NV32F100x ADC 模块编程示例

第一章 所有库函数简介

库函数列表

void ADC_Init(ADC_Type *pADC, ADC_ConfigTypePtr pADC_Config)
用结构体来初始化 ADC
void ADC_DeInit(ADC_Type *pADC)
回复出厂初始化设置
unsigned int ADC_PollRead(ADC_Type *pADC, uint8_t u8Channel)
单次 ADC 转换，并获取 ADC 转换值
void ADC_SetCallBack(ADC_CallbackType pADC_CallBack)
设置回调函数
void ADC_SetChannel(ADC_Type *pADC, uint8_t u8Channel)
设置 ADC 转换的通道
void ADC_VrefSelect(ADC_Type *pADC, uint8_t u8Vref)
设置 ADC 转换参考电压
void ADC_SelectClockDivide(ADC_Type *pADC, uint8_t u8Div)
设置 ADC 时钟分频系数
void ADC_SetMode(ADC_Type *pADC, uint8_t u8Mode)
设置 ADC 转换模式（8/10/12）
void ADC_SelectClock(ADC_Type *pADC, uint8_t u8Clock)
设置 ADC 模块时钟选择
void ADC_SetFifoLevel(ADC_Type *pADC, uint8_t u8FifoLevel)
设置 ADC FIFO 的深度
void ADC_Isr(void)
ADC 模块中断服务程序

另外还有寄存器操作的静态内联函数（内联函数调用不会产生多余跳转，但会增加代码）
void ADC_IntEnable(ADC_Type *pADC); //中断使能
void ADC_IntDisable(ADC_Type *pADC);//禁止中断
void ADC_ContinuousConversion(ADC_Type *pADC);//设置连续转换模式
void ADC_SingleConversion(ADC_Type *pADC);//设置为单次转换模式
void ADC_SetSoftwareTrigger(ADC_Type *pADC);//设置为软件触发模式
void ADC_SetHardwareTrigger(ADC_Type *pADC);//设置为硬件触发模式
void ADC_CompareEnable(ADC_Type *pADC);//转换结果比较器使能
void ADC_CompareDisable(ADC_Type *pADC); //关闭结果比较
void ADC_CompareGreaterFunction(ADC_Type *pADC);//大于
void ADC_CompareLessFunction(ADC_Type *pADC);
void ADC_SetLowPower(ADC_Type *pADC);//低功耗模式
void ADC_SetHighSpeed(ADC_Type *pADC);//高速模式
void ADC_SetLongSample(ADC_Type *pADC);//设置为长采样模式
void ADC_SetShortSample(ADC_Type *pADC);//设置为短采样模式
void ADC_FifoScanModeEnable(ADC_Type *pADC); //FIFO 扫描模式，持续转换

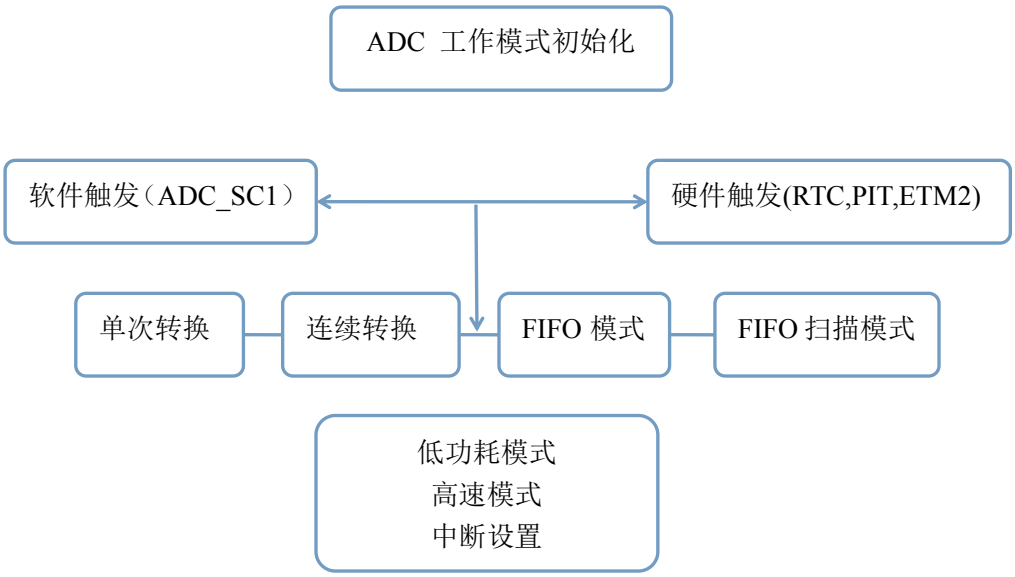
void ADC_FifoScanModeDisable(ADC_Type *pADC); //禁止 FIFO 扫描模式
void ADC_CompareFifoOr(ADC_Type *pADC);
void ADC_CompareFifoAnd(ADC_Type *pADC);
uint16_t ADC_ReadResultReg(ADC_Type *pADC); //读取 ADC 转换值
uint8_t ADC_IsConversionActiveFlag(ADC_Type *pADC);//ADC 转换值是否有效
uint8_t ADC_IsCOCOFlag(ADC_Type *pADC);//转换完成标志是否置位
uint8_t ADC_IsFIFOEmptyFlag(ADC_Type *pADC);//FIFO 是否为空
uint8_t ADC_IsFIFOFullFlag(ADC_Type *pADC);//FIFO 满标志

