为转换的标准，该参考电压（VREFH和VREFL）可以选择MCU供电电压（3.3V或5V）或者选择独立的电源作为参考电压，选择独立的电源作为参考电压要将VREFH接高精度电源，VREFL通过0欧电阻接VSS，该接法适用于高精度场景，会提高其信号采集的稳定性，减少信号采集波动。另外ADC需要VDDA供电，最好将VDD和VDDA供电保持一致，并且VREFH＜=VDDA。

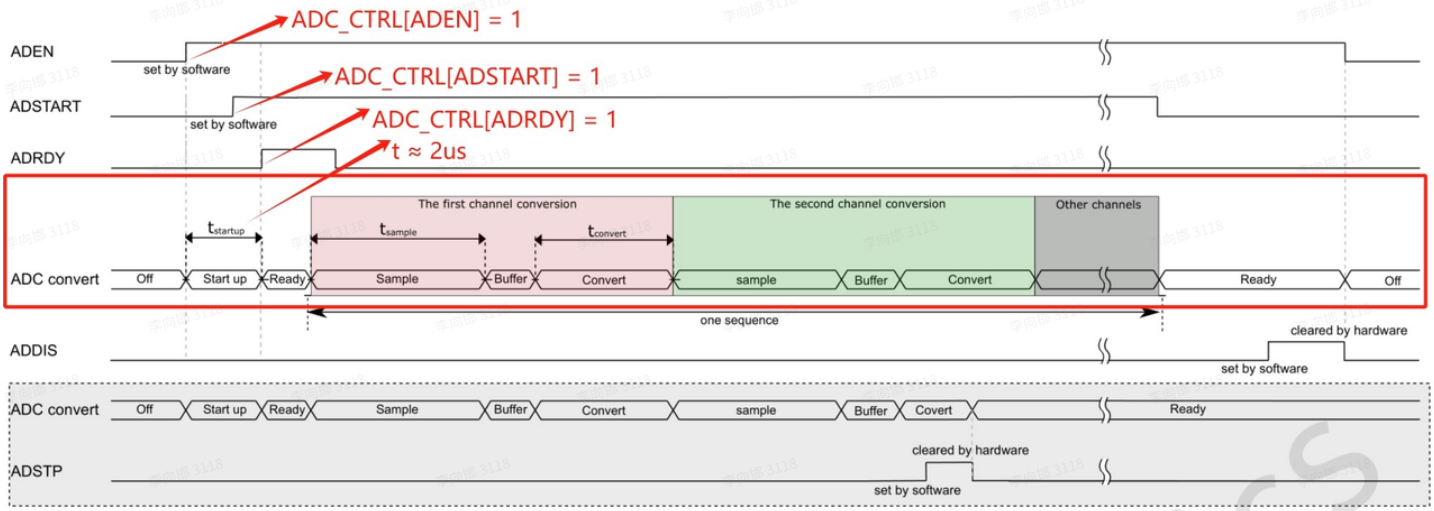


ADC的总线时钟为SLOW_BUS_CLK,支持Peripheral Functional Clock。在Sleep，Deepsleep休眠模式下ADC可作为唤醒源。另外当MCU在Sleep和Deepsleep模式下唤醒时ADC可全功能工作，在

Standby模式下唤醒时ADC寄存器配置会被保持，在Powerdown模式下唤醒时需要重启ADC才能继续工作。

本章将逐步介绍ADC模块的配置方式，详解配置过程中的参数，并最终列举1.interrupt采集用例（见ADC模块配置及应用一）,2.DMA采集用例（见ADC模块配置及应用二），3.polling采集用例（见ADC模块配置及应用三），并分别加以说明。

1. ADC工作时序图：



若想启用ADC转换器首先要将ADEN = 1,然后进入start up时间（≈ 2us），在start up期间将ADSTART = 1,待ADC转换器供电稳定后将READY = 1,此时可以开始转换

2. ADC采样率的算法：

Fadc_func = Ffirc / adc_div

Fadc_func_clk_div = Fadc_func / (adc_clockDivider + 1)

