第7讲

ADC

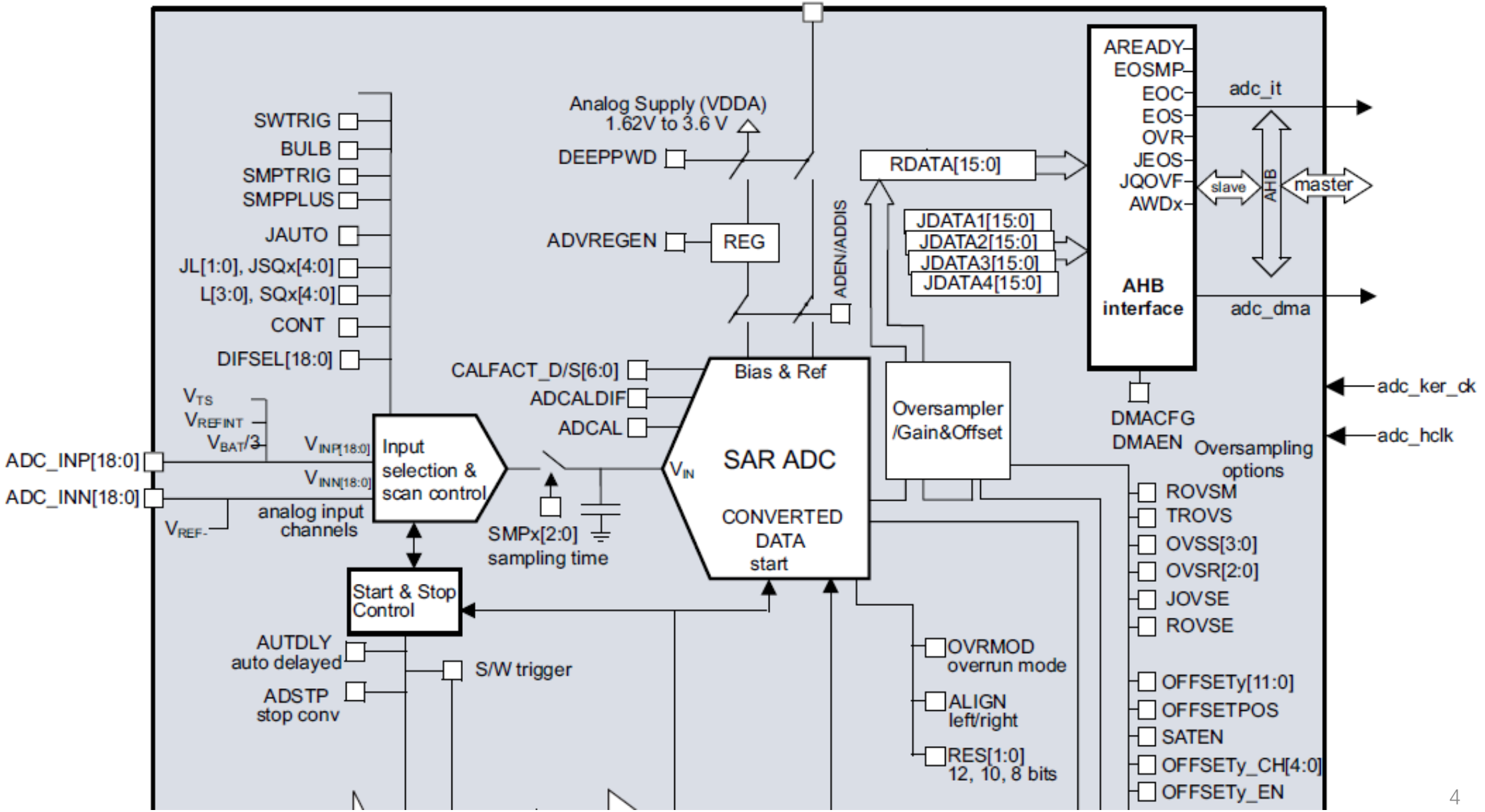
主要内容

- STM32G4的ADC
- ADC的单次采样
- ADC的连续采样
- 用定时器控制ADC采样
- 动手练习7

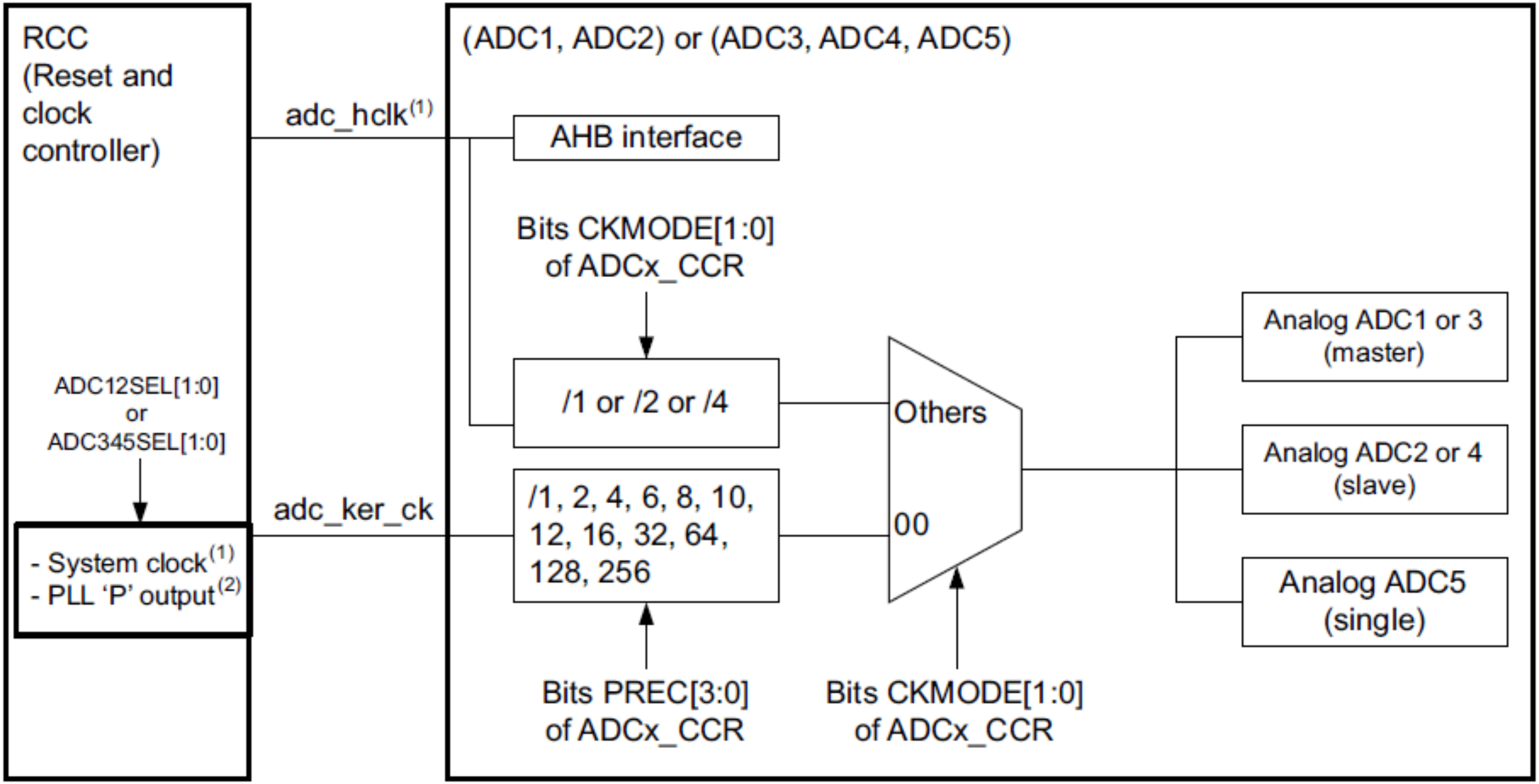
STM32G4的ADC

- 不同的型号所含ADC模块数量不同，最多有5个ADC（ADC1~5）；但也并非完全独立，其中ADC1和ADC2是一对，ADC3和ADC4是一对，ADC5可独立控制。
- 每个ADC都包含一个12位逐次比较型模拟数字转换器。
- 每个ADC有最多至19个通道，不同的通道具有单次、连续和扫描或断续等采样模式。