ADC 特性说明

* 8/10/12bit 3 种精度可选
*单次转换和连续转换 模式可选
*可选软件触发模式和硬件触发模式
*ADC 有普通 FIFO 模式和 FIFO 连续扫描模式
*可选的异步硬件触发信号有 RTC,PIT,ETM 计数器（SIM_OPT 寄存器来选择）
*ADC 转换值可以通过中断，也可以通过对转换标识位轮询来获取
*FIFO 转换模式，可以最大程度的减少 CPU 中断，CPU 处理 ADC 服务例程的负荷

ADC 使用说明

*ADC 共有 7 个 32 位寄存器，包括控制状态寄存器有 4 个，结果寄存器，比较值寄存器和引脚寄存器。
*配置好 ADC 模块的时钟，以及分频系数(AD_SC3)。
*配置好转换精度 8/10/12 位可选(默认 8bit)(AD_SC3）。
*选择转换模式为单次转换还是连续转换（默认单次）。
*选择转换是软件触发还是硬件触发（默认软件触发，只要对 ADC_SC1 写就触发一次）。
*采样时间配置(长采样/短采样)。
*转换结果比较设置。
*低功耗还是高速模式。
*设置好之后，就可以启动转换，如果是软件触发，则对 ADC_SC1 写就触发，如果是硬件触发，则等待硬件条件，通过中断或者轮询的方式获取结果。



注意事项

*ADC 有 2 个 FIFO：通道 FIFO，结果 FIFO。通道 FIFO 最大深度为 8，结果 FIFO 最大深度为 12。
*ADC 软件触发是指 AD_SC1 寄存器触发，非连续的，需要每次对寄存器写才触发。
*ADC 启动，功率控制，采样和转换时间，等详细信息请参阅参考手册。

1.1 模块初始化

函数名	ADC_Init
函数原形	ADC_Init(ADC_Type *pADC, ADC_ConfigTypePtr *pADC_Config)
功能描述	以配置结构体 pADC_Config 来初始化 ADC
输入参数	配置结构体 pADC_Config，模块结构体 ADC_Type
输出参数	无
返回值	无
先决条件	无
函数使用实例	先设置配置结构体，ADC_Init(ADC, &sADC_Config);

```
/*****************************************************************************//**
 *
 * @ 通过 pADC_Config 结构体来配置 ADC 模块，没有重新设置的，用默认值，是否打开中断也
 *在这里设置
 * @param[in]   ADC 指针
 * @param[in]   pADC_Config 配置结构体
 *
 * @return   无
 *
 * @ Pass/ Fail
 *****************************************************************************/
void ADC_Init(ADC_Type *pADC, ADC_ConfigTypePtr pADC_Config)
{
    if( pADC == ADC)
    {
        SIM->SCGC |= SIM_SCGC_ADC_MASK; //使能 ADC 时钟
    }
    ADC_SelectClock(pADC,pADC_Config->u8ClockSource); //配置时钟源
    ADC_SelectClockDivide(pADC,pADC_Config->u8ClockDiv);//时钟分频系数
    ADC_SetMode(pADC,pADC_Config->u8Mode); //精度选择
    ADC_SetFifoLevel(pADC,pADC_Config->u8FiFoLevel); //设置 FIFO 的深度，默认为 0
    pADC->APCTL1 = pADC_Config->u16PinControl; //
    if( pADC_Config->sSetting.bCompareEn )
    {
        ADC_CompareEnable(pADC); //使能转换结果比较
    }
    if( pADC_Config->sSetting.bCompareGreaterEn )
```

```
{
    ADC_CompareGreaterFunction(pADC); //是否启用比较功能
}
if( pADC_Config->sSetting.bContinuousEn )
{
    ADC_ContinuousConversion(pADC);//连续转换模式
}
if( pADC_Config->sSetting.bCompareAndEn )
{
    ADC_CompareFifoAnd(pADC);
}
if( pADC_Config->sSetting.bFiFoScanModeEn ) //FIFO SCAN 模式使能
{
    ADC_FifoScanModeEnable(pADC); //
}
if( pADC_Config->sSetting.bHardwareTriggerEn )
{
    ADC_SetHardwareTrigger(pADC);//设置为硬件触发模式
}
if( pADC_Config->sSetting.bIntEn )
{
    ADC_IntEnable(pADC); //使能中断
    NVIC_EnableIRQ( ADC0_IRQn );
}
if( pADC_Config->sSetting.bLongSampleEn )
{
    ADC_SetLongSample(pADC); //设置为长采样模式
}
if( pADC_Config->sSetting.bLowPowerEn )
{
    ADC_SetLowPower(pADC);//低功耗模式
}
}
```

