串口&IIC&数模

IT & DMA

DMA （DIRECT MEMORY ACCESS）：是一种无须CPU的参与就可以让外设与系统内存之间进行双向数据传输的硬件机制，使用DMA可以使系统CPU从实际的I/O数据传输过程中摆脱出来，从而大大提高系统的吞吐率.

中断：是指CPU在执行程序的过程中，出现了某些突发事件时CPU必须暂停执行当前的程序，转去处理突发事件，处理完毕后CPU又返回源程序被中断的位置并继续执行。

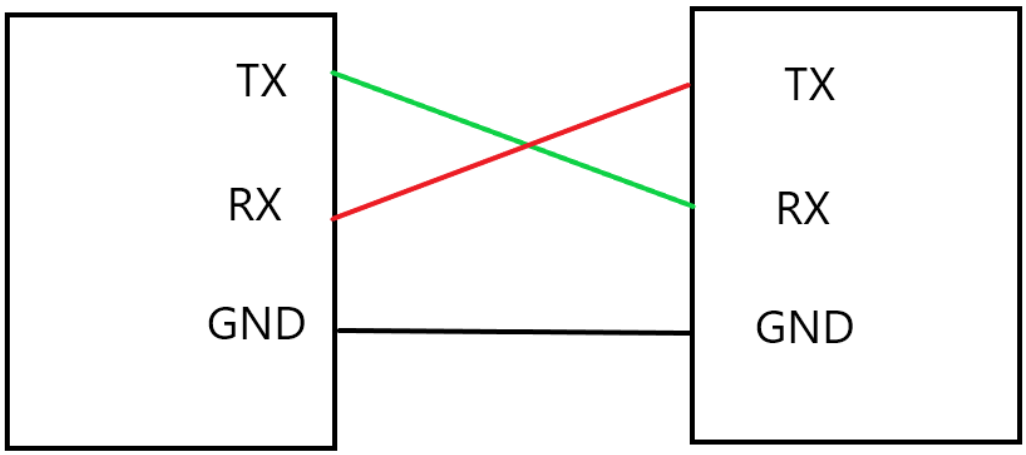
所以中断和*DMA*的区别就是*DMA*不需*CPU*参与而中断是需要*CPU*参与的。

◆中断方式是在数据缓冲寄存区满后，发中断请求，CPU进行中断处理
◆DMA方式则是以数据块为单位传输的,在所要求传送的数据块全部传送结束时要求CPU进行中断处理,大大减少了CPU进行中断处理的次数
◆中断方式的数据传送是由设备到CPU再到内存，或者相反。
◆DMA方式的数据传送则是将所传输的数据由设备直接送入内存，或是由内存直接送到设备。

串口

串行接口简称**串口**，也称串行通信接口或串行通讯接口（通常指**COM接口**），是采用串行通信方式的扩展接口。串行接口 （Serial Interface）是指数据一位一位地顺序传送。其特点是通信线路简单，只要一对传输线就可以实现双向通信（可以直接利用电话线作为传输线），从而大大降低了成本，特别适用于远距离通信，但传送速度较慢。

硬件连接



- TX：Transmit(传送) 发送端
- RX：Receive(接收) 接收端
- GND： 地， 两边需要共地

实例代码

1. 串口发送/接收函数

```
HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size,
uint32_t Timeout)//阻塞式发送
HAL_UART_Transmit_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)//
中断式发送
HAL_UART_Transmit_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t
Size)//DMA式发送
HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size,
uint32_t Timeout)//阻塞式接收
HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size)//中
断式接收
HAL_UART_Receive_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t
Size)//DMA式接收
```

2. 串口中断函数

可以通过重写回调函数来实现发送/接收后的操作

```
HAL_UART_IRQHandler(UART_HandleTypeDef *huart);  //串口中断处理函数
HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart);  //串口发送中断回调函数
HAL_UART_TxHalfCpltCallback(UART_HandleTypeDef *huart);  //串口发送一半中断回调函数
（用的较少）
HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart);  //串口接收中断回调函数
HAL_UART_RxHalfCpltCallback(UART_HandleTypeDef *huart);//串口接收一半回调函数（用的较
少）
HAL_UART_ErrorCallback();//串口接收错误函数
```

