

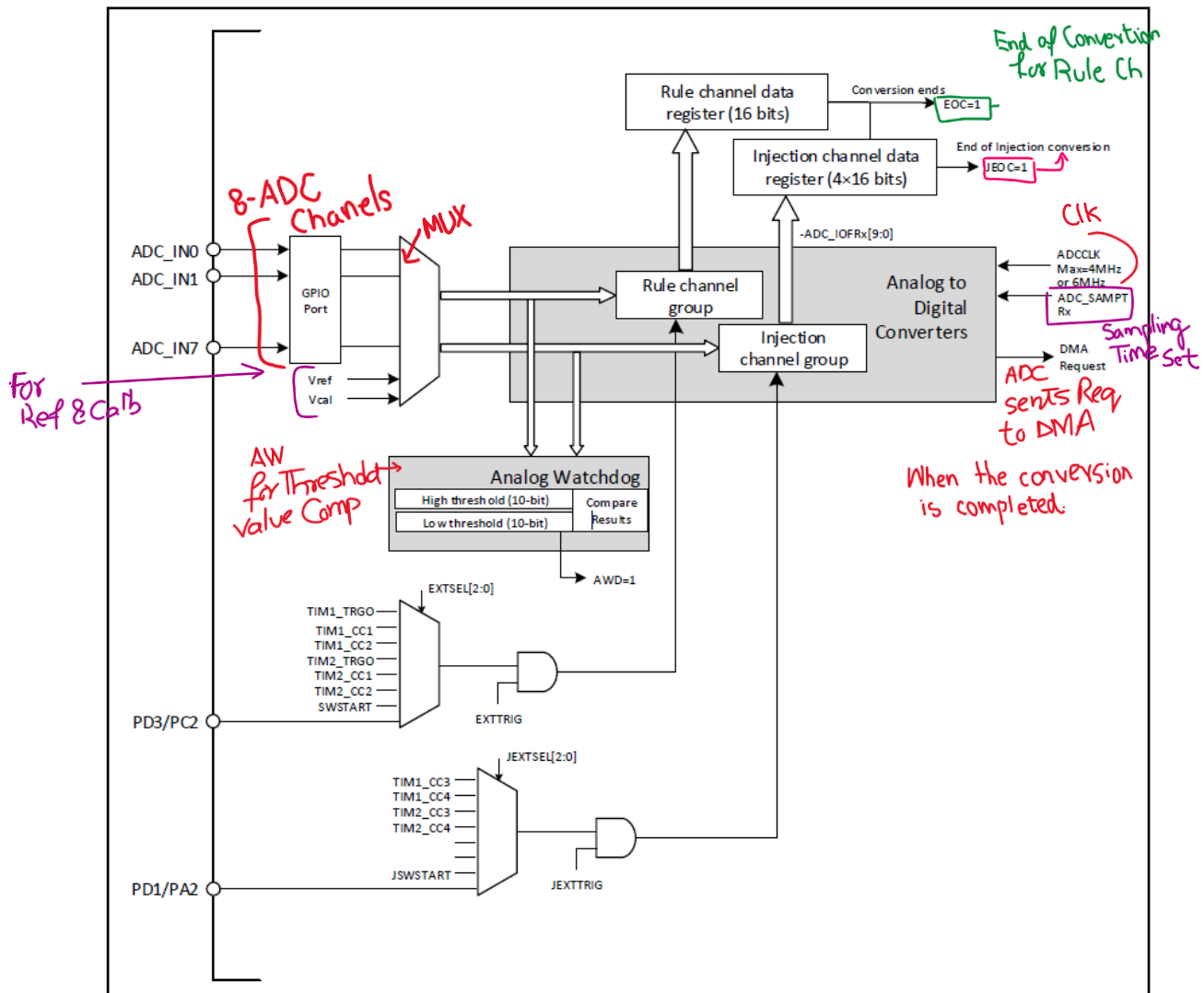
Chapter 4: Analog to Digital Conversion (ADC)

4.1 Main Features:

- **10-bit** resolution
- Supports **8 external** channels and **2 internal** signal sources for sampling
- Multiple sampling conversion methods for multiple channels: **single, continuous, scan, trigger, intermittent**, etc.
- Data alignment modes: **left-aligned, right-aligned**
- Sampling time can be programmed separately by channel
- Both **rule conversion** and **injection conversion** support external triggering
- Analog watchdog to monitor channel voltage*, self-calibration function
- ADC channel input range: $0 \leq V_{IN} \leq V_{DDA}$ **
- Trigger delay

* The channel voltage can be monitored to see if it is within the threshold range by using the analog watchdog function

** $V_{DD} = 2.7 \sim 5.5V$: Power supply for some I/O pins and internal voltage regulator (V_{DD} performance decreases if less than 2.9V when using ADC).