1.2 单次转换

函数名	ADC_PollRead
函数原形	ADC_PollRead(ADC_Type *pADC, uint8_t u8Channel)
功能描述	单次 ADC 转换并返回 ADC 转换结果
输入参数	模块结构体 ADC_Type，要转换的通道 u8Channel
输出参数	无
返回值	转换结果
先决条件	无
函数使用实例	ADC_PollRead(ADC,ADC_CHANNEL_AD29_VREFH)

```
/*****************************************************************************//*!
    *
    * @库函数简介:启动一个 ADC 转换并返回转换结果
    *
    * @param[in]   ADC 指针
    * @param[in]   通道值
    *
    * @返回 ADC  转换结果
    *
    * @ Pass/ Fail criteria: none
    *****************************************************************************/
unsigned int ADC_PollRead( ADC_Type *pADC, uint8_t u8Channel )
{
        ADC_SetChannel(pADC,u8Channel); //设置要转换的通道值
        while( !ADC_IsCOCOFlag(pADC) ); //等待完成转换
        return ADC_ReadResultReg(pADC);
}
```

1.3 设置转换通道

函数名	ADC_SetChannel
函数原形	ADC_SetChannel(ADC_Type *pADC, uint8_t u8Channel)
功能描述	设置 ADC 转换通道，如果是 FIFO 模式，也通过这个函数对通道 FIFO 写
输入参数	模块结构体 ADC_Type，要设置的通道 u8Channel
输出参数	无
返回值	无
先决条件	无
函数使用实例	ADC_SetChannel(ADC, ADC_CHANNEL_AD22_TEMPSENSOR); ADC_SetChannel(ADC, ADC_CHANNEL_AD29_VREFH); ADC_SetChannel(ADC, ADC_CHANNEL_AD30_VREFL);

```
/*****************************************************************************//*!
    *
    * @设置 ADC 转换通道
    *
    * @param[in]   ADC 指针
    * @param[in]   设置通道
    *
    * @无返回
    *
    * @ Pass/ Fail criteria: none
    *****************************************************************************/
void ADC_SetChannel( ADC_Type *pADC, uint8_t u8Channel )
{
```

```
    uint32_t u32temp;
    u32temp = pADC->SC1; //先读取出 SC1 的值，设置通道值，不能影响原有的其他设置
    u32temp &= ~ADC_SC1_ADCH_MASK; //先把 SC1 channel 位清 0
    pADC->SC1 = u32temp|ADC_SC1_ADCH(u8Channel);   //或上通道值再赋给 SC1
}
```