3. 串口状态

`HAL_UART_GetState(UART_HandleTypeDef *huart)` 这一函数可以获取当前串口状态

比如:

```
while(HAL_UART_GetState(&huart1) == HAL_UART_STATE_BUSY_TX);    //检测UART发送结束
```

HAL库预定义了这些状态:

```
HAL_UART_STATE_RESET             //外围设备未初始化
HAL_UART_STATE_READY             //外围设备已经初始化并可以使用
HAL_UART_STATE_BUSY              //内部进程正在进行
HAL_UART_STATE_BUSY_TX           //数据传输正在进行
HAL_UART_STATE_BUSY_RX           //数据接收正在进行
HAL_UART_STATE_BUSY_TX_RX        //数据传输接收正在进行
HAL_UART_STATE_TIMEOUT           //超时
HAL_UART_STATE_ERROR             //错误
```

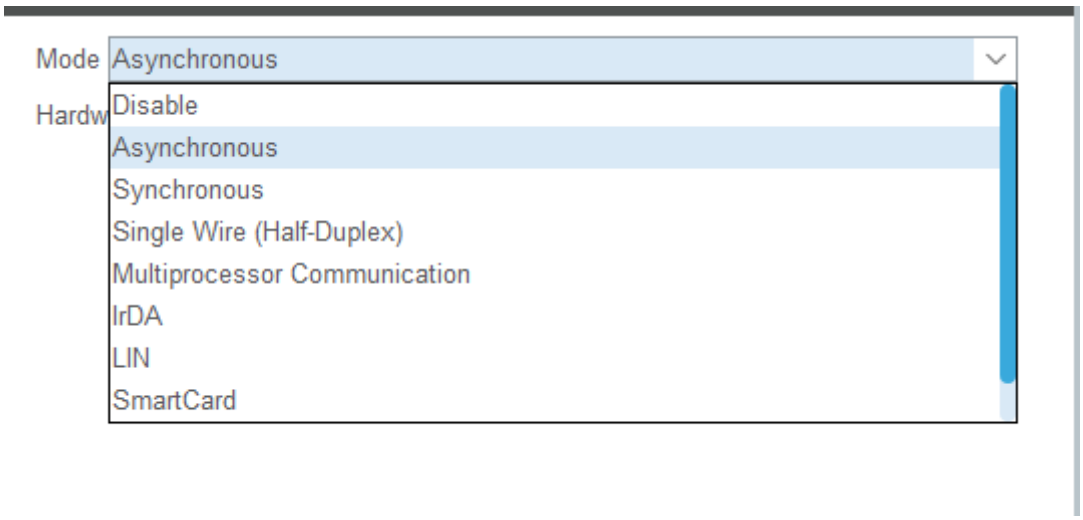
黑科技（但实际不怎么用）：

重定向printf输出到串口, 重写fputc:

```c
int fputc(int ch, FILE *f){
    uint8_t temp[1] = {ch};
    HAL_UART_Transmit(&huart1, temp, 1, 2); //huart1根据实际情况修改
    return ch;
}
```

