ADC多通道单一转换

**实验目标：**实现ADC多通道使用，即在同一个工程下，实现多种ADC功能。单一转换，即虽然用到多个输入通道，但是其并不是一起转换，而是ADC资源轮换使用。

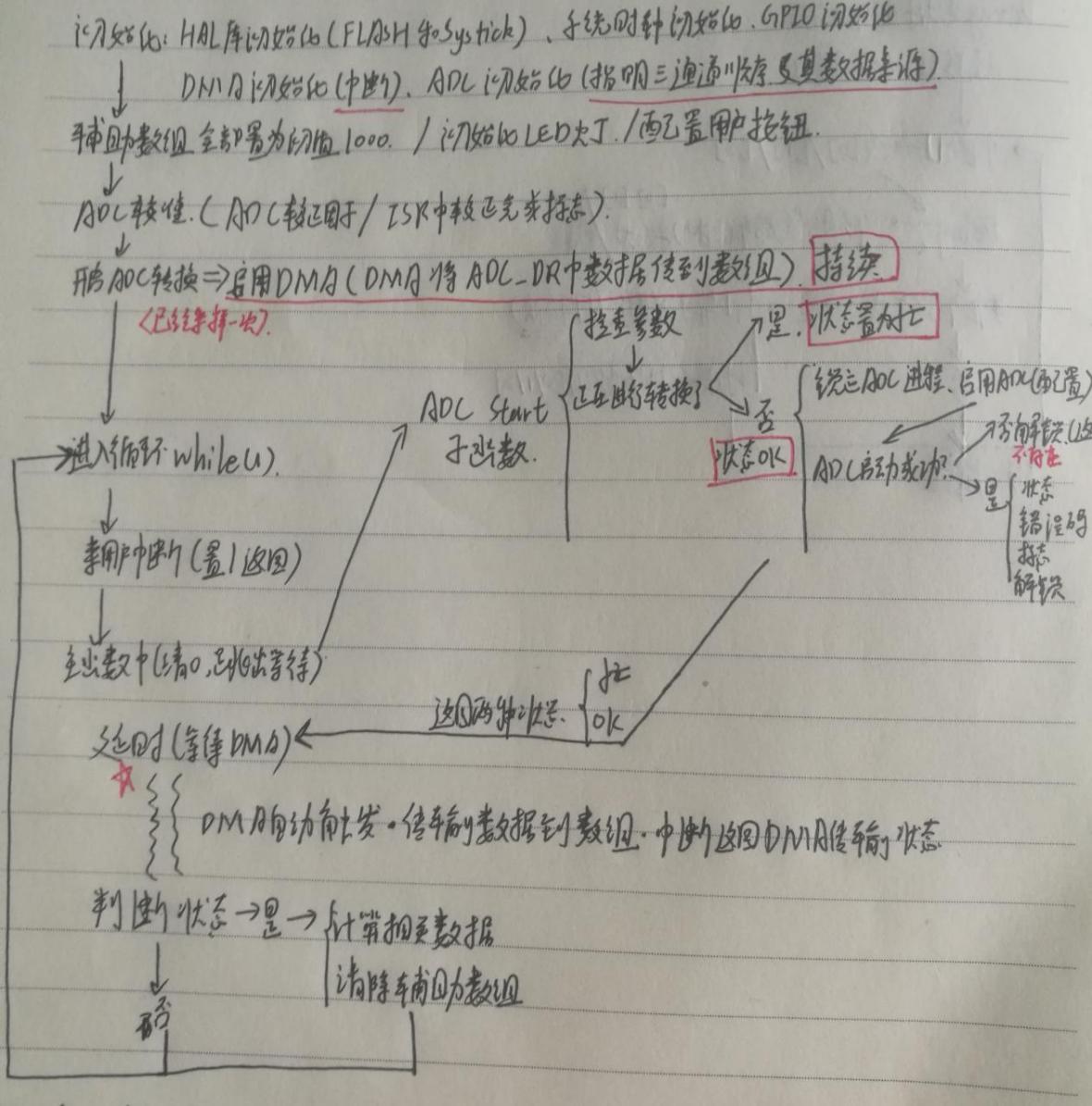
**第一步：**根据ADC与DMA设计实验思路

ADC数据来源：内部配置的温度传感器以及用PA4引脚测量外部电压。

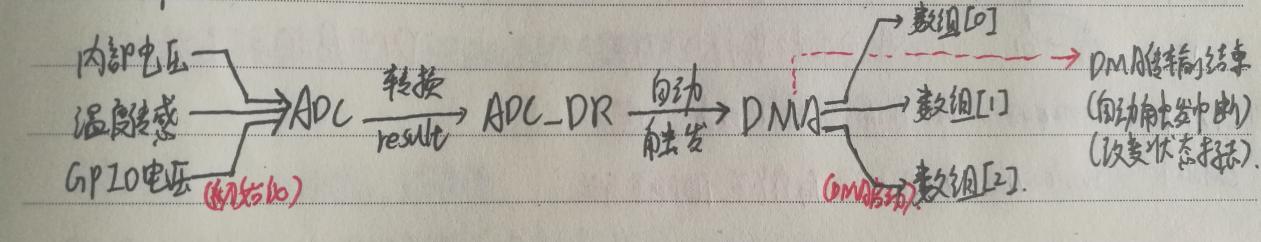
ADC相关寄存器：中断状态寄存器ISR，中断使能寄存器IER，控制寄存器CR，配置寄存器CFGR1，采样时间寄存器SMPR，通道选择寄存器CHSELR，数据寄存器DR，校正因子CALFACT，公共配置寄存器CCR。