第二章 样例程序

2.1 ADC_FIFO_demo

```
/*****************************************************************************
*
* 样例程序介绍
*该样例程序实现 FIFO 操作模式的 ADC 转换，FIFO 模式和单次模式基本一样
*转换在通道 FIFO 填充到 ADC_SC4[AFDEP]水平时，转换开始
*要注意：对通道 FIFO 读操作将读取当前活动通道值，对通道值 ADC_SC1[ADCH]写操作
*将会重新填充 FIFO 以开始新的转换
*
*****************************************************************************/

#include "common.h"
#include "ics.h"
#include "rtc.h"
#include "uart.h"
#include "adc.h"
#include "pmc.h"
#include "sysinit.h"


uint16_t u16ADC_ConversionBuff[16];
uint16_t u16ADC_ConversionCount = 0;
volatile uint8_t   u8ADC_ConversionFlag = 0;

int main (void);
void ADC_CallBack( void );

/*****************************************************************/
int main (void)
{
```

```
uint8_t              u8Ch;
ADC_ConfigType    sADC_Config = {0};

 sysinit(); //系统初始化

printf("\nRunning the ADC_FIFO_demo project.\n");

UART_WaitTxComplete(TERM_PORT);

/* initiaze ADC module */
sADC_Config.u8ClockDiv = ADC_ADIV_DIVIDE_8; //分频系数设置为 8
sADC_Config.u8ClockSource = CLOCK_SOURCE_ADACK; //时钟源为 ADACK
sADC_Config.u8Mode = ADC_MODE_12BIT; //设置为 12 位模式
sADC_Config.sSetting.bIntEn = 1; //使能中断
sADC_Config.u8FiFoLevel = ADC_FIFO_LEVEL3; //fifo 深度为 3
ADC_SetCallBack(ADC_CallBack);
ADC_Init( ADC, &sADC_Config);
 //对 ADC 进行初始化，这个时候转化没有开始，必须把通道 FIFO 填充到 3 级时，才开始一次转换
//默认情况下为软件触发，只要对 ADC_SC1 寄存器写，就会引发一次转换


/* echo chars received from terminal */
while(1)
{
    /// set channel to start a new conversion /
    u8ADC_ConversionFlag = 0;
    //ADC FIFO 操作单次转换模式，需要每次对通道 FIFO 写满才引发一次转换

    ADC_SetChannel(ADC,ADC_CHANNEL_AD22_TEMPSENSOR);
    ADC_SetChannel(ADC,ADC_CHANNEL_AD29_VREFH);
    ADC_SetChannel(ADC,ADC_CHANNEL_AD30_VREFL);
    //设置好通道 FIFO 之后，转换开始
    //转换结束后
    while( !u8ADC_ConversionFlag);
    //输出转换值
    printf("ADC conversion result as below:\n");
    for( u8Ch =0 ;u8Ch< u16ADC_ConversionCount; u8Ch ++)
    {
        printf("0x%x,",u16ADC_ConversionBuff[u8Ch]);
    }
    printf("\n");
    printf("input any character to start a new conversion!\n");
    u8Ch = UART_GetChar(TERM_PORT);
    u16ADC_ConversionCount = 0;
```

```
    }

}

/*****************************************************************************
* @中断服务程序
*
* @ADC  中断服务函数
*            every ADC interrupt.
*
* @param    none
*
* @无返回
*
* @ Pass/ Fail criteria: none
*****************************************************************************/

void ADC_CallBack( void )
{
    //如果是直接用 isr，而不用回调，第一个事情应该是清中断标志位
    uint8_t              u8Ch;
    while( !ADC_IsFIFOEmptyFlag(ADC) ) //如果 fifo 不为空
    {
        if( u16ADC_ConversionCount < 16 )
        //如果转换结果计数器小于 16，把结果从 FIFO 中读取出来放在数组内。
        {
            u16ADC_ConversionBuff[u16ADC_ConversionCount++] = ADC_ReadResultReg(ADC);
        }
        else
        {
            ADC_ReadResultReg(ADC);
        }
        // 通过串口输出结果。
            for( u8Ch =0 ;u8Ch< u16ADC_ConversionCount; u8Ch ++)
        {
            printf("0x%x,",u16ADC_ConversionBuff[u8Ch]);
        }
    }
    u8ADC_ConversionFlag = 1;
}
```