Fadc_smp = Fadc_func_clk_div / (resolution +sample + 2)

注：L系列ADC采样率最大不能超过1M，M系列不能超过2M

3. Config Tool 配置：

3.1 引脚配置：

Pin Number	Pin Name	Feature	ALT Value	Laber	Direction	Interrupt Status	Interrupt Configuration	Pull Enable	Pull Select	Digtal Filter	Passive Filter	Driver Strength	Slew Rate	Init Value
115	PTA_0	ADC0_SE0 ACMP0_IN0	PCTRL_PIN_DISABLED			Don't modify	ISF Disable	Disabled		Disabled	N/A	Low	Fast	Low
113	PTA_1	ADC0_SE1 ACMP0_IN1	PCTRL_PIN_DISABLED			Don't modify	ISF Disable	Disabled		Disabled	N/A	Low	Fast	Low
85	PTA_6	ADC0_SE2	PCTRL_PIN_DISABLED			Don't modify	ISF Disable	Disabled		Disabled	N/A	Low	Fast	Low
83	PTA_7	ADC0_SE3	PCTRL_PIN_DISABLED			Don't modify	ISF Disable	Disabled		Disabled	N/A	Low	Fast	Low

3.2 时钟配置：

设置ADC时钟源

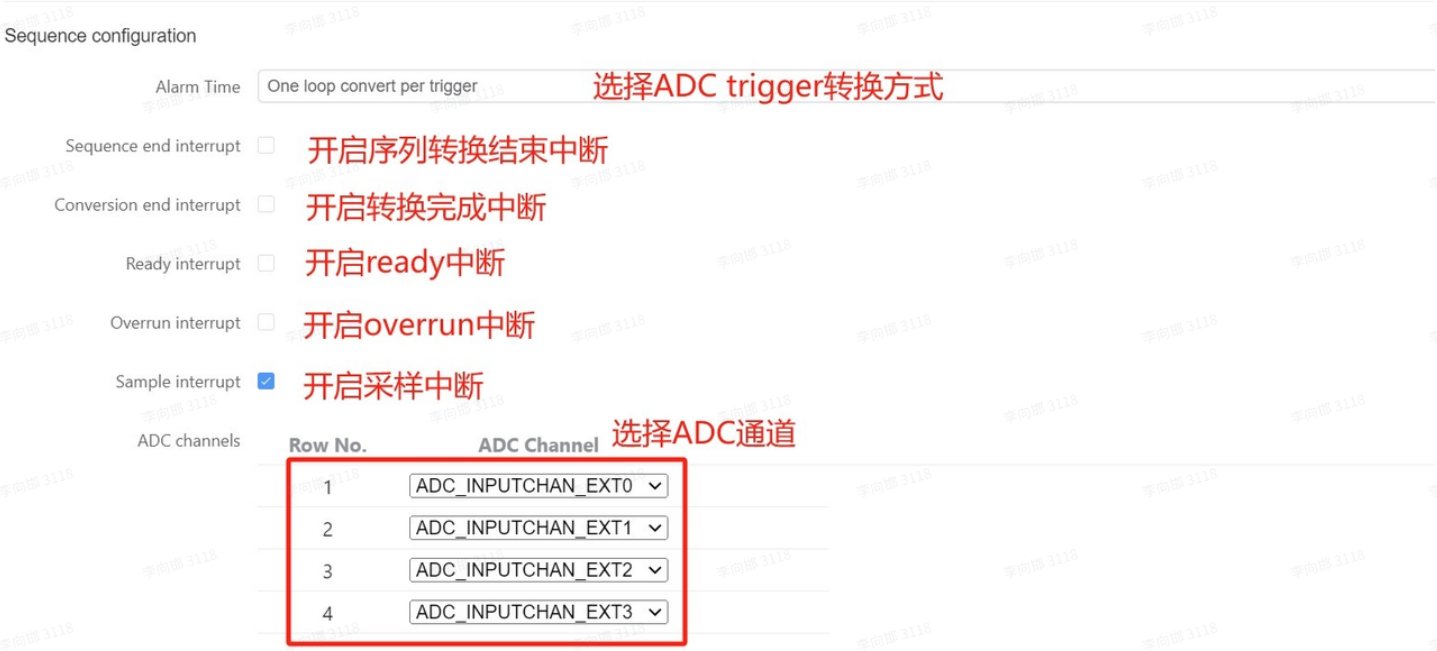
Name	UserCtrl	Gate	Divider	Peripheral Functional Clock
FlexCAN5_CLK	☐	☐		
ADC0_CLK	☑	☑	2	CLK_SRC_FIRC
ADC1_CLK	☐	☐	1	CLK_SRC_DISABLED
ACMP0_CLK	☐	☐	1	CLK_SRC_DISABLED