STM32G4的ADC



ADC clock

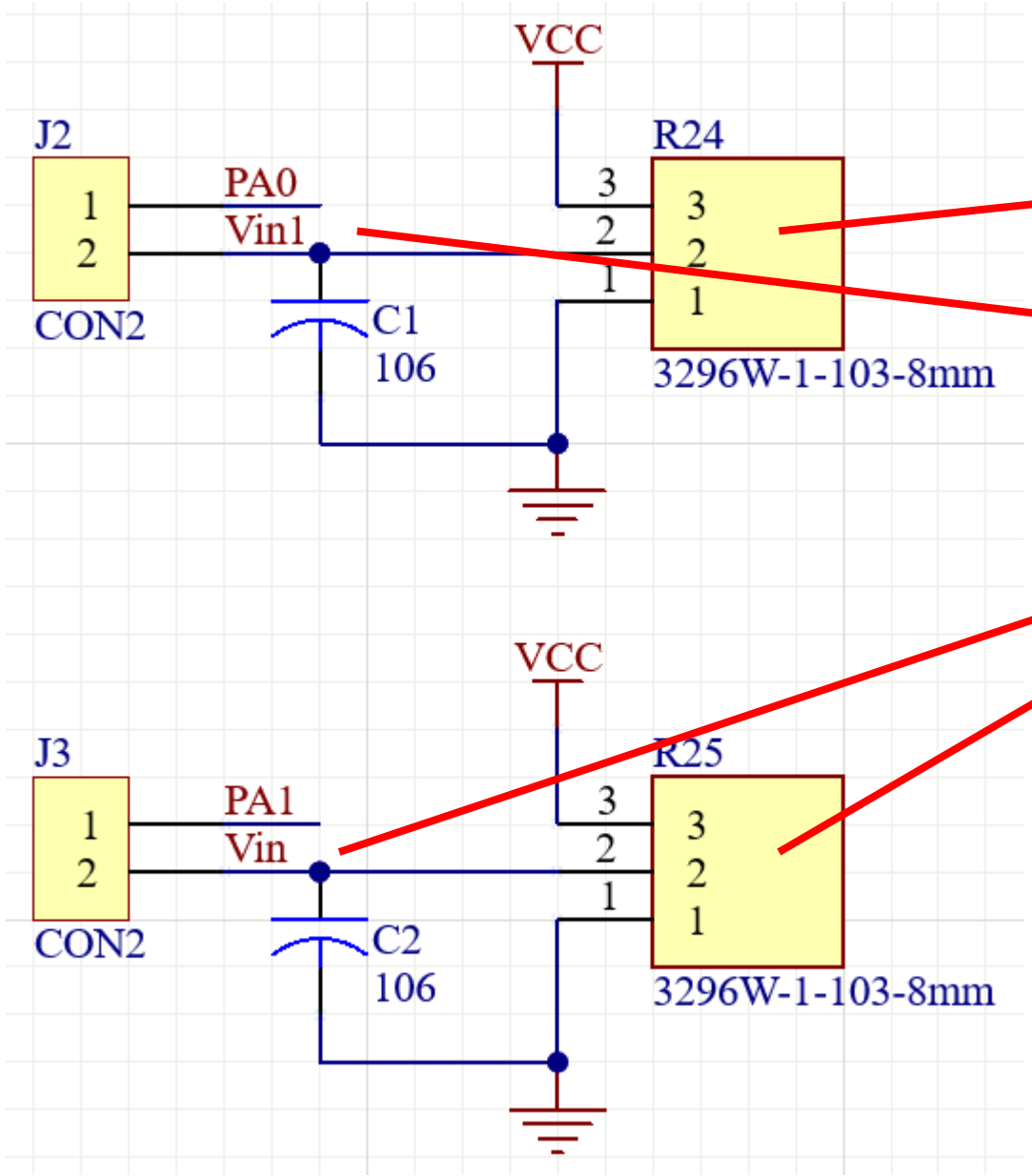

RCC
(Reset and
clock
controller)
(ADC1, ADC2) or (ADC3, ADC4, ADC5)
adc_hclk(1)
AHB interface
Bits CKMODE[1:0]
of ADCx_CCR
ADC12SEL[1:0]
or
ADC345SEL[1:0]
/1 or /2 or /4
Others
Analog ADC1 or 3
(master)
Analog ADC2 or 4
(slave)
adc_ker_ck
- System clock(1)
- PLL 'P' output(2)
/1, 2, 4, 6, 8, 10,
12, 16, 32, 64,
128, 256
00
Analog ADC5
(single)
Bits PREC[3:0]
of ADCx_CCR
Bits CKMODE[1:0]
of ADCx_CCR