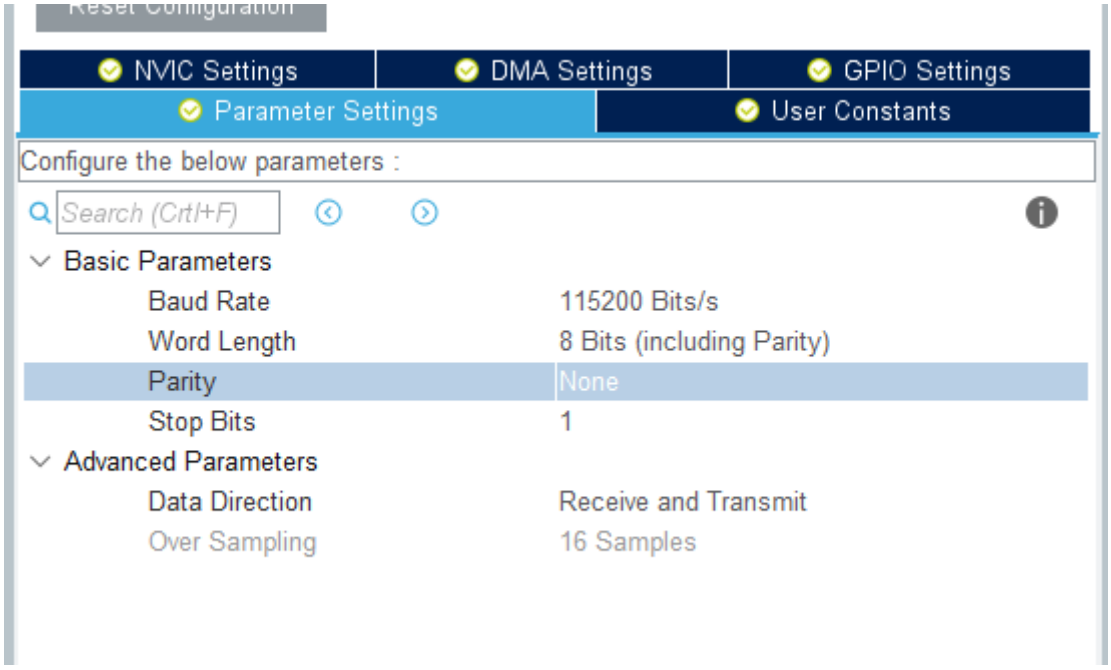
Cube配置

1. 串口模式选择异步

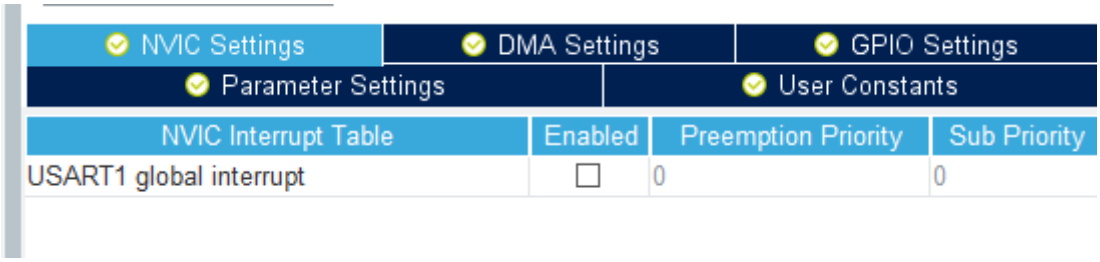


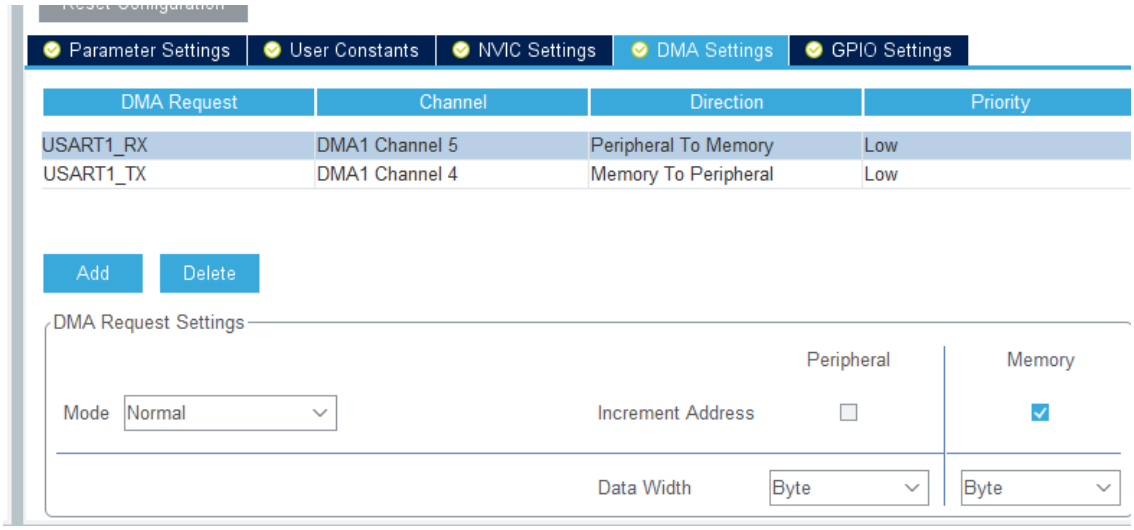
2. 配置波特率，数据位，奇偶校验，停止位

   如果需要8位数据，无奇偶校验，则WordLength=8
   如果需要8位数据，有奇偶校验，则WordLength=9



3. 配置中断 or DMA





完成

IIC

原理：https://blog.csdn.net/shaguahaha/article/details/70766665

Cube&实操 https://www.cnblogs.com/xingboy/p/9647326.html

主从模式参考代码 https://blog.csdn.net/u011456016/article/details/70233599

弄明白： 硬件连接 & Cube配置 & 函数功能

DAC & ADC

简单DAC https://www.cnblogs.com/zjx123/p/12038164.html

简单ADC https://www.cnblogs.com/xingboy/p/10018749.html

ADC详解 https://www.likecs.com/default/index/show?id=128689