3.2.1 外设模块配置：

关于ADC分频，在IPC分频后，在经过ADC内部分频后的时钟必须小于48M

关于ADC采样时间（sample time）不可以设置太小，时间太短会影响切换的采样结果

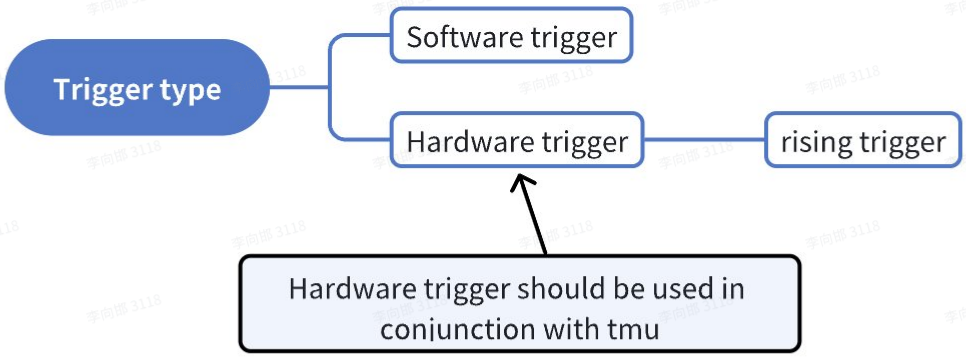
关于ADC的左右对齐选择，虽然config tool中可以选择左右对齐方式，但是芯片不支持左右对齐选择，并且默认是右对齐



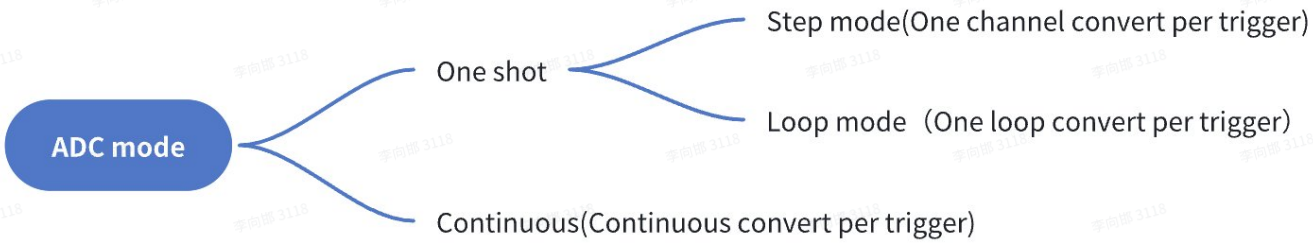
这里从ADC模块配置中挑选了几个配置项进行介绍：

- Trigger type可选择：

只有LD支持多种触发方式，LE/ME/MD/HA都只支持rising trigger



- ADC mode可选择：



ADC转换模式可分为One-shot和Continuous两种，One-shot为一次转换，一次转换包含单步转换（Step Mode）和多步转换（Loop Mode），单步转换每次转换一个channel，多步转换每次转换一组channel