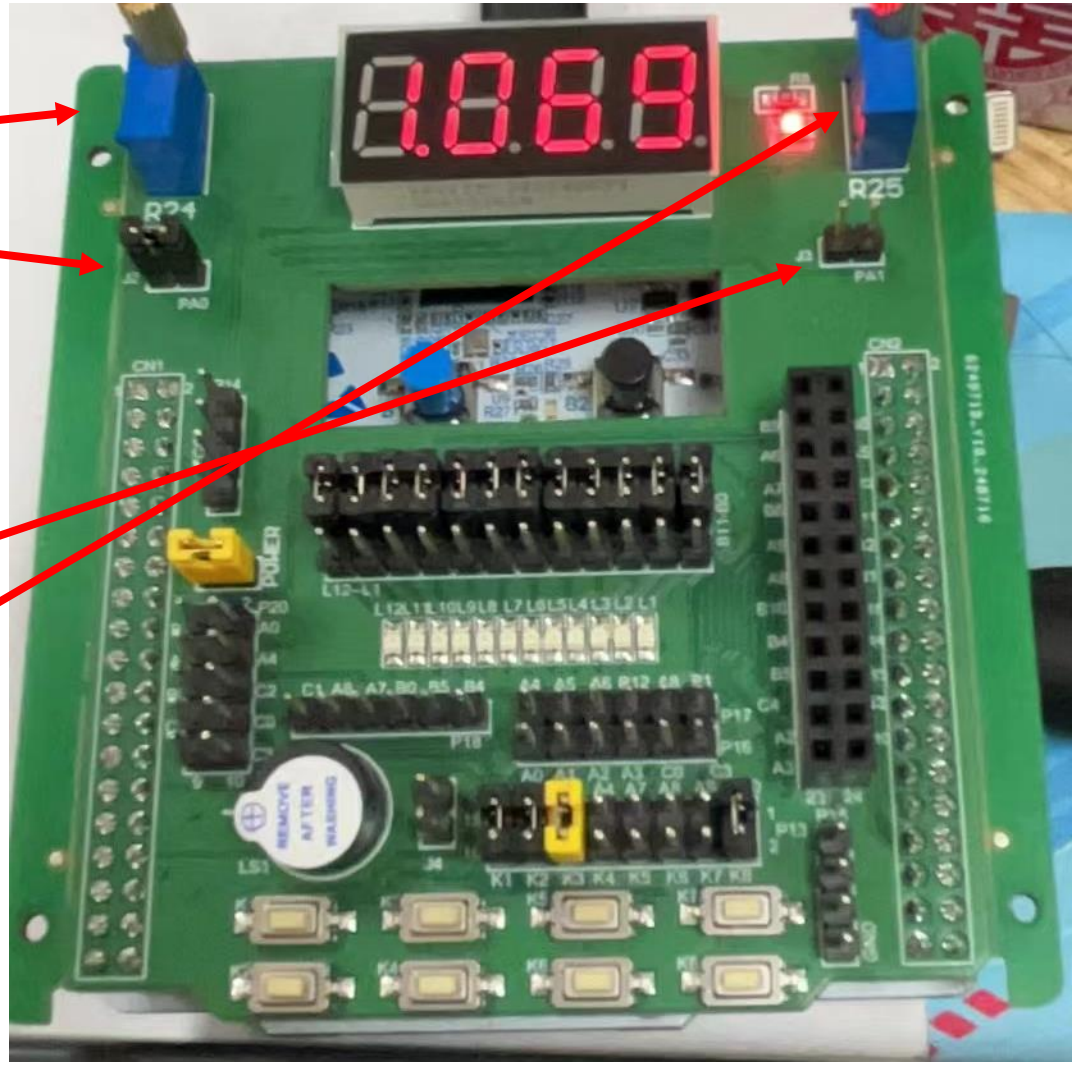
练习7：ADC

- 使用ADC1的一个通道，以**单次采样**的模式，采集外部输入的直流电压信号。
- 用NUCLEO-G431RB板上的按键B1来启动ADC采样
  - ✓ 每按下一次B1，进行一次ADC转换。
  - ✓ 通过查询方式判断是否转换完成；
  - ✓ 一旦转换完成，主程序会从ADC数据寄存器中读取转换结果，并将结果通过串口送出；
  - ✓ 当输入信号的幅值大于一定值时，点亮板上发光二极管LD2。
- 用到ADC、串口、输入输出等多个模块。AD转换虽用查询模式，但对按键状态的识别，将采用中断的方式。

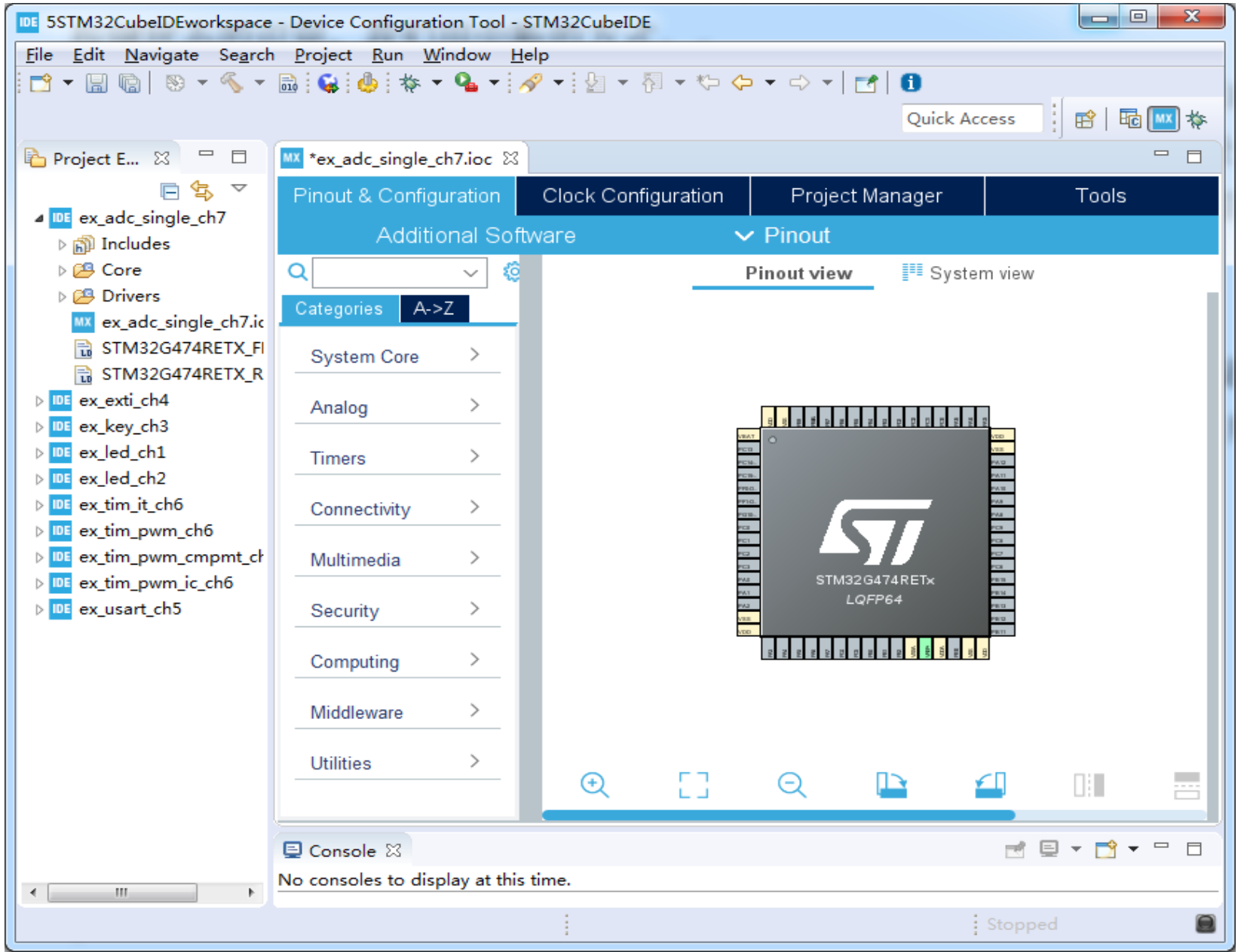
ADC电压范围及对应引脚

- 由于ADC的输入电压范围为0~3.3V，所以要保证外部施加的信号不能超过此电压范围内，否则可能会导致硬件损坏。
- 采用ADC1的第一个通道，对应STM32G431RB的引脚为PA0，在NUCLEO-G431RB板上通过CN7端子的第28引脚引出。
- 按键B1连接的引脚为PC13，LD2的控制引脚为PA5。


VCC
J2
1
2
CON2
PA0
Vin1
C1
106
R24
3
2
1
3296W-1-103-8mm
VCC
J3
1
2
CON2
PA1
Vin
C2
106
R25
3
2
1
3296W-1-103-8mm


R24
R25

建立新工程


5STM32CubeIDEworkspace - Device Configuration Tool - STM32CubeIDE
File Edit Navigate Search Project Run Window Help
Quick Access
Project E...
ex_adc_single_ch7
Includes
Core
Drivers
ex_adc_single_ch7.ic
STM32G474RETX_FI
STM32G474RETX_R
ex_exti_ch4
ex_key_ch3
ex_led_ch1
ex_led_ch2
ex_tim_it_ch6
ex_tim_pwm_ch6
ex_tim_pwm_cmpmt_ch
ex_tim_pwm_ic_ch6
ex_usart_ch5
*ex_adc_single_ch7.ioc
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Additional Software
Pinout
Categories
A->Z
System Core
Analog
Timers
Connectivity
Multimedia
Security
Computing
Middleware
Utilities
Pinout view
System view
STM32G474RETx
LQFP64
Console
No consoles to display at this time.
Stopped