ADC

Analog-to-Digital Converter的缩写。指模/数转换器或者模拟/数字转换器。是指将**连续变量的模拟信号转换为离散的**
数字信号的器件。
STM32的ADC， 是 **12 位逐次逼近型**的模数转换器。ADC 的结果可以左对齐或右对齐方式存储在 16 位数据寄存器
中。
STM32将 ADC 的转换分为 2 个通道组：**规则通道组和注入通道组。**规则通道相当于 你正常运行的程序，而注入通道
呢，就相当于中断。
STM32的ADC的各通道可以组成规则通道组或注入通道组，但是在转换方式还可以有**单次转换、连续转换、扫描转换**
模式。至于他们的不同，感兴趣的话上网搜。我们一般都是采用连续转换模式。

有关ADC主要的hal库函数：

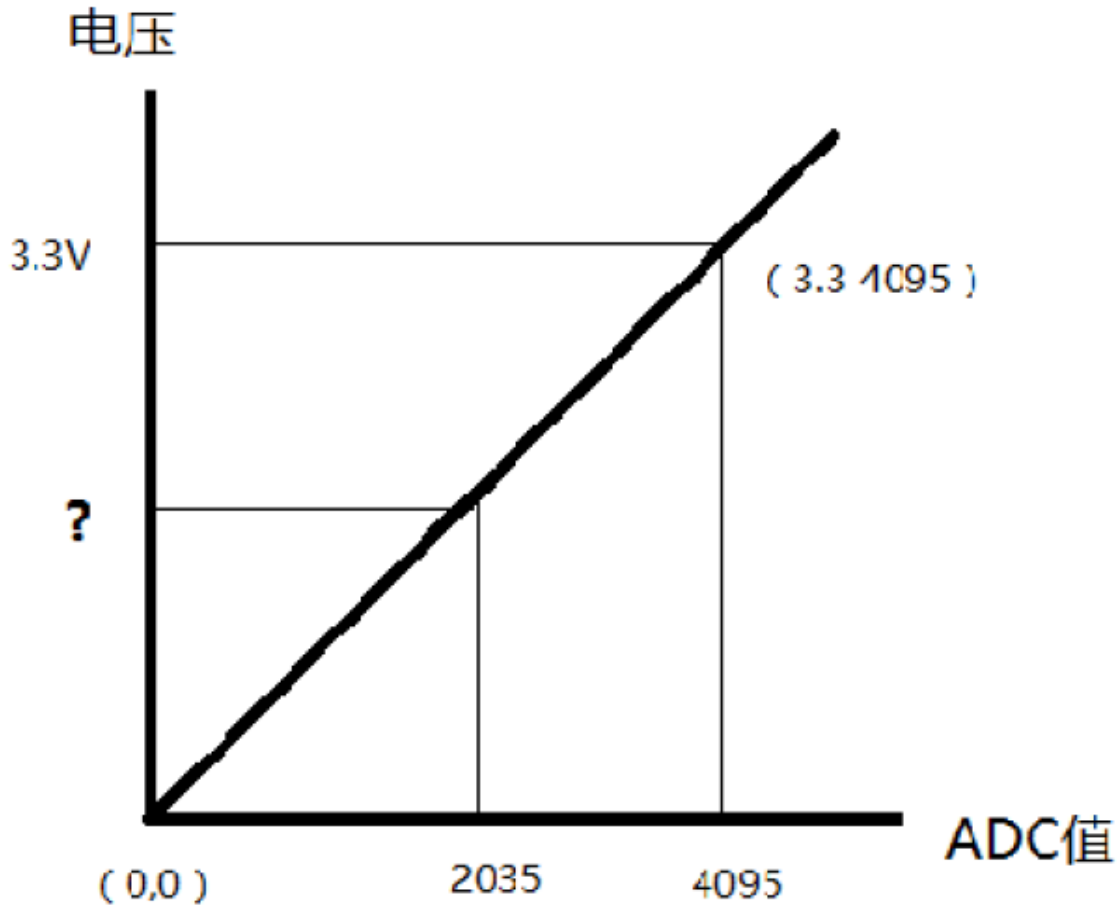
```
HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef* hadc);//使能ADC
HAL_StatusTypeDef HAL_ADC_PollForConversion(ADC_HandleTypeDef* hadc, uint32_t
Timeout); //等待ADC转换完成
uint32_t HAL_ADC_GetValue(ADC_HandleTypeDef* hadc); //获取ADC转换的结果
```