DMA相关寄存器：通道配置寄存器CCR1，通道数据转换寄存器CNDTR1，通道外部设备寄存器，通道存储器地址寄存器。

程序执行流程如下：



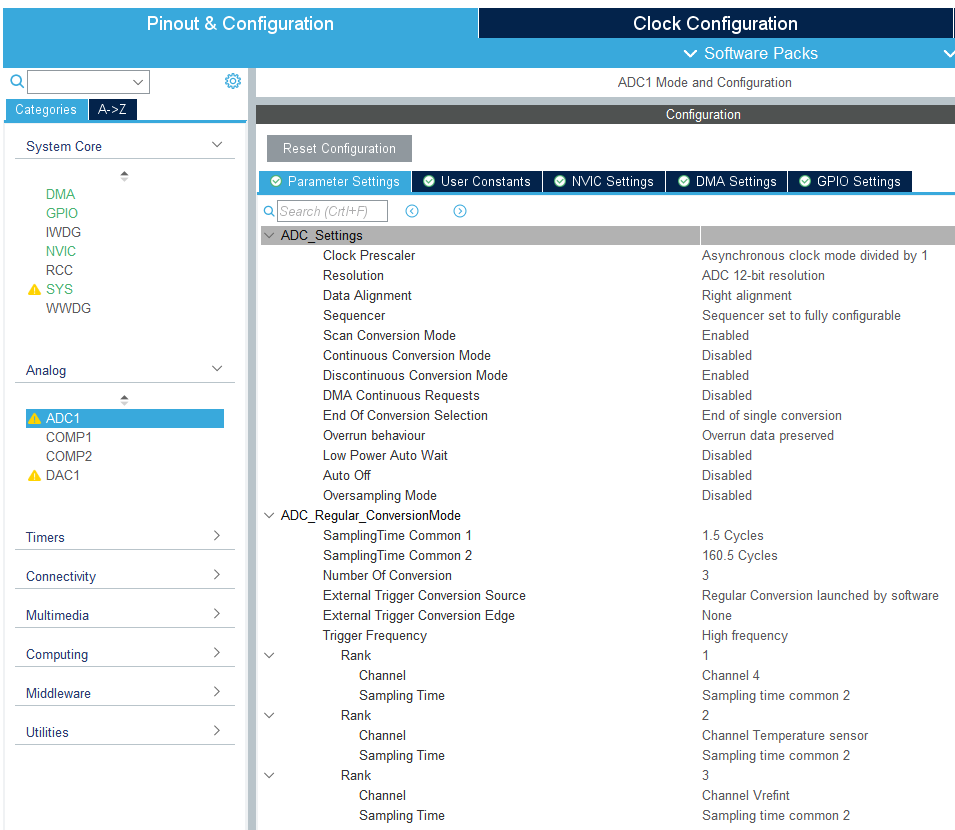
硬件各个通道工作方式与自动触发操作图解：



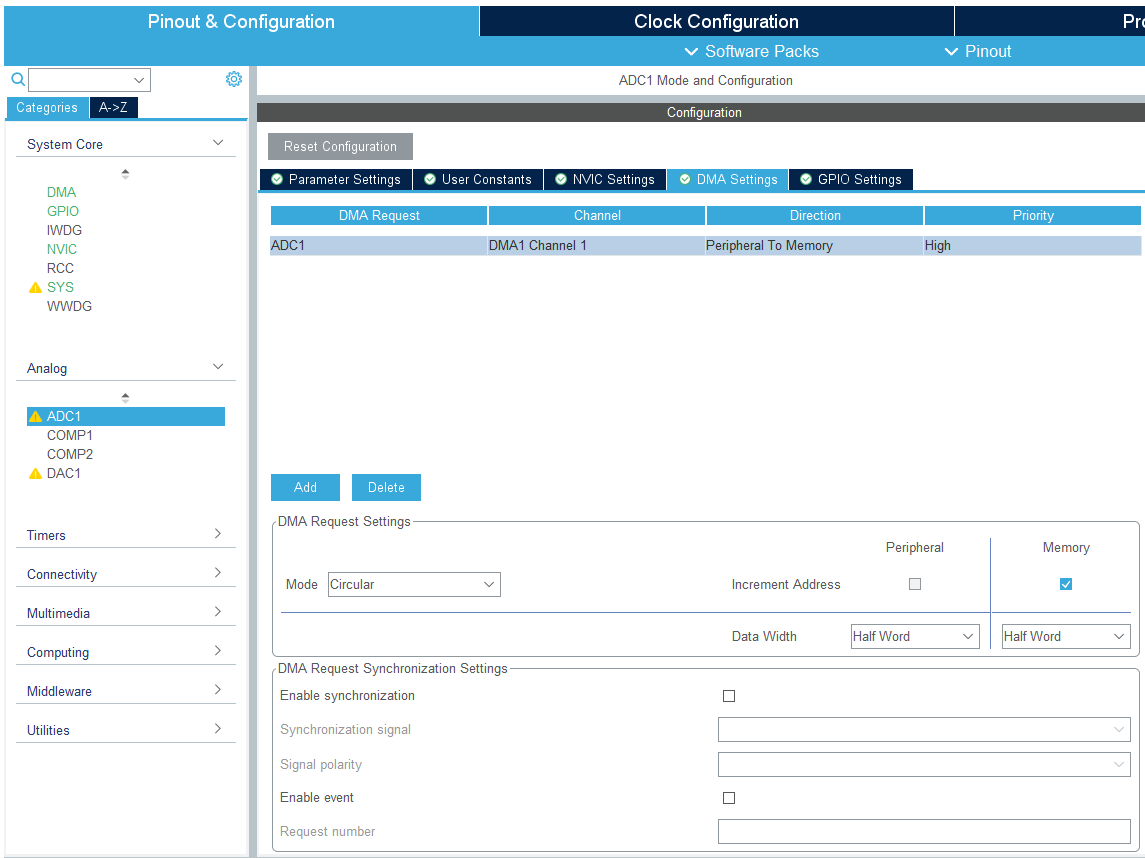