时钟源和Debug模式配置

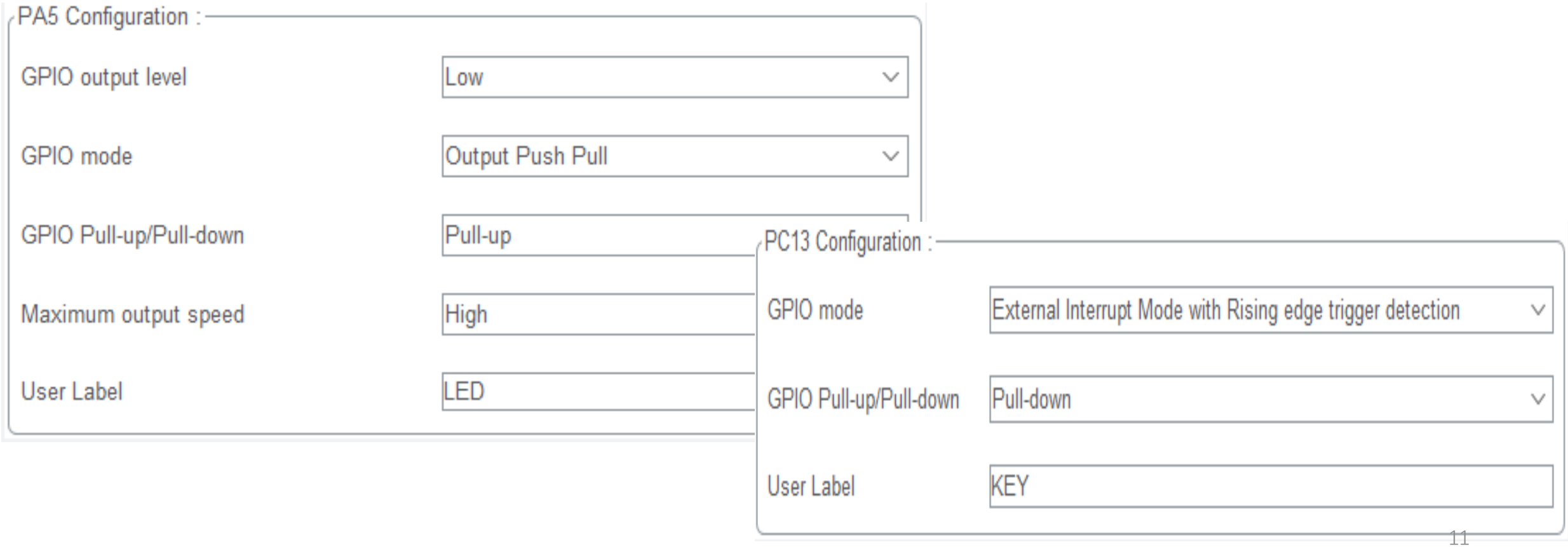
■选择时钟源和Debug模式

✓ System Core->RCC->将高速时钟（HSE）选择为Crystal/Ceramic Resonator

✓ SYS->Debug选择为Serial Wire

配置GPIO

■ 配置PA5为输出（GPIO_Ooutput）,配置PC13为中断模式（GPIO_EXTI13）



配置中断

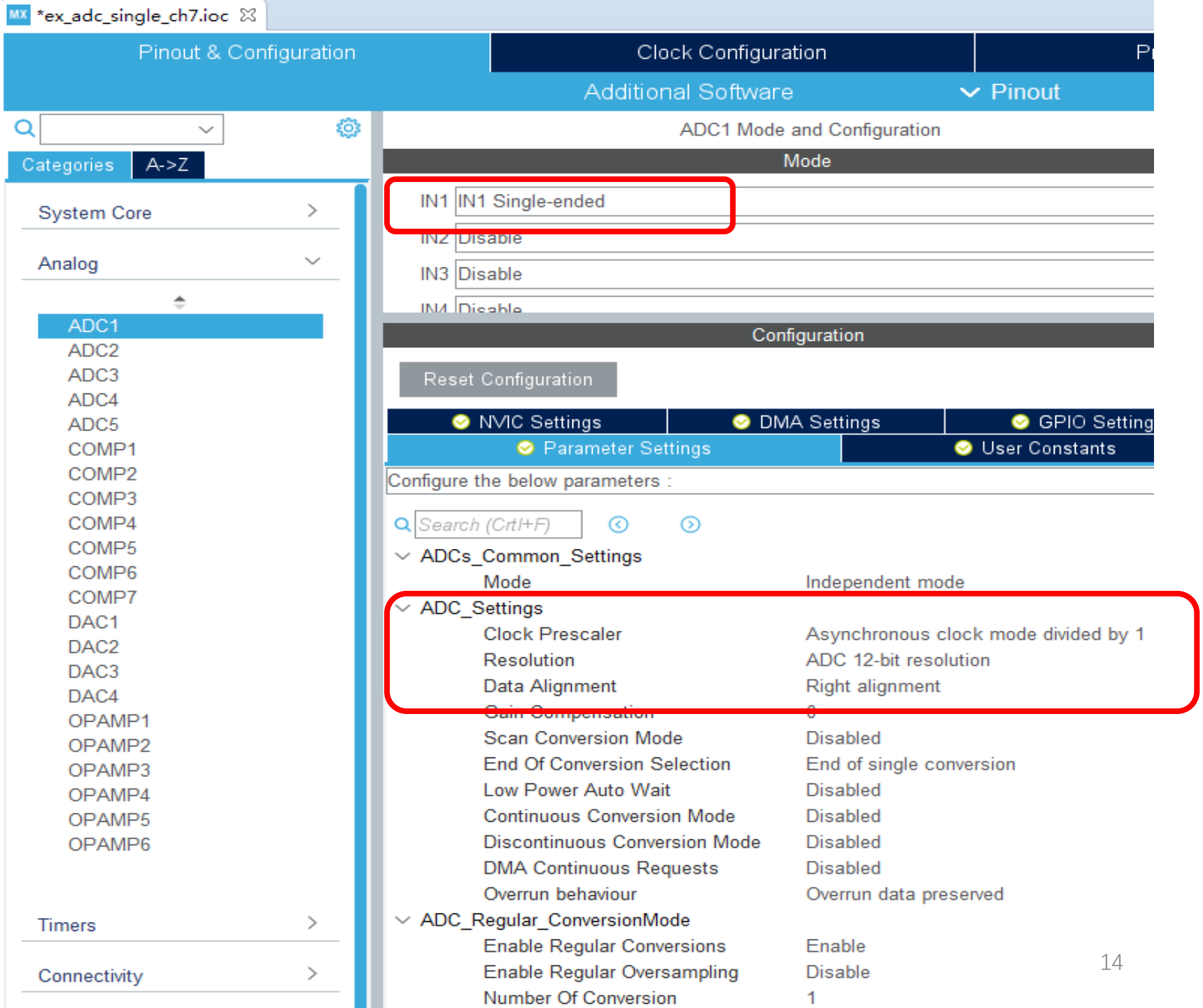
- System Core下的NVIC，在NVIC中断表中，将EXTI line[15:10] interrupts使能，并将其抢占式优先级设置为2（由于仅用到的一个中断，级数选择可任意）

配置串口

- 选择“Connectivity”中的USART2，在其模式（Mode）栏选“异步”（Asynchronous），其他参数设置均保持默认（波特率115200），不开启中断。
- 将USART2的两个引脚PA2和PA3均设置为上拉（Pull-up）。