ADC采样会得到什么值

由于STM32的ADC是12位逐次逼近型的模拟数字转换器，也就是说ADC模块读到的数据是12位的数据。因此：**STM32读到的ADC值，是从0到4095（111111111111）。当把ADC引脚接了GND，读到的就是0；当把ADC引**
脚接了VDD，读到的就是4095。

读到的值怎么换算成实际的电压值

前面提到了，我们输入GND，读到的值是0，输入VDD，得到的值是4095，那么，当读到2035的时候，怎么求输入电
压多少V吗？这个问题，归根接地，就到了数学XY坐标，已知两点坐标值（0,0）（3.3,4095），给出任意X坐标值，
求Y值的问题了吧？简单不简单？



参考电压

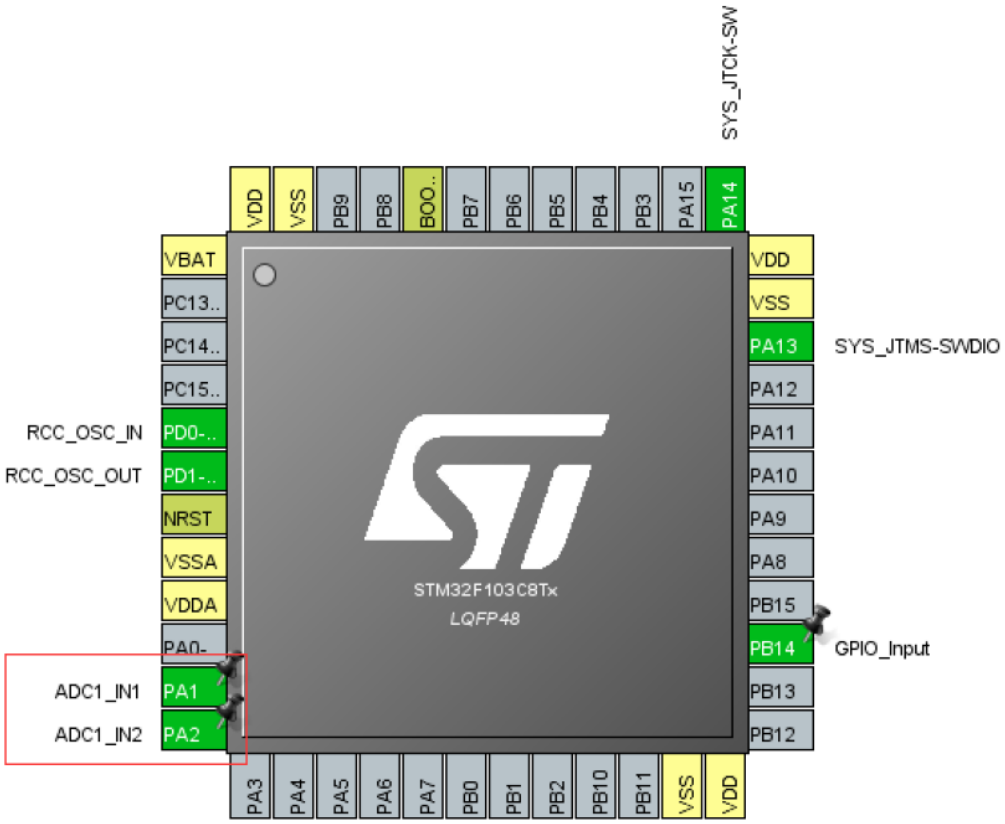
我们板子默认参考电压是3.3V。

ADC引脚的输入电压范围是多大

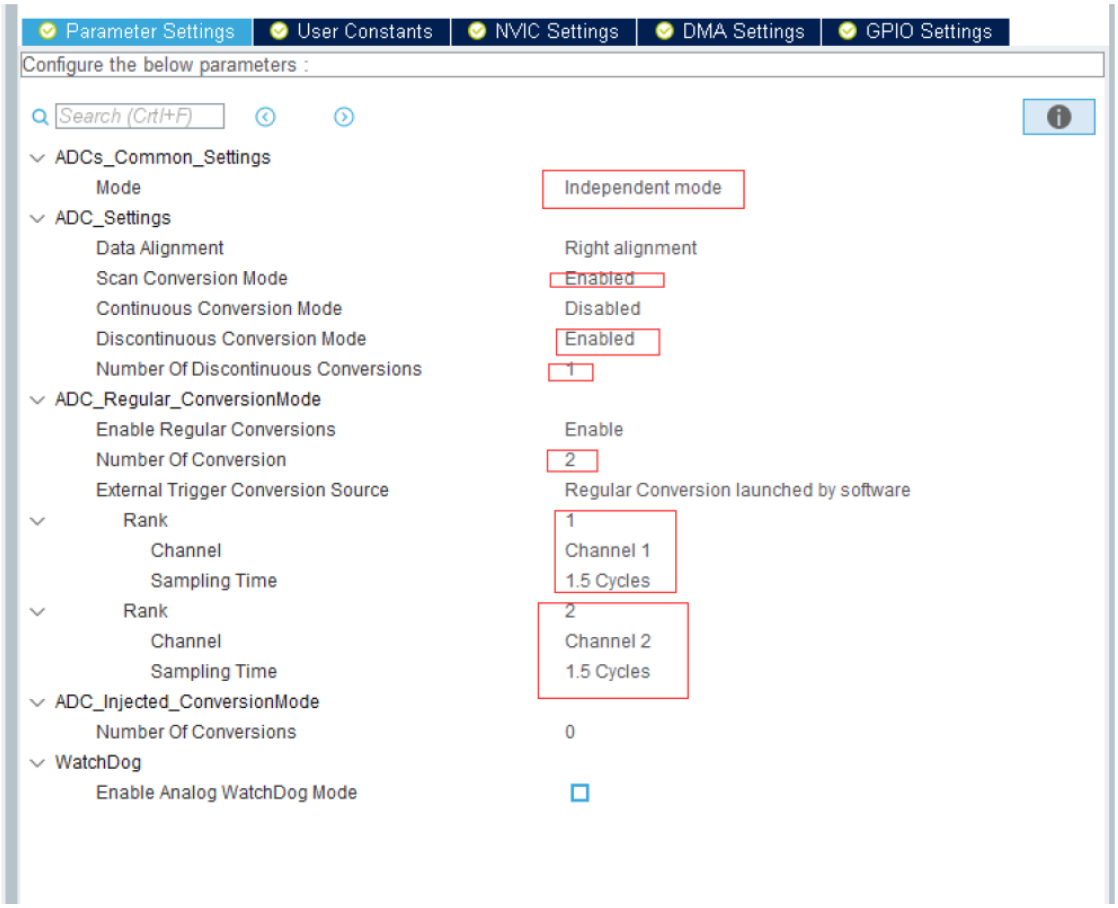
0~Vref。
如果待检测电压大于ADC的最大输入电压，可以通过电阻分压减小输入电压。