4.2 ADC Configuration:

4.2.1) Module Power up:

An **ADON bit** of 1 in the **ADC_CTLR2 register** indicates that the ADC module is **powered up**. When the ADC module enters the power-up state (ADON=1) from the power-down mode (ADON=0), **a delay period t_{STAB} is required for the module stabilization time**. After that, the ADON bit is written to 1 again and is used as the **start signal** for software to start the ADC conversion. By clearing the **ADON bit to 0**, the current conversion can be terminated and the ADC module placed in **power-down mode**, a state in which the ADC consumes almost **no power**.

4.2.2) Sampling clock:

The register operation of the module is based on the **AHBCLK (AHB bus) clock**, and the clock reference of its conversion unit, ADCCLK, is configured by the **ADCPRE field of the RCC_CFGR0 register** to divide the frequency.

4.2.3) Channel configuration

The ADC module provides 10 channel sampling sources, including 8 external channels and 2 internal channels. They can be configured into two types of conversion groups: regular groups and injection groups. to achieve a group conversion consisting of a series of conversions in any order on any number of channels.

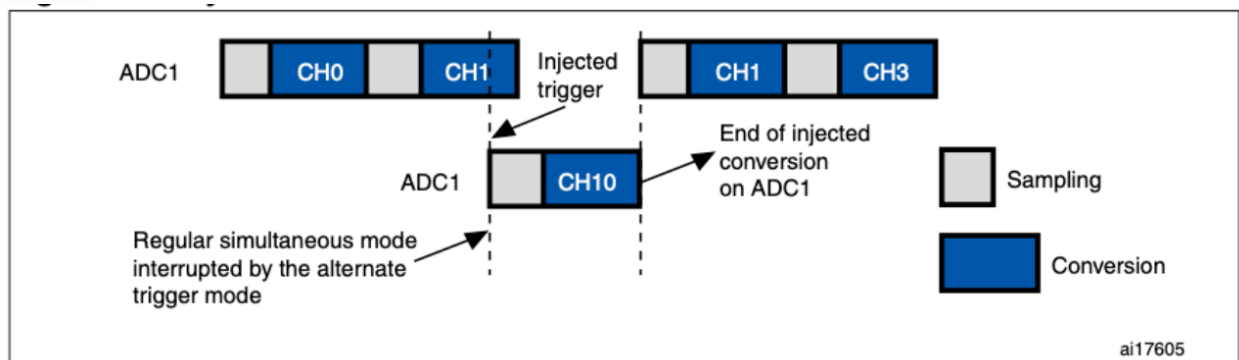
Conversion group.

- **Rule group:** consists of up to **16 conversions**. The rule channels and their *conversion order are set in the ADC_RSQRx register*. The total number of conversions in the rule group should be written to **L[3:0]** in the ADC_RSQR1 register.
- **Injection group:** consists of up to **4 conversions**. *The injection channels and the order of their conversions are set in the ADC_ISQR register*. The total number of conversions in the injection group should be written in **JL[1:0]** of the ADC_ISQR register.

Note: If the ADC_RSQRx or ADC_ISQR registers are changed during conversion, the current conversion is terminated and a new start signal is sent to the ADC to convert the newly selected group.

→ Difference between rule and injection modes:

Injected conversions can be triggered by software or by hardware (timers or external pins). Injected conversions have **higher priority** and they can **interrupt regular conversions immediately**. A use case I can think of is that we run regular sequence conversion in a loop or something and interrupt that when needed.



[https://lonesometraveler.github.io/2020/05/15/ADC-Injected-Conversion.html#:~:text=The%20injected%20group%20has%20priority,\(timers%20or%20external%20pins\).](https://lonesometraveler.github.io/2020/05/15/ADC-Injected-Conversion.html#:~:text=The%20injected%20group%20has%20priority,(timers%20or%20external%20pins).)

2 internal channels.

- **Vref** internal reference voltage: connected to **ADC_IN8** channel.
- **Vcal** internal calibration voltage: connected to **ADC_IN9** channel, 2 steps selectable.

4.2.4 Calibration:

The ADC has a built-in self-calibration mode. A calibration session significantly reduces accuracy errors due to variations in the internal capacitor banks. During calibration, an error correction code is calculated

on each capacitor, which is used to eliminate the errors generated on each capacitor in subsequent conversions.