配置ADC

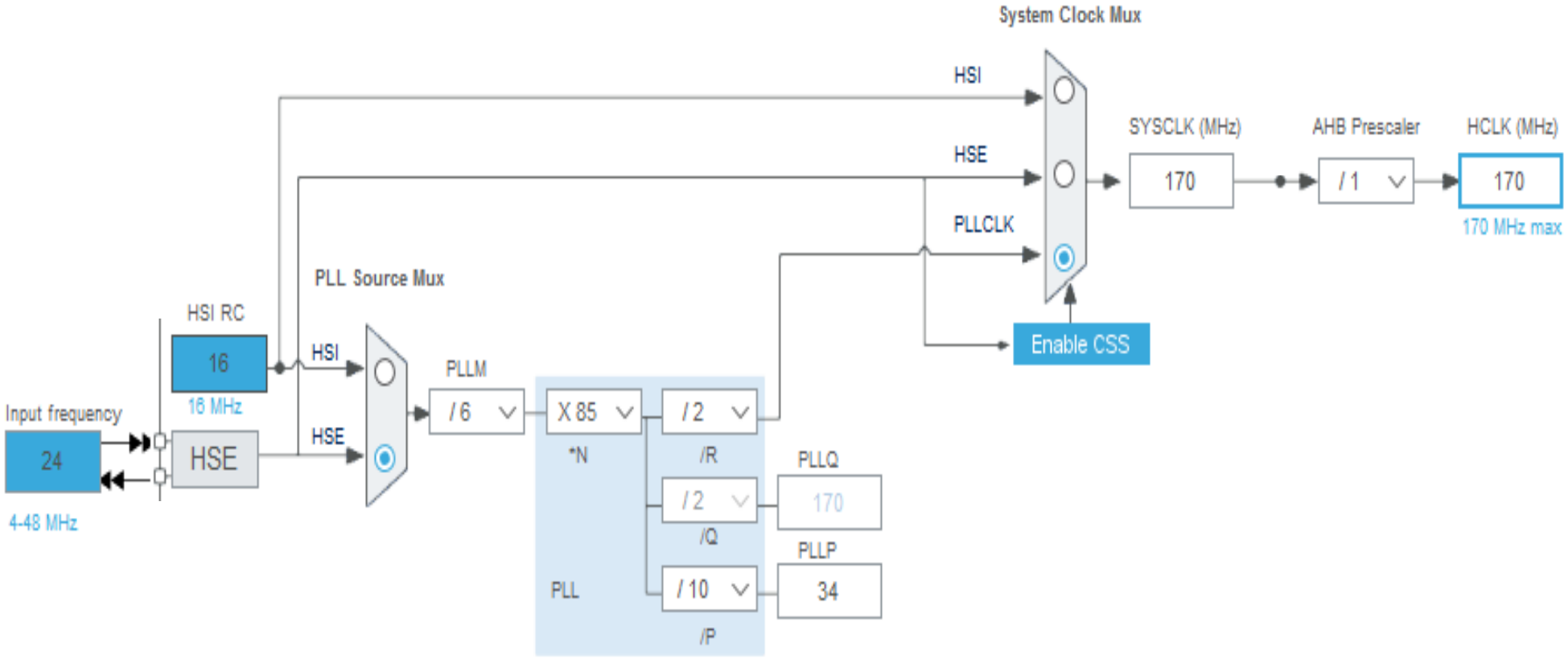
- 选择Analog中的ADC1。在ADC1的Mode栏，选择IN1为IN1 Single-ended。在下面的Configuration栏，参数设置~~***可暂时均保持默认值***~~



时钟配置

■配置系统时钟

✓ 在“Clock Configuration”中，将系统时钟（SYSCLK）配置为170Mhz



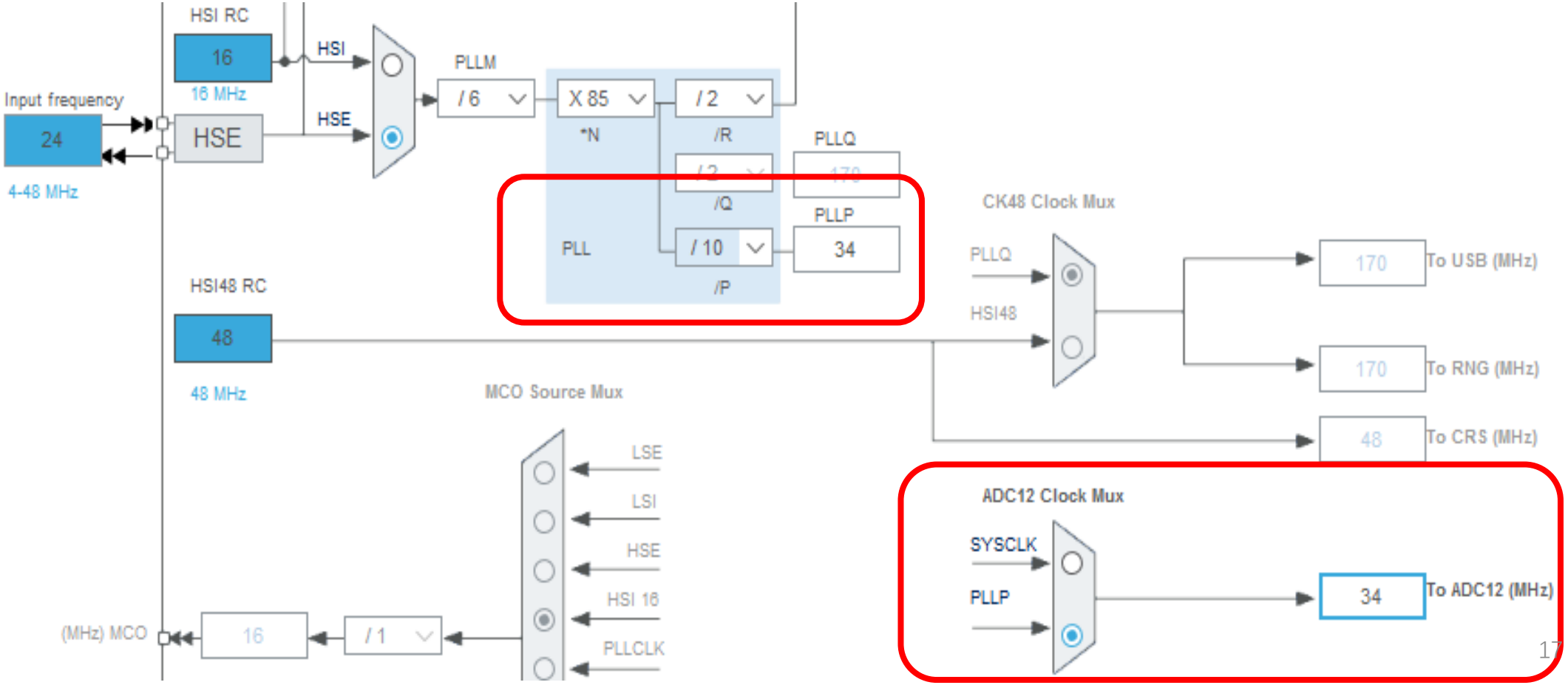
ADC时钟配置

符号	参数	条件	最小值	最大值	单位
f_{ADC}	ADC的时钟频率	Range1，单路ADC操作	0.14	60	MHz
		Range2	-	26	
		Range1，所有ADCs操作，单端模式 V_{DDA}≥2.7V	0.14	52	
		Range1，所有ADCs操作，单端模式 V_{DDA}≥1.62V	0.14	42	
		Range1，单路ADC操作，差分模式 V_{DDA}≥1.62V	0.14	56	