ADC实验

1. 开启ADC通道



2. 配置如图



多路ADC采集时，必须要选用扫描模式和间断模式。

连续模式适用于单路采集。

可以设置优先级Rank，即采集顺序。

设置每次采集一个，分两次采集。

3. 定义两个全局变量

```
uint32_t value_x,value_y;
/* USER CODE END Includes */
```

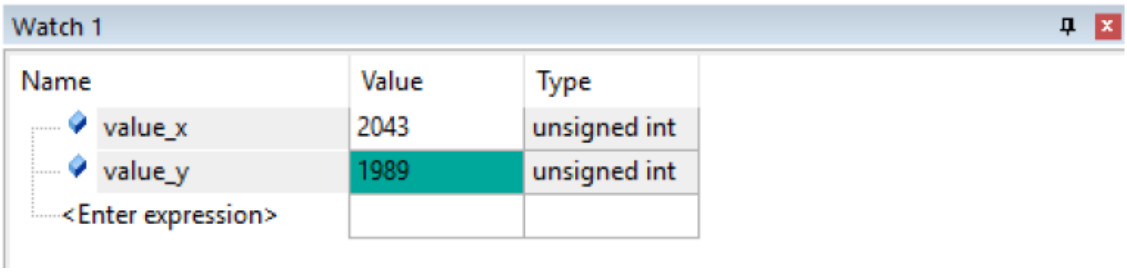
4. 在主函数while循环中加入如下代码，有注释自己看。

```
/* USER CODE BEGIN WHILE */
while (1)
{
  HAL_ADC_Start(&hadc1);//只能转换一次
  HAL_ADC_PollForConversion(&hadc1,10);//等待ad转换完成
  value_x=HAL_ADC_GetValue(&hadc1);//获取in1ad转换结果
  HAL_ADC_Start(&hadc1);//继续开启转换
  HAL_ADC_PollForConversion(&hadc1,10);//等待ad转换完成
  value_y=HAL_ADC_GetValue(&hadc1);//获取in2ad转换结果
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

每一个通道adc采集需要三行语句。

然后在调试器里面便可以看到value_x和value_y的结果。



DAC

Digital-to-Analog Converter的缩写。指数/模转换器。是指将离散的数字信号转换为连续变量的模拟信号的器件。
STM32的DAC模块是**12位数字输入**，电压输出型的DAC。
STM32F4xx有2个DAC，每个DAC对应一个输出通道。

表29　ADC/DAC

ADC/DAC引脚	GPIO配置
ADC/DAC	模拟输入

为什么要使用模拟输入模式呢？因为一旦使能DACx通道后，相应的GPIO引脚会自动与DAC的模拟输出相连，设置为
输入，是为了防止额外的干扰。
DAC数据格式，一般都采用12位数据右对齐，这样便于数据处理。

DAC输出电压

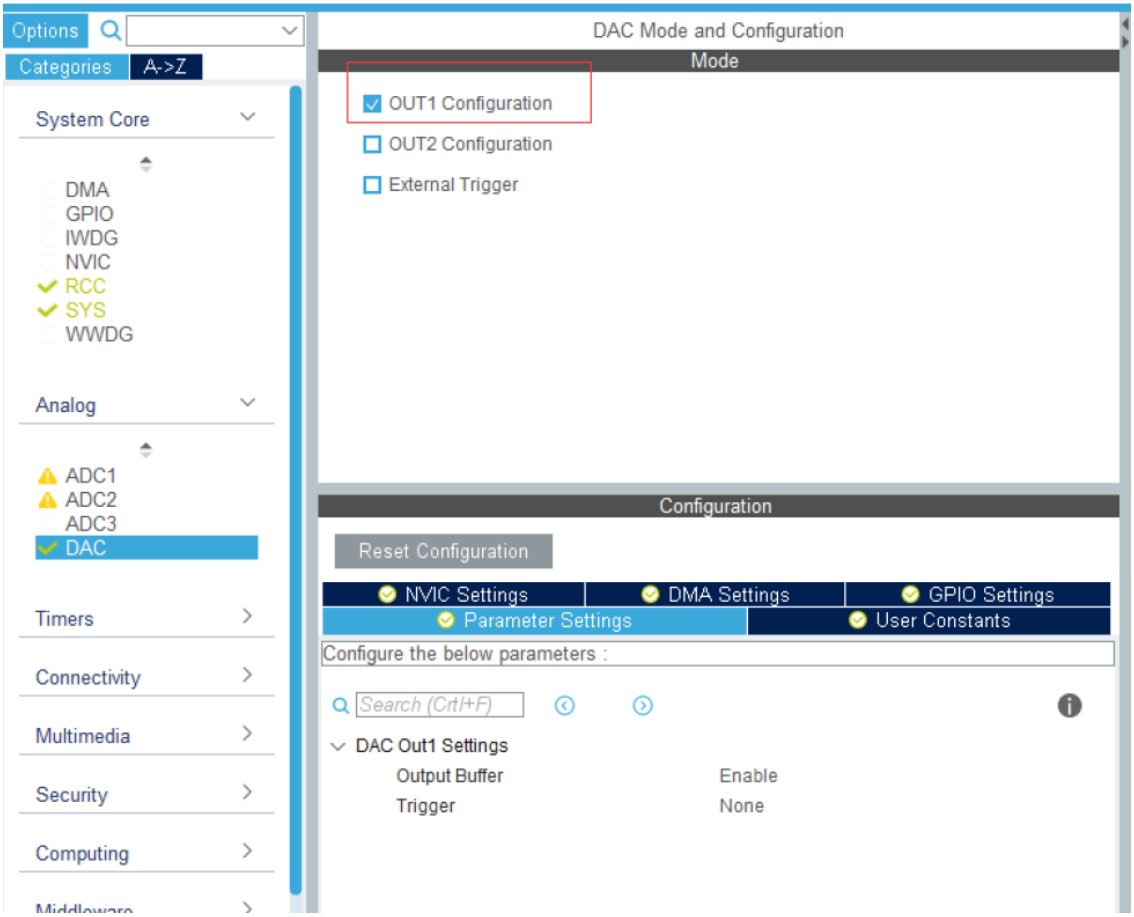
当DAC的参考电压位VREF+的时候，数字输入经过DAC被线性地转换为模拟电压输出，其范围为0到VREF+。
DAC输出 = VREF x (DOR / 4095)。
注意：数据格式应该选择12位数据右对齐。

主要用到的函数