Steps For Calibration:

1. Initialize the calibration register by writing **RSTCAL position 1 of ADC_CTLR2** register and wait for **RSTCAL hardware to clear 0** to indicate the **completion of initialization**.
2. Set the **CAL bit** to **start the calibration function**. Once the calibration is finished, the hardware will automatically **clear the CAL** bit and store the calibration code into **ADC_RDATAR**.
3. After that, the normal conversion function can be started. *It is recommended to perform an ADC calibration when the ADC module is **powered up**.*

Note: Before starting the calibration, you must ensure that the ADC module is in the power-up state (ADON=1) for more than at least two ADC clock cycles.

4.2.5 Programmable sampling time:

The ADC uses several ADCCLK cycles to sample the input voltage. **The number of sampling cycles for a channel can be changed using the SMPx[2:0] bits in the ADC_SAMPTR1 and ADC_SAMPTR2 registers.**

Each channel can be sampled separately using a different time. The total conversion time is calculated as follows.

$$T_{\text{CONV}} = \text{sampling time} + 11T_{\text{ADCCLK}}$$

The ADC's rule channel conversion supports the DMA function. The value of the rule channel conversion is stored in a data-only register, ADC_RDATAR.

To prevent the data in ADC_RDATAR register from being fetched in time when multiple rule channels are converted in succession, the DMA function of ADC can be enabled.

The hardware will generate a DMA request at the end of the conversion of a rule channel (EOC set) and transfer the converted data from the ADC_RDATAR register to the user-specified destination address.

After the channel configuration of the DMA controller module is completed, **write DMA position 1 of the ADC_CTLR2 register to enable the DMA function of the ADC.**

Note: Injection group conversion does not support DMA function.

4.2 ADC Read Example Code with DMA:

```
/****** (C) COPYRIGHT *****  
*****  
* File Name      : main.c  
* Author        : WCH  
* Version       : V1.0.0  
* Date         : 2022/08/08  
* Description    : Main program body.  
  
*****  
****  
* Copyright (c) 2021 Nanjing Qinheng Microelectronics Co., Ltd.  
* Attention: This software (modified or not) and binary are used for  
* microcontroller manufactured by Nanjing Qinheng Microelectronics.  
  
*****  
**/  
  
/*  
* @Note  
* ADC DMA sampling routine:  
* ADC channel 2 (PC4), the rule group channel obtains ADC conversion data for  
10 consecutive  
* times through DMA.  
*  
*/  
  
#include "debug.h"  
  
/* Global Variable */  
u16 TxBuf[10];  
  
/******  
* @fn      ADC_Function_Init  
*  
* @brief    Initializes ADC collection.  
*  
* @return   none  
*/  
void ADC_Function_Init(void)  
{  
    ADC_InitTypeDef  ADC_InitStructure = {0};  
    GPIO_InitTypeDef GPIO_InitStructure = {0};  
  
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);  
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);  
    RCC_ADCCLKConfig(RCC_PCLK2_Div8);  
  
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;  
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;  
    GPIO_Init(GPIOC, &GPIO_InitStructure);  
  
    ADC_DeInit(ADC1);  
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
```

```

    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfChannel = 1;
    ADC_Init(ADC1, &ADC_InitStructure);

    ADC_Calibration_Vol(ADC1, ADC_CALVOL_50PERCENT);
    ADC_DMACmd(ADC1, ENABLE);
    ADC_Cmd(ADC1, ENABLE);

    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));
}

/*****
 * @fn      Get_ADC_Val
 *
 * @brief    Returns ADCx conversion result data.
 *
 * @param    ch - ADC channel.
 *            ADC_Channel_0 - ADC Channel0 selected.
 *            ADC_Channel_1 - ADC Channel1 selected.
 *            ADC_Channel_2 - ADC Channel2 selected.
 *            ADC_Channel_3 - ADC Channel3 selected.
 *            ADC_Channel_4 - ADC Channel4 selected.
 *            ADC_Channel_5 - ADC Channel5 selected.
 *            ADC_Channel_6 - ADC Channel6 selected.
 *            ADC_Channel_7 - ADC Channel7 selected.
 *            ADC_Channel_8 - ADC Channel8 selected.
 *            ADC_Channel_9 - ADC Channel9 selected.
 *
 * @return    none
 */
u16 Get_ADC_Val(u8 ch)
{
    u16 val;

    ADC-RegularChannelConfig(ADC1, ch, 1, ADC_SampleTime_241Cycles);
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);

    while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC));
    val = ADC_GetConversionValue(ADC1);

    return val;
}

/*****
 * @fn      DMA_Tx_Init
 *
 * @brief    Initializes the DMAy Channelx configuration.
 *
 * @param    DMA_CHx - x can be 1 to 7.
 *            ppadr - Peripheral base address.
 *            memadr - Memory base address.
 */