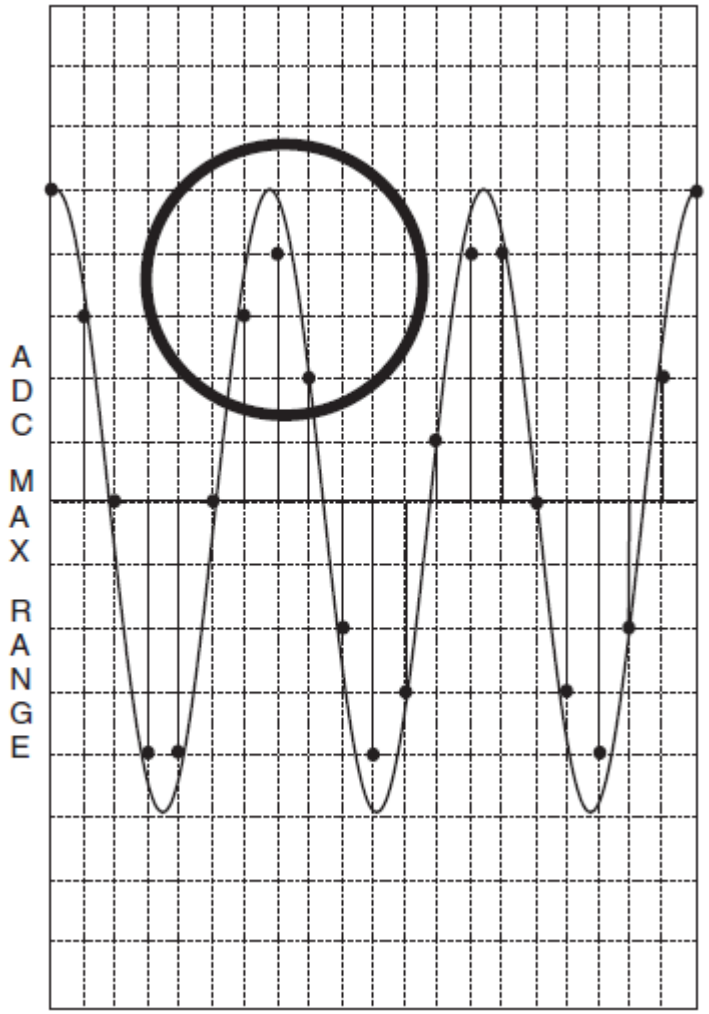
Range 1和Range 2是STM32G431的两种电压模式。Range 2主要用于低功耗模式

ADC时钟范围

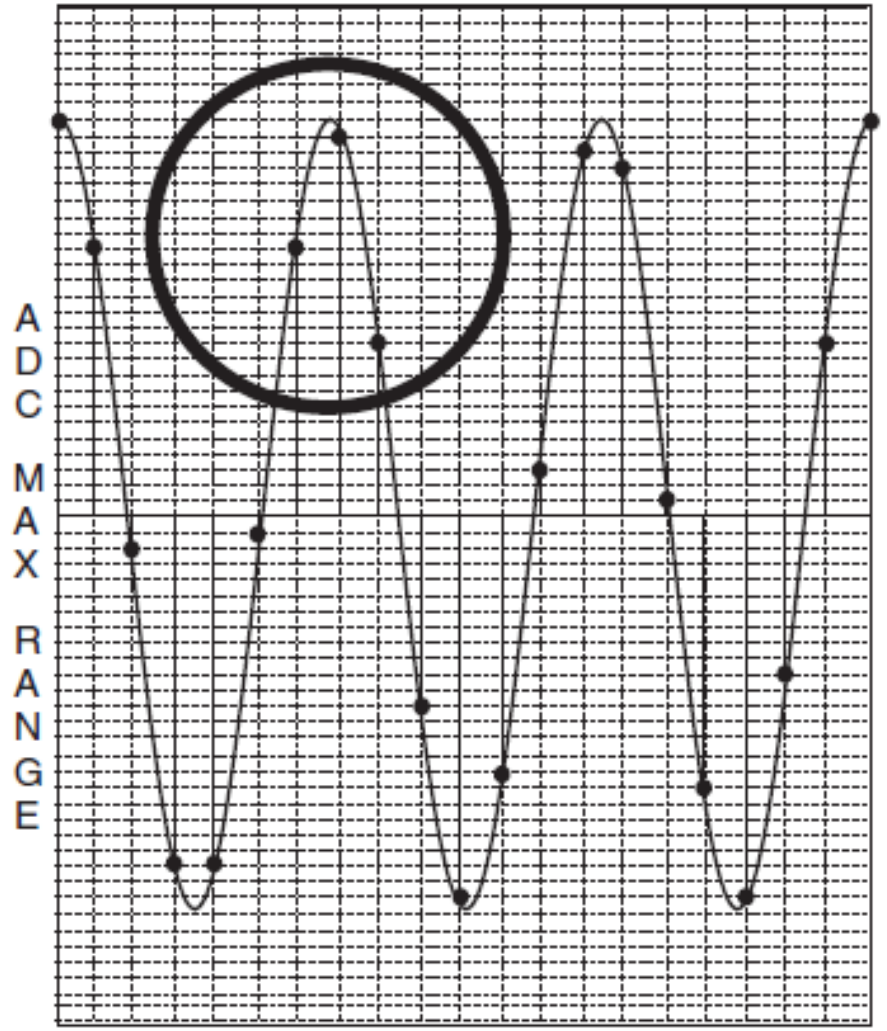
■ 在本例中，没有使用低功耗模式，并且是让ADC进行单次采样，所以最高时钟可以到60MHz。为了可靠起见，配置ADC的时钟为34MHz