**第二步：**配置相关模块工作状态

将PA5设置为通用输出模式（即LED提示灯），将PC13设置为外部中断、下降沿触发并且使能该中断。

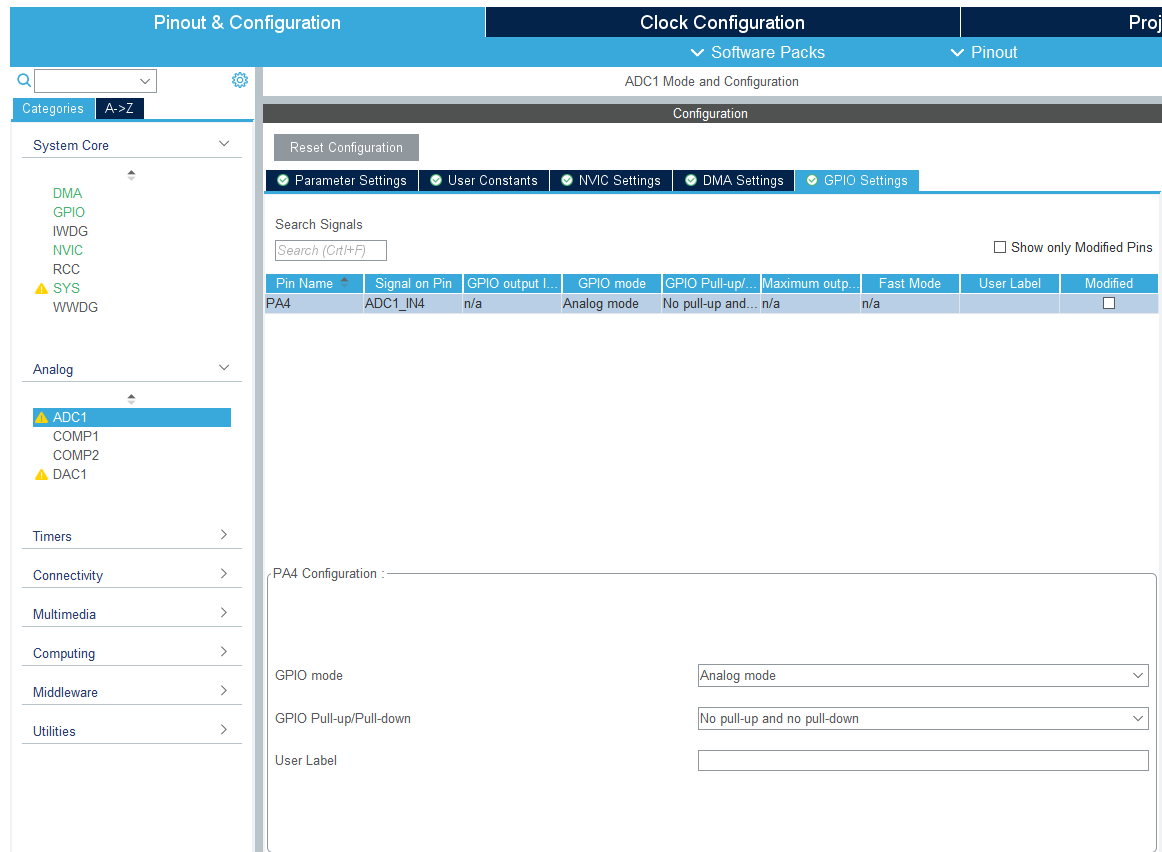
在ADC1模式与配置页面，设置其工作模式的多个通道：通道4，温度传感器通道和内部电压通道（简单无图）。接下来设置ADC系统预分频，将其工作模式设置为扫描模式、不连续模式；再配置扫描模式下，三个通道的排序（依次轮转）；最后设置每个通道的抽样时间。如下图：



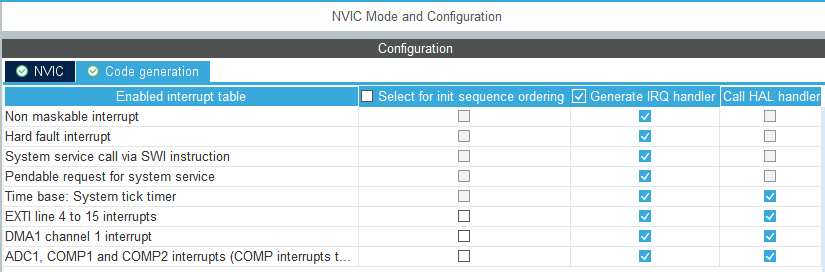
配置完ADC工作模式后，点击旁边的“DMA设置选项”，让DMA与ADC协同工作。如下图：