```

```

*          bufsize - DMA channel buffer size.
*
* @return  none
*/
void DMA_Tx_Init(DMA_Channel_TypeDef *DMA_CHx, u32 ppadr, u32 memadr, u16
bufsize)
{
    DMA_InitTypeDef DMA_InitStructure = {0};

    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);

    DMA_DeInit(DMA_CHx);
    DMA_InitStructure.DMA_PeripheralBaseAddr = ppadr;
    DMA_InitStructure.DMA_MemoryBaseAddr = memadr;
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
    DMA_InitStructure.DMA_BufferSize = bufsize;
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_PeripheralDataSize =
DMA_PeripheralDataSize_HalfWord;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
    DMA_InitStructure.DMA_Priority = DMA_Priority_VeryHigh;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
    DMA_Init(DMA_CHx, &DMA_InitStructure);
}

/*****
* @fn      main
*
* @brief    Main program.
*
* @return   none
*/
int main(void)
{
    u16 i;
    SystemCoreClockUpdate();
    Delay_Init();
#ifdef (SDI_PRINT == SDI_PR_OPEN)
    SDI_Printf_Enable();
#else
    USART_Printf_Init(115200);
#endif
    printf("SystemClk:%d\r\n", SystemCoreClock);
    printf("ChipID:%08x\r\n", DBGMCU_GetCHIPID());

    ADC_Function_Init();

    DMA_Tx_Init(DMA1_Channel1, (u32)&ADC1->RDATAR, (u32)TxBuf, 10);
    DMA_Cmd(DMA1_Channel1, ENABLE);

    ADC_RegularChannelConfig(ADC1, ADC_Channel_2, 1,
ADC_SampleTime_241Cycles);
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
    Delay_Ms(50);
    ADC_SoftwareStartConvCmd(ADC1, DISABLE);

```

```

        for(i = 0; i < 10; i++){
            printf("%04d\r\n", TxBuf[i]);
            Delay_Ms(10);
        }

        while(1);
}

```

4.3 ADC Read Example code without DMA:

```

#include "debug.h"

/* Global Variable */
u16 adc_values[10]; // Array to store 10 ADC readings

/*****
 * @fn          ADC_Function_Init
 *
 * @brief       Initializes ADC collection.
 *
 * @return      none
 */
void ADC_Function_Init(void)
{
    ADC_InitTypeDef  ADC_InitStructure = {0};
    GPIO_InitTypeDef GPIO_InitStructure = {0};

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
    RCC_ADCCLKConfig(RCC_PCLK2_Div8);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    ADC_DeInit(ADC1);
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE; // Enable continuous
conversion
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfChannel = 1;
    ADC_Init(ADC1, &ADC_InitStructure);

    ADC_Calibration_Vol(ADC1, ADC_CALVOL_50PERCENT);
    ADC_Cmd(ADC1, ENABLE);

    ADC_ResetCalibration(ADC1);
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));
}

```



```

/*****
 * @fn      Get_ADC_Val
 *
 * @brief    Reads and returns the ADC conversion value for a specific
channel.
 *
 * @param    ch - ADC channel (unused in this modified version)
 *
 * @return   ADC conversion value (u16)
 */
u16 Get_ADC_Val(u8 ch) // Unused parameter ch removed
{
    ADC-RegularChannelConfig(ADC1, ADC_Channel_2, 1,
ADC_SampleTime_241Cycles); // Select channel 2
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);

    while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC));
    return ADC_GetConversionValue(ADC1);
}

/*****
 * @fn      main
 *
 * @brief    Main program.
 *
 * @return   none
 */
int main(void)
{
    u16 i;
    SystemCoreClockUpdate();
    Delay_Init();
#ifdef SDI_PRINT == SDI_PR_OPEN
    SDI_Printf_Enable();
#else
    USART_Printf_Init(115200);
#endif
    printf("SystemClk:%d\r\n", SystemCoreClock);
    printf("ChipID:%08x\r\n", DBGMCU_GetCHIPID());

    ADC_Function_Init();

    for (i = 0; i < 10; i++) {
        adc_values[i] = Get_ADC_Val(0); // Read and store ADC value (ch
parameter removed)
        printf("%04d\r\n", adc_values[i]);
        Delay_Ms(10);
    }

    while (1) {
        // You can implement further processing of the ADC values stored in
the adc_values array here
    }
}

```