ADC的位数

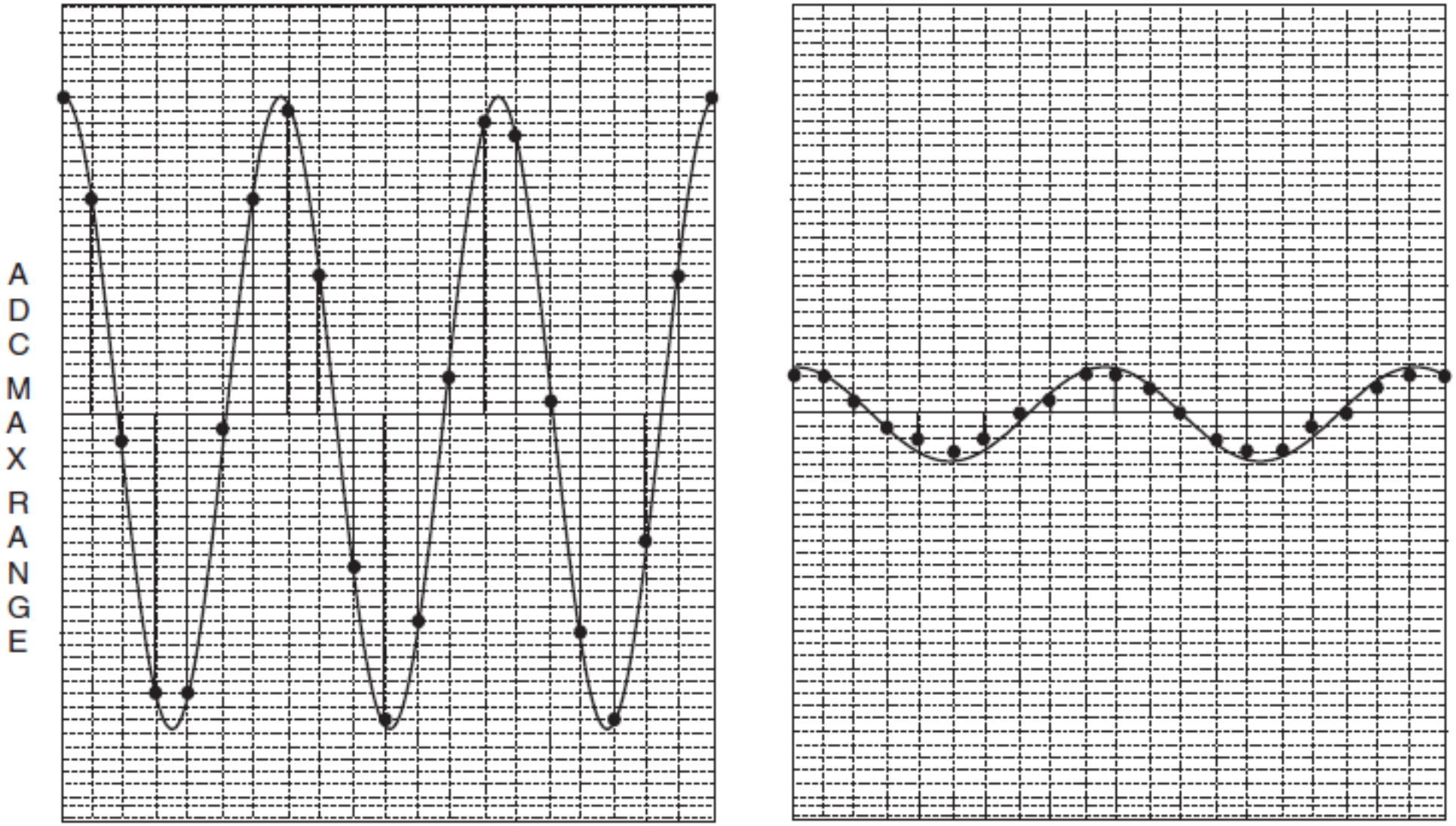


4 bit

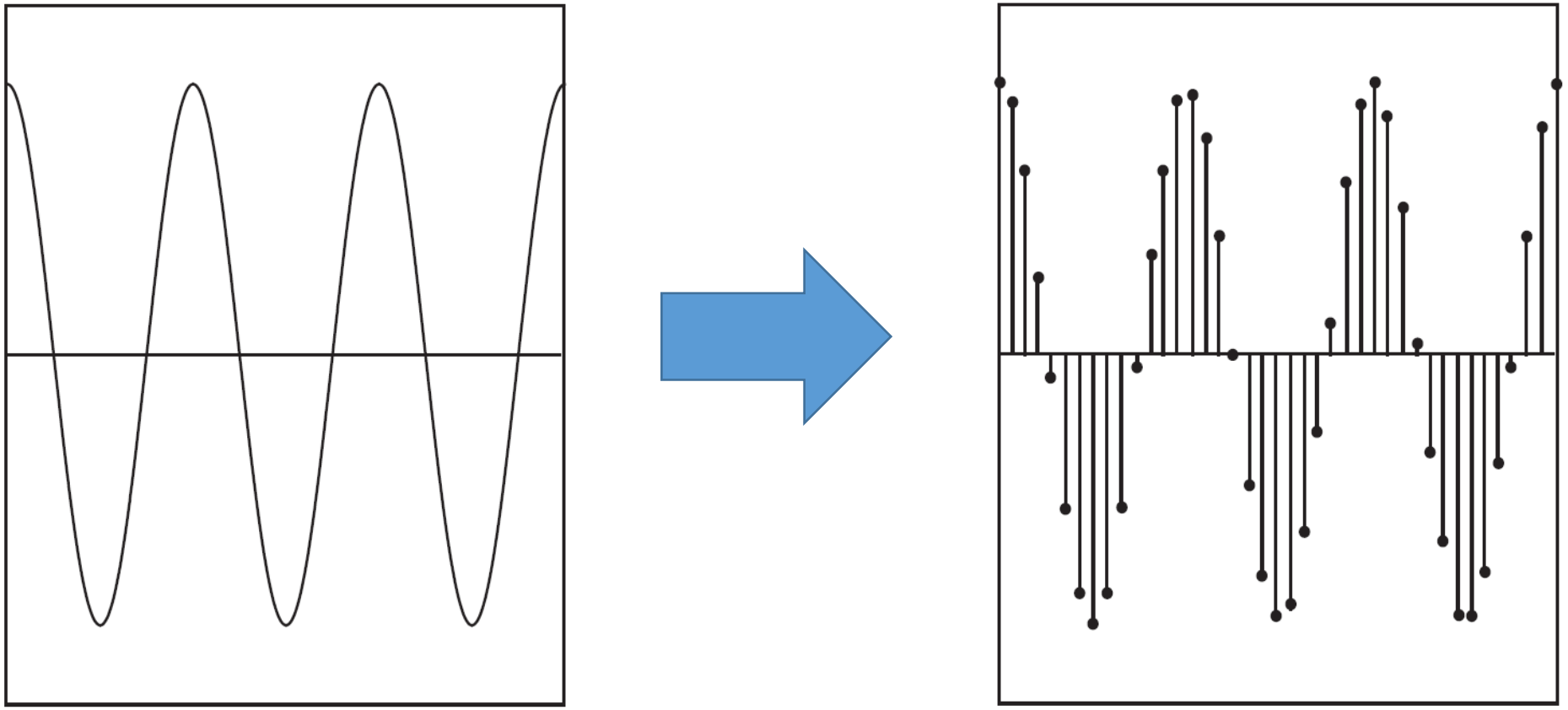


6 bit

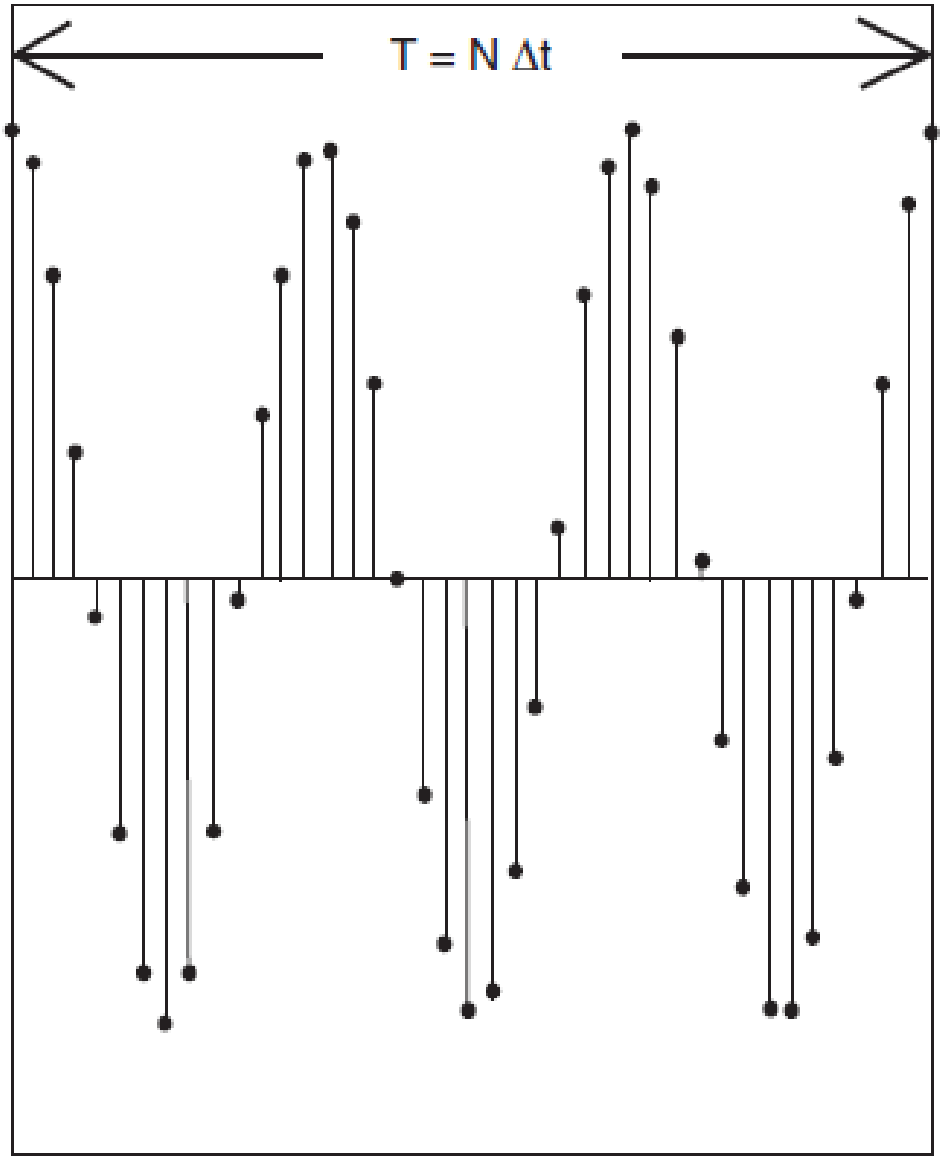
ADC的量程



ADC采样频率



ADC采样频率



ADC采样频率与ADC时钟频率

■ 如果采样频率用符号f_s表示，ADC的时钟频率用f_{ADC}表示，则在连续采样模式（continuous mode）下，它们之间的关系可以表示为：

$$f_s = f_{ADC} / [采样时间[周期]+位数[位]+0.5\ Msps]$$

- ✓ 采样时间是指某个AD通道的转换时间，单位是ADC时钟的周期数。
- ✓ STM32G431中，ADC是12位的，如果采样频率要求很高，也可以降低位数来使用
- ✓ 12位时，如采样时间为2.5个周期，若f_{ADC}为60MHz，则采样频率为60/(2.5+12+0.5)=4Msps
- ✓ 当ADC的位数为12位时，转换时间T_{conv}=采样时间+12.5，单位为ADC的时钟周期。采样时间可配置为：2.5/6.5/12.5/24.5/47.5/92.5/247.5/640.5
- ✓ 思考：ADC位数、分辨率和精确度的关系

生成代码分析

- 保存硬件配置界面（*.ioc），启动代码生成
- Core->Src，打开main.c
- 由于希望在按键中断时，启动ADC采样过程，所以，首先定义外部中EXTI的回调函数。这个回调函数可以写在main.c文件后面的一个注释对中
- 先重定义EXTI回调函数

重定义回调函数

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
	HAL_ADC_Start(&hadc1);
	HAL_ADC_PollForConversion(&hadc1, 10);
	ADC1ConvertedValue = HAL_ADC_GetValue(&hadc1);
	if (ADCxConvertedValue > 2048)
		HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
	else
		HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
	printf("ADCResult = %d \r\n", ADC1ConvertedValue);
}
```

回调函数：启动ADC