- 添加API函数：

4. ADC模块应用实例：

4.1 interrupt采集用例：

分别用ADC0_SE0,ADC0_SE1,ADC0_SE2,ADC0_SE3四个通道进行ADC采样，分别输入0V,2.7V,3.3V,5V，读出ADC采样值，做法如下：

Config tool配置方法请按照上图Config tool配置教程配置，配置完点击Generate生成工程，然后打开工程，添加ADC0_IRQHandler接口

```
/* Private function declare ----------------------------------------------------*/
/* USER CODE BEGIN PFDC */
/* USER CODE END PFDC */
static void Board_Init(void);

/* Private user code -----------------------------------------------------------*/
/* USER CODE BEGIN 0 */
static uint16_t data[4];
void ADC0_IRQHandler(void)
{
    ADC_DRV_ClearEoseqFlagCmd(0);

    for (int i = 0; i < 4; i++)
    {
        data[i] = ADC_DRV_ReadFIFO(0);
    }
}
/* USER CODE END 0 */


/**
  * @brief  The application entry point.
  * @retval int
  */
```

adc_config0配置如下：


```c
*
* @file adc_config.c
* @brief
*
*/

#include "adc_config.h"

/* adc_config0 */
const adc_converter_config_t adc_config0={
    .clockDivider=2,
    .startTime=32,
    .sampleTime=2,
    .overrunMode=false,
    .autoOffEnable=false,
    .waitEnable=false,
    .triggerSource=0,
    .trigger=ADC_TRIGGER_SOFTWARE,
    .align=ADC_ALIGN_RIGHT,
    .resolution=ADC_RESOLUTION_12BIT,
    .dmaWaterMark=0,
    .dmaEnable=false,
    .sequenceConfig={
      .sequenceMode=ADC_CONV_CONTINUOUS,
        .sequenceIntEnable=true,
        .convIntEnable=false,
        .readyIntEnable=false,
        .ovrunIntEnable=false,
        .sampIntEnable=false,
        .channels={
            ADC_INPUTCHAN_EXT0,
            ADC_INPUTCHAN_EXT1,
            ADC_INPUTCHAN_EXT2,
            ADC_INPUTCHAN_EXT3,
        },
        .totalChannels=4,
    },
    .compareConfig={
        .compareEnable=false,
        .compareAllChannelEnable=false,

        .compHigh=4095,
        .compLow=0,
        .compIntEnable=false,
    },
};
```

上面未提到的参数在本例程中并未用到，默认就好，下面解释一下个参数用法

.autoOffEnable为ADC自动关断模式使能，在转换结束后可自动关闭

.waitEnable为ADC等待模式使能，防止FIFO装满时ADC溢出

灵活运用上述两种模式可以达到省电的效果

.overrunMode为ADC溢出管理模式，当.overrunMode = 0时，当检测到溢出时，FIFO register仍保留之前的数据，并不会被新数据覆盖。当.overrunMode = 1时，当检测到溢出时，最后一个数据将会被覆盖掉

.sequenceIntEnable为序列结束中断使能

.convIntEnable为转换结束中断使能

.readyIntEnable为准备转换中断使能

.ovrunIntEnable为ADC overrun中断使能

.sampIntEnable为采样结束中断使能

其中时钟分频和采样时间的设定和ADC选用的时钟源和ADC时钟分频有关,ME0 adc采样率最大不能超过2M,计算公式如下：

```
22
23  /*! @brief peripheral clock PeripheralClockConfig */
24
25 peripheral_clock_config_t clock_config0PeripheralClockConfig[1] = {
26     {
27         .clkName = ADC0_CLK,
28         .clkGate = true,
29         .divider = DIV_BY_2,
30         .clkSrc = CLK_SRC_FIRC,
31     },
32 };
33
```