```
HAL_StatusTypeDef HAL_DAC_SetValue(DAC_HandleTypeDef* hdac, uint32_t Channel,
uint32_t Alignment, uint32_t
Data);
```

DAC实验

1. 开启DAC，F4系列只有一个DAC，开启DAC的CH1，配置默认。



同时，会发现Cube为我们开好了一个PA4作为该DAC输出引脚。

2. 进入keil。定义一个变量，作为输入数字量。通过修改数字量的大小，可以得到不同的输出电压。

```
/* USER CODE BEGIN 1 */
  uint32_t value=0;
/* USER CODE END 1 */
```

3. 开启DAC通道

```
  /* USER CODE BEGIN 2 */
  HAL_DAC_Start(&hdac,DAC_CHANNEL_1);//开启dac的1通道
  /* USER CODE END 2 */
```

4. 开启转换

```
/* USER CODE BEGIN WHILE */
while (1)
{
  if(value>4025)//防止超过2^12-1
  {
    value=0;
  }
  HAL_DAC_SetValue(&hdac,DAC_CHANNEL_1,DAC_ALIGN_12B_R,value);//DA转换函数
  value+=100;
  HAL_GPIO_TogglePin(GPIOF,GPIO_PIN_14);
  HAL_Delay(1000);
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
```

5. 至此，代码编写完毕。用万用表测量PA4的引脚可以看到输出电压在0~3.3V之间线性变化，符合预期效果。

DAC的两个通道的赋值是互不干扰的。

Debug - 在线修改参数

watch窗口中的变量值可以手动修改

Homework

- 搞明白串口设置的所有参数
- 了解IIC几种模式的不同
- 搞明白ADC多通道采集方法和对应函数