2.2 ADC 中断样例程序

```
/*****************************************************************************
*
* @ADC 中断样例程序
*每次转换发生在 ADC_SC1 寄存器被设置之后，转换完成之后都会引发一次 ADC 中断
*样例程序，adc 中断服务程序把 adc 转换值放入数组，然后通过串口输出
*
*****************************************************************************/

#include "common.h"
#include "ics.h"
#include "rtc.h"
#include "uart.h"
#include "adc.h"
#include "sysinit.h"


uint16_t u16ADC_ConversionBuff[16];
uint16_t u16ADC_ConversionCount = 0;
volatile uint8_t    u8ADC_ConversionFlag = 0;
int main (void);
void ADC_CallBack( void );


/*********************************************************************/
int main (void)
{
    uint8_t             u8Ch;
    ADC_ConfigType    sADC_Config = {0};

   //系统初始化
    sysinit();

    printf("\nRunning the ADC_Int_demo project.\n");

    UART_WaitTxComplete(TERM_PORT);

    /* initiaze ADC module */
    sADC_Config.u8ClockDiv = ADC_ADIV_DIVIDE_4; //分频系数为 4，时钟不超过 8M
    sADC_Config.u8ClockSource = CLOCK_SOURCE_BUS_CLOCK; //ADC 时钟
    sADC_Config.u8Mode = ADC_MODE_12BIT; //12bit 模式
```

```
    sADC_Config.sSetting.bIntEn = 1; //使能中断

    ADC_SetCallBack(ADC_CallBack);
    ADC_Init( ADC, &sADC_Config);
    //初始化 ADC
    /* echo chars received from terminal */
    while(1)
    {
        /* 每次设置完 ADC_SC1，都会引发一次 ADC 转换*/
        u8ADC_ConversionFlag = 0;
        ADC_SetChannel(ADC,ADC_CHANNEL_AD22_TEMPSENSOR);
        /* 转换完成之后，中断程序读取 ADC 转换值至 数组 */
        while( !u8ADC_ConversionFlag);

          /* 每次设置完 ADC_SC1，都会引发一次 ADC 转换*/
        u8ADC_ConversionFlag = 0;
        ADC_SetChannel(ADC,ADC_CHANNEL_AD29_VREFH);
       /* 转换完成之后，中断程序读取 ADC 转换值至 数组 */
        while( !u8ADC_ConversionFlag);

          /* 每次设置完 ADC_SC1，都会引发一次 ADC 转换*/
        u8ADC_ConversionFlag = 0;
        ADC_SetChannel(ADC,ADC_CHANNEL_AD30_VREFL);
       /* 转换完成之后，中断程序读取 ADC 转换值至 数组 */
        while( !u8ADC_ConversionFlag);

        printf("ADC conversion result as below:\n");
        for( u8Ch =0 ;u8Ch< u16ADC_ConversionCount; u8Ch ++)
        {

            printf("0x%x,",u16ADC_ConversionBuff[u8Ch]);
        }
        printf("\n");
        printf("input any character to start a new conversion!\n");
        u8Ch = UART_GetChar(TERM_PORT);
        u16ADC_ConversionCount = 0;
    }

}
```

```
/*****************************************************************************
* @中断服务程序
*
* @brief callback routine of ADC driver which does what you want to do at
*             every ADC interrupt.
*
* @param    none
*
* @return none
*
* @ Pass/ Fail criteria: none
*****************************************************************************/

void ADC_CallBack( void )
{
//如果是直接用 isr，而不用回调，第一个事情应该是清中断标志位
    if( u16ADC_ConversionCount < 16 )
    {
        u16ADC_ConversionBuff[u16ADC_ConversionCount++] = ADC_ReadResultReg(ADC);
    }

     u8ADC_ConversionFlag = 1;
}
```

2.3 ADC 单次转换模式

```
/*****************************************************************************
*
* @brief ADC  单次转换模式，不需要中断，每次转换完等待完成标识位 COCO. 然后读取出
*ADC  转换值
*
*****************************************************************************/

#include "common.h"
#include "ics.h"
#include "rtc.h"
#include "uart.h"
#include "adc.h"
#include "sysinit.h"



int main (void);
```

```
/*****************************************************************************
* Global functions
*****************************************************************************/


/*****************************************************************/
int main (void)
{
    ADC_ConfigType    sADC_Config = {0};

    /* Perform processor initialization */
    sysinit();
    printf("\nRunning the ADC_Poll_demo project.\n");

    UART_WaitTxComplete(TERM_PORT);

    /* 初始化 ADC 模块，*/
    sADC_Config.u8ClockDiv = ADC_ADIV_DIVIDE_8;
    sADC_Config.u8ClockSource = CLOCK_SOURCE_BUS_CLOCK;
    sADC_Config.u8Mode = ADC_MODE_12BIT;
    ADC_Init( ADC, &sADC_Config);

    /* 每次把 ADC 转换值输出到串口 */
    while(1)
    {
        printf("VREFH conversion value:0x%x\n",ADC_PollRead(ADC,ADC_CHANNEL_AD29_VREFH));
        printf("VREFL conversion value:0x%x\n",ADC_PollRead(ADC,ADC_CHANNEL_AD30_VREFL));
        printf("Temperature sensor conversion
value:0x%x\n",ADC_PollRead(ADC,ADC_CHANNEL_AD22_TEMPSENSOR));

        printf("input any character to start a new conversion!\n");
        UART_GetChar(TERM_PORT);

    }

}
```