Fadc_func = Ffirc/2 = 96M/2 = 48M

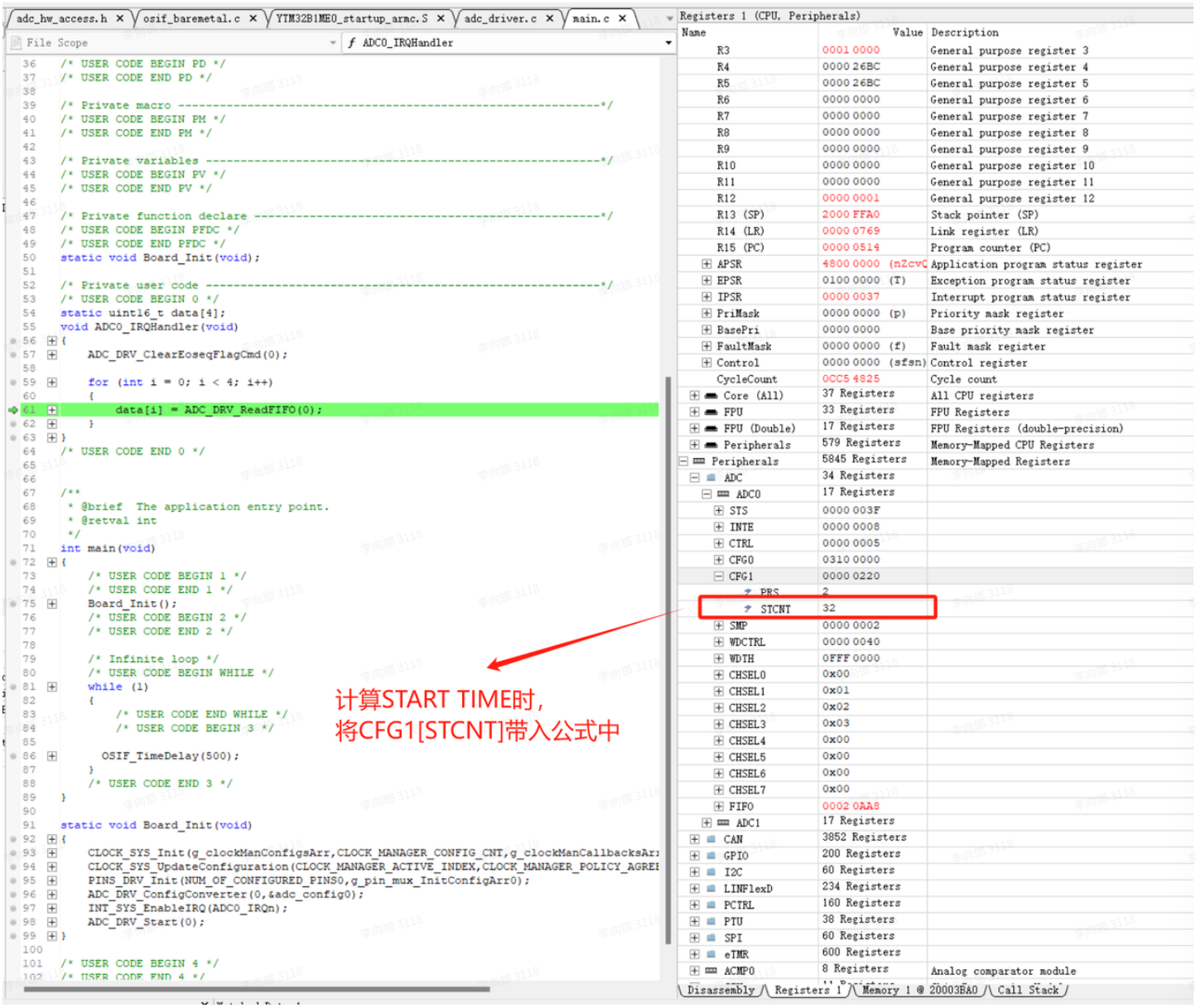
Fadc_func_clk_div = Fadc_func /（PRS +1）= Fadc_func / (clockdriver + 1) = 48 /（2 + 1）= 16M

Fadc_smp = Fadc_func_clk_div / (resolution + sample + 2) = 16 / (12 + 2 + 2) = 1M