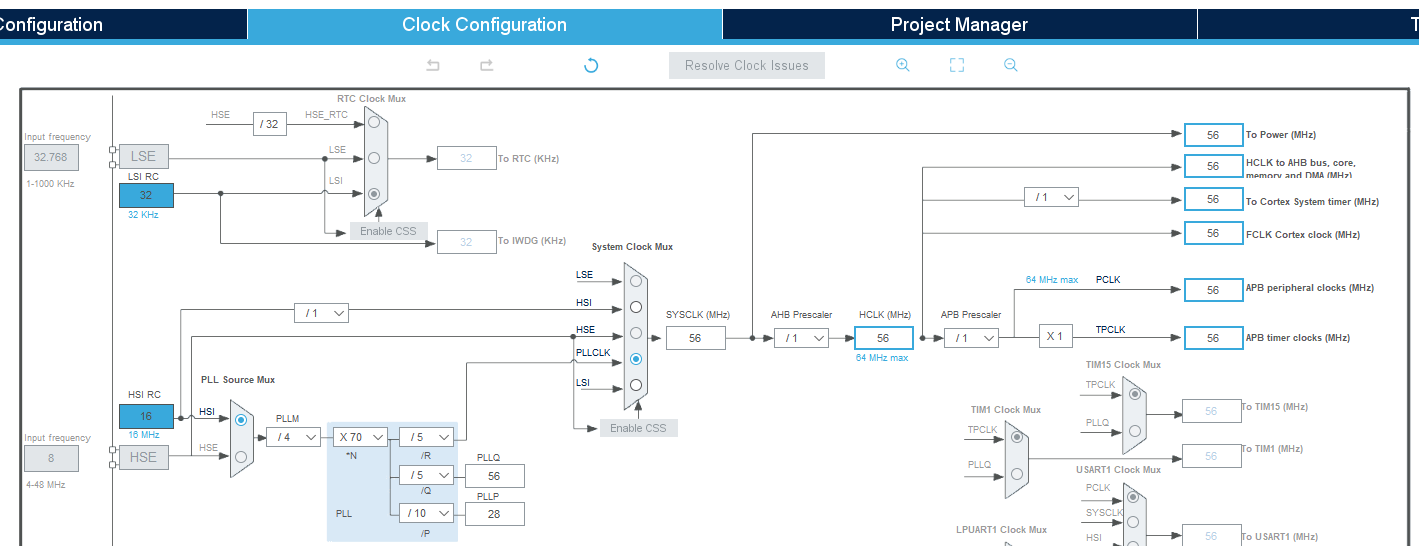
然后配置GPIO端口的工作模式，将其设置为“模拟模式”来获取外部电压（模拟量），如下图：



打开NVIC设置，将对应中断使能。



最后，按照下图所示配置时钟：



**第三步：**设置工程名、保存路径等，生成初始代码

**第四步：**添加设计代码

**①**变量定义、函数声明

/\* 环境模拟值的定义 \*/

/\* 模拟参考电压值(Vref+)，连接模拟电压 \*/

/\* 供电侧 (单位: mV). \*/

#define VDDA\_APPLI (3300UL)

#define ADC\_CONVERTED\_DATA\_BUFFER\_SIZE (3)

\_\_IO uint16\_t aADCxConvertedData[ADC\_CONVERTED\_DATA\_BUFFER\_SIZE]; /\* ADC转换数据(数据数组) 大小为3 \*/

/\* 用ADC转换数据计算得到物理值 \*/

\_\_IO uint16\_t uhADCxConvertedData\_VoltageGPIO\_mVolt = 0U; /\* GPIO管脚(对应ADC通道)上的电压值(单位:mV) \*/

\_\_IO uint16\_t uhADCxConvertedData\_VrefInt\_mVolt = 0U; /\* 内部电压参考VrefInt值(单位:mV) \*/

\_\_IO int16\_t hADCxConvertedData\_Temperature\_DegreeCelsius = 0U; /\* 温度值(单位:摄氏度) \*/

\_\_IO uint16\_t uhADCxConvertedData\_VrefAnalog\_mVolt = 0U; /\* 模拟参考电压值(Vref+)，连接模拟电压电源Vdda，从ADC转换数据计算(单位:mV) \*/

/\* ADC转换数据的DMA传输状态 \*/

/\* 0: DMA 传输没有完成 \*/

/\* 1: DMA 传输完成 \*/

/\* 2: DMA 尚未开始传输(初始状态) \*/

\_\_IO uint8\_t ubDmaTransferStatus = 2; /\*变量集到DMA中断回调 \*/

\_\_IO uint8\_t ubUserButtonClickEvent = RESET; /\* 事件检测:用户按下中断后置一，松开后清零 \*/

**②**写中断标志位处理的相关函数，并注释掉原有的4-15号外部中断入口

ITStatus EXTI\_GetITStatus(uint32\_t EXTI\_Line)