Tips：计算采样率时实际的分频是clockdriver + 1

ADC启动时间（start time）要≥2us，计算公式为：

T（start up）= CFG1[STCNT] / F(func_clk_div) = 32 / 2 = 16us



参数设定完成后，分别给四个ADC通道输入0V,2.7V,3.3V,5V电压信号，读取ADC 4个通道采样值，采样数值分别为：

data[0] = 0,data[1] = 2239,data[2] = 2733,data[3] = 4094,换算成电压为：

ADC_channel0 = 0V,ADC_channel1 = 2.73V,ADC_channel2 = 3.3V,ADC_channel3 = 4.99V

Watched Data 1

Expression	Value	Location
⊟ data		1FFF 0490
[0]	0	1FFF 0490
[1]	2 239	1FFF 0492
[2]	2 733	1FFF 0494
[3]	4 094	1FFF 0496
▸		

DMA采集用例和polling采集用例请见ADC模块配置及应用（二&三）

5. ADC常用API简介:

void ADC_DRV_ConfigConverter();	ADC配置转换函数
void ADC_DRV_InitConverterStruct();	ADC配置转换初始化函数
void ADC_DRV_InitHwCompareStruct();	ADC硬件比较配置初始化函数
void ADC_DRV_InitSequenceStruct();	ADC序列控制初始化函数
void ADC_DRV_ClearEocFlagCmd();	清除EOC标志位函数
void ADC_DRV_ClearEoseqFlagCmd();	清除EOSEQ标志位函数
void ADC_DRV_ClearOvrFlagCmd();	清除Ovr标志位函数
void ADC_DRV_ClearReadyFlagCmd();	清除Ready标志位函数
void ADC_DRV_ClearSampEndFlagCmd();	清除采样结束标志位函数
void ADC_DRV_ClearWdFlagCmd();	清除watchdog标志位函数
void ADC_DRV_ConfigHwCompare();	ADC硬件比较配置函数
void ADC_DRV_ConfigSequence();	ADC配置序列转换函数
void ADC_DRV_Disable();	ADC失能函数

void ADC_DRV_Enable();	ADC使能函数
bool ADC_DRV_GetConvCompleteFlag();	返回转换完成的状态函数
void ADC_DRV_GetConverterConfig();	获取当前转换配置函数
bool ADC_DRV_GetEndOfConversionFlag();	返回转换结束的状态函数
bool ADC_DRV_GetEndOfSequenceFlag();	返回序列转换完成的状态函数
bool ADC_DRV_GetFullOfConversionFlag();	返回转换已完成的状态函数
void ADC_DRV_GetHwCompareConfig();	返回硬件比较配置功能函数
IRQn_Type ADC_DRV_GetInterruptNumber();	返回ADC中断向量号函数
bool ADC_DRV_GetOvrRunOfConversionFlag();	返回ADC转换溢出状态函数
bool ADC_DRV_GetReadyFlag();	返回ADC ready状态函数
bool ADC_DRV_GetSampEndFlag();	返回ADC采样结束状态函数
void ADC_DRV_GetSequenceConfig();	返回ADC当前序列配置函数
bool ADC_DRV_GetWatchdogFlag();	返回ADC_watchdog状态函数
uint16_t ADC_DRV_ReadFIFO();	读取FIFO转换结果函数
uint32_t ADC_DRV_ReadSeqtagAndData();	读取FIFO转换结果函数
void ADC_DRV_Reset();	ADC复位函数
void ADC_DRV_Start();	ADC转换启动函数
void	ADC转换终止函数
void ADC_DRV_WaitConvDone();	ADC等待转换完成函数
void ADC_DRV_WaitSequenceDone();	ADC等待序列转换完成函数