{

ITStatus bitstatus = RESET;

uint32\_t enablestatus = 0;

assert\_param(IS\_GET\_EXTI\_LINE(EXTI\_Line));

enablestatus = EXTI->IMR1 & EXTI\_Line;

if (((EXTI->FPR1 & EXTI\_Line) != (uint32\_t)RESET) && (enablestatus != (uint32\_t)RESET))

bitstatus = SET;

else

bitstatus = RESET;

return bitstatus;

}

void EXTI\_ClearITPendingBit(uint32\_t EXTI\_Line)

{

assert\_param(IS\_EXTI\_LINE(EXTI\_Line));

EXTI->FPR1 = EXTI\_Line;

}

**③**在mian主函数中，进入循环工作模式之前，需要初始化ADC输出的数组，关闭LED4状态提示灯，进行ADC校准，使用DMA启用ADC常规数据转换（DMA协同ADC工作）

uint32\_t tmp\_index\_adc\_converted\_data = 0; //数组赋值辅助变量

for (tmp\_index\_adc\_converted\_data = 0; tmp\_index\_adc\_converted\_data < ADC\_CONVERTED\_DATA\_BUFFER\_SIZE; tmp\_index\_adc\_converted\_data++)

aADCxConvertedData[tmp\_index\_adc\_converted\_data] = 0x1000; //数组值初始化

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_5,RESET); //LED初始状态灭

/\* 运行ADC校准 \*/

if (HAL\_ADCEx\_Calibration\_Start(&hadc1) != HAL\_OK)

Error\_Handler(); /\* 校准错误 \*/

/\*## 开始ADC转换 ###############################################\*/

/\* 使用DMA启动ADC组常规转换 \*/

if (HAL\_ADC\_Start\_DMA(&hadc1,(uint32\_t \*)aADCxConvertedData,

ADC\_CONVERTED\_DATA\_BUFFER\_SIZE) != HAL\_OK)

Error\_Handler(); /\* ADC转换启动错误 \*/

**④**进入while(1)循环后执行以下内容

/\* 等待用户按下按键执行以下操作 \*/

while (ubUserButtonClickEvent == RESET)

{

}

/\* 清除按下标志位 \*/

HAL\_Delay(200);

ubUserButtonClickEvent = RESET;

/\* 开始ADC转换 \*/

// 由于排序器是在不连续模式下启用的，这将执行转换的下一个排名在排序器。

// 注: 本例由软件启动触发转换。因此，每次转换都必须调用"HAL\_ADC\_Start()".

/\* DMA传输已经由函数"HAL\_ADC\_Start\_DMA()"在之前发起, 此函数保持DMA传输持续运行 \*/

if (HAL\_ADC\_Start(&hadc1) != HAL\_OK)

{

Error\_Handler();

}

/\* 延时以等待ADC转换和DMA传输完成(更新变量ubdmateransferstatus) \*/

HAL\_Delay(1000);

/\* 检查ADC是否转换了序列的所有级别 \*/

if (ubDmaTransferStatus == 1) //如果DMA传输完成

{

/\* ADC的计算将原始数据转换为物理值 \*/

/\* 使用LL ADC驱动程序帮助宏。 \*/

/\* Note: ADC结果被转移到数组“aADCxConvertedData” \*/

/\* 在他们的排名顺序在ADC序列。 \*/

uhADCxConvertedData\_VoltageGPIO\_mVolt = \_\_LL\_ADC\_CALC\_DATA\_TO\_VOLTAGE(VDDA\_APPLI, aADCxConvertedData[0], LL\_ADC\_RESOLUTION\_12B); //（供电电压\*数组数据/分辨率----量化）

uhADCxConvertedData\_VrefInt\_mVolt = \_\_LL\_ADC\_CALC\_DATA\_TO\_VOLTAGE(VDDA\_APPLI, aADCxConvertedData[1], LL\_ADC\_RESOLUTION\_12B);

hADCxConvertedData\_Temperature\_DegreeCelsius = \_\_LL\_ADC\_CALC\_TEMPERATURE(VDDA\_APPLI, aADCxConvertedData[2], LL\_ADC\_RESOLUTION\_12B);

/\* 可选地，为了这个例子的目的，从内部电压参考VrefInt的ADC转换计算模拟参考电压(Vref+)。\*/

/\* 这个电压应该对应于文字“VDDA\_APPLI”的值。 注:应用中电压Vref+值未知时可进行此计算。 \*/

uhADCxConvertedData\_VrefAnalog\_mVolt = \_\_LL\_ADC\_CALC\_VREFANALOG\_VOLTAGE(aADCxConvertedData[1], LL\_ADC\_RESOLUTION\_12B);

/\* 清除数组 \*/

for (tmp\_index\_adc\_converted\_data = 0; tmp\_index\_adc\_converted\_data < ADC\_CONVERTED\_DATA\_BUFFER\_SIZE; tmp\_index\_adc\_converted\_data++)

aADCxConvertedData[tmp\_index\_adc\_converted\_data] = 0x00;

ubDmaTransferStatus = 0; // 更新DMA传输状态变量

}

**⑤**编写外部13号中断处理程序

void EXTI4\_15\_IRQHandler(void)

{

if (EXTI\_GetITStatus(0x2000) != 0x00) // 0x2000==EXTI\_PIN\_13

{

EXTI\_ClearITPendingBit(0x2000); //Clear interrupt flag bit

/\* 设置变量向主程序报告按钮事件 \*/

ubUserButtonClickEvent = SET;

}

}

**⑥**编写ADC与DMA相关中断回调函数

void HAL\_ADC\_ConvCpltCallback(ADC\_HandleTypeDef \*hadc)

{

/\* 更新DMA传输状态变量 1完成\*/

ubDmaTransferStatus = 1;

/\* 点亮————DMA传输完成 \*/

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_5,SET);

}

void HAL\_ADC\_ConvHalfCpltCallback(ADC\_HandleTypeDef \*hadc)

{

/\* 不更新DMA传输状态变量未完成\*/

/\* 关闭————DMA传输未完成 \*/

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_5,RESET);

}

void HAL\_ADC\_ErrorCallback(ADC\_HandleTypeDef \*hadc)

{

//当ADC出错时，执行主要的错误处理函数提示用户

Error\_Handler();

}

**⑦**改写发生错误的函数——LED4闪烁用以提示用户

void Error\_Handler(void)

{

while(1)

{

/\* LED4闪烁 \*/

HAL\_GPIO\_TogglePin(GPIOA,GPIO\_PIN\_5);

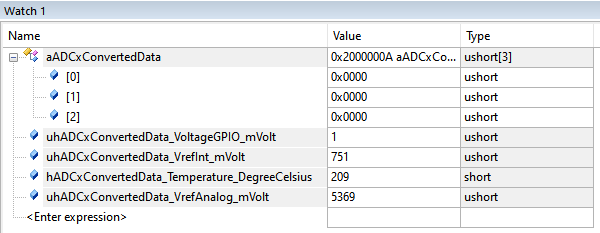
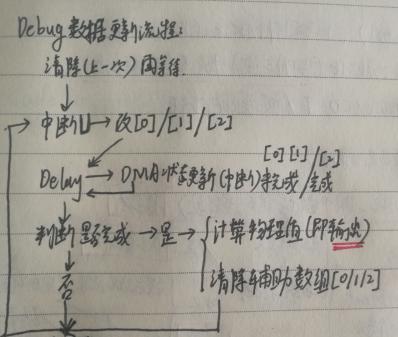
HAL\_Delay(50);

}

}

**第五步：**编译调试

打开debug，将需要观测的数据添加在可视化窗口watch1，例如：ADC转换数据数组，以及多个物理值还有状态标志参量。添加方法：选中变量，右键单击，选择“ADD……to watch1”。开始运行后，我们每按一次中断，ADC就会执行一次转换，其按下中断后数据更新流程图如下图（左）。当三个通道都转换完成之后，统一计算我们需要的物理值，赋值给相应变量，结果如下图（右）：



注1：在循环外开启DMA的时候就已经执行了一次ADC转换，故第一轮只需要按下两次中断按键就可以得到物理值。往后，则是以三次中断为一组，更新物理值变量。

注2：在此窗口下，鼠标右键单击变量，可以改变其显示格式（十进制和十六进制切换）。

**第六步：**改进与展望

可以将计算到的物理变量值，输出到LCM12864屏幕，方便观察。

方法：只需要在计算完成之后，将数据转换为字符串，并添加LCM输出即可。