6. 有关ADC问题解答：

6.1 ADC去耦电容该加多大：

答：

典型值为100nf，电容尽可能放置在VREFH/VREFL或者VDD/VSS旁边

6.2 ADC最多能转换几个通道

答：

ADC一次最多转换8个通道，超过8个通道需分多次转换

6.3 ADC有没有内部参考电压源

答：

没有内部参考电压源， 需要用户外接电压到VDDA和VREFH上面

6.4 内部温度采集怎样将采样值转化成温度

答：

运用公式T = C - K*V

K = 616.9,V = (D/4096)*Vrefh （D为ADC采样值）

先将室温稳定到温度：T0，求出C = T0 + K*V

求出C后再将C值带入T = C-K*V求T值

6.5 ADC检测温度的精度范围是多少

答：

校正后偏差±5°C

6.6 ADC的function clock频率过高会导致采集结果不准确

答：

L系列ADC function clock不能高于32M，M系列ADC function clock不能高于40M

原因：ADC在convert段进行转换时，一个时钟周期对应一个比特，如果是12bit精度，则需要12个时钟周期，ADC硬件转换是通过电阻和电容的组合完成的，当单个时钟周期时间太短时，电容充电不完全会导致转换不完全，时序如下图所示

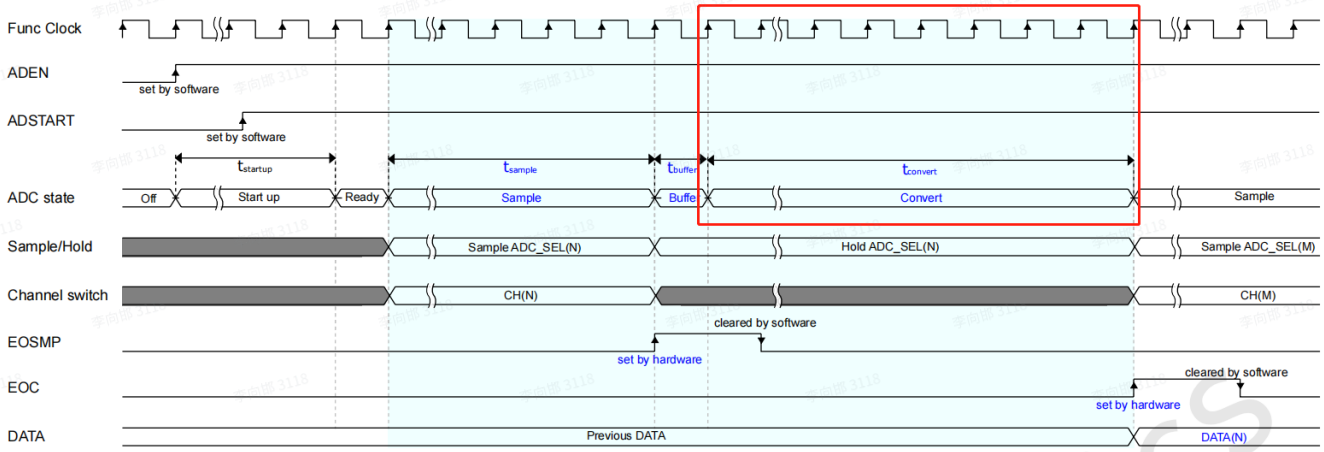


Figure 24.4: ADC Convert Time

6.7 ADC基准电压采集通道

答：

LE有一个bandgap基准电压（1V），但是未对用户开放，若要使用该电压需在ADC start之前增加如下代码，并且将ADC采集通道配置为23，代码：PCU->RESERVED_4 = (uint32_t)(1 << 3);，再添加以下代码可以将bandgap电压引到PTB4上测量，rdata = *(volatile uint32_t *)(0x40073030); //可将bandgap电压引到PTB4上测量，先读后写，*(volatile uint32_t *)(0x40073030) = rdata | 0x1000;，该通道是留给内部测试用的，只有一个地方需要注意一下，LE的bandgap这个通道可以保证ADC的采样频率在低于300kHz的时候是比较准的，如果高于300kHz可能偏差就会大一点，其他方面和其他通道无异，ME内部通道VDD25、MD内部通道PMC(1.2V)可直接使用。

6.8 ADC模拟输入口对对硬件有哪些要求

答：

1. 上拉电阻要＞15K
2. 端口处接≤50pF的电容