```
HAL_ADC_Start(&hadc1);
HAL_ADC_PollForConversion(&hadc1, 10);
ADC1ConvertedValue = HAL_ADC_GetValue(&hadc1);
```

- 启动ADC：HAL_ADC_Start(ADC_HandleTypeDef *hadc)。此函数只有一个参数，就是ADC结构体变量。由于在硬件配置中用了ADC1，所以自动生成的代码中，已经给定义了它的结构体变量，就是hadc1，这里直接使用即可
- HAL_ADC_PollForConversion(ADC_HandleTypeDef *hadc, uint32_t Timeout)

  这个函数实际上是以查询方式等待AD转换过程的结束

  该函数的第二个参数是 Timeout，单位为ms
- HAL_ADC_GetValue(ADC_HandleTypeDef *hadc)，读取AD转换结果

变量定义

- 用变量ADC1ConvertedValue来存放AD转换的结果。需要在main.c中定义该变量，可以放到main函数前的注释对中：

```
/* USER CODE BEGIN PV */
uint16_t ADC1ConvertedValue = 0;
/* USER CODE END PV */
```

- 在回调函数HAL_GPIO_EXTI_Callback()中，根据AD采样值的大小，来控制发光二极管的亮灭

配置printf函数

- 回调函数中，使用了printf函数，将AD转换的结果通过串口送出
- 要使用printf函数从串口送出数据，需要在main.c中，将stdio.h包含进来

```
/* USER CODE BEGIN Includes */
#include "stdio.h"
/* USER CODE END Includes */
```

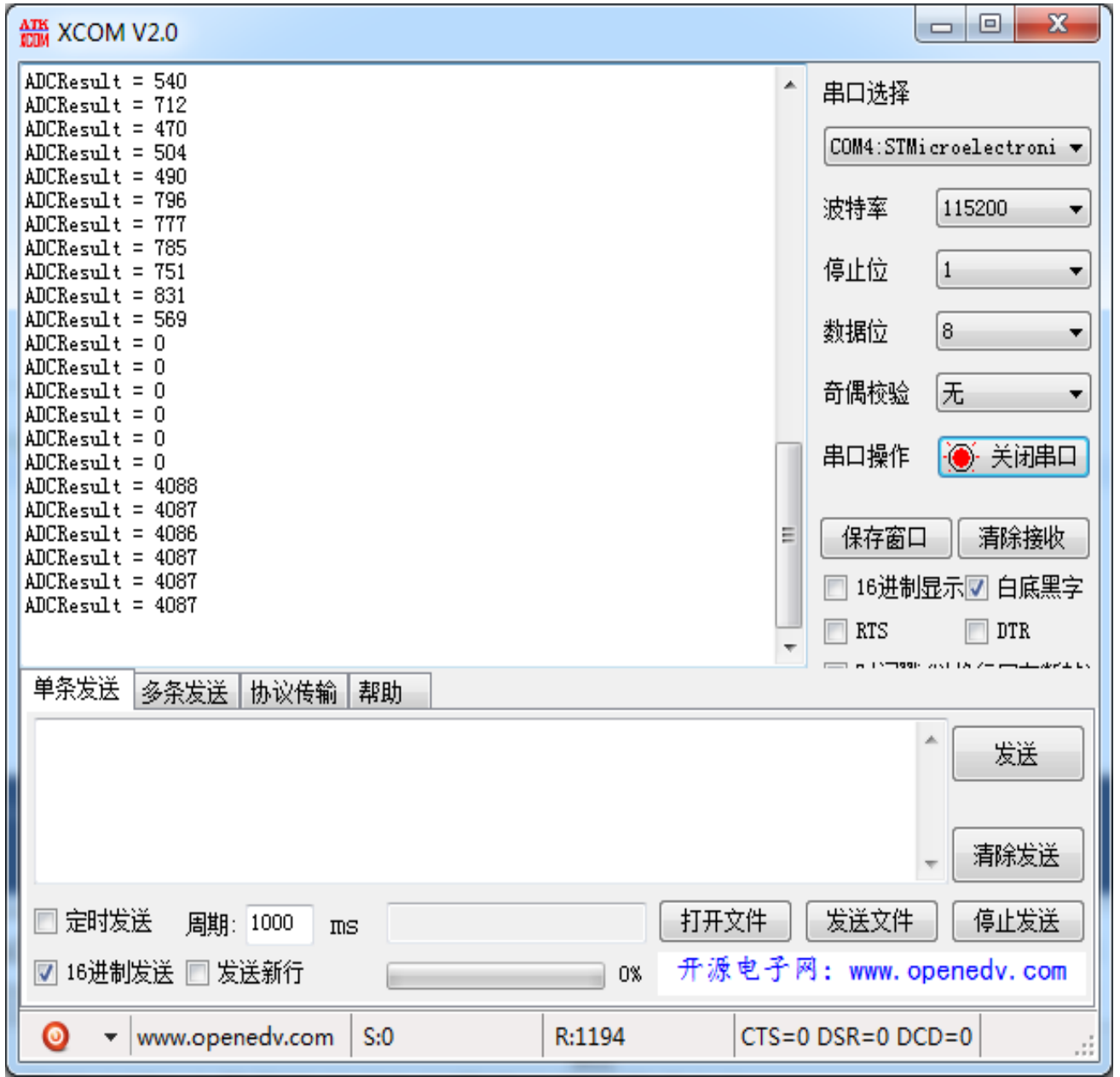
- 还要给出putchar函数的定义，并将它与上面定义的回调函数，一起放到注释对中

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
        ……
}
int __io_putchar(int ch)
{
  HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFF);
  return ch;
}
/* USER CODE END 4 */
```

编译、下载，运行，查看结果

- 编译工程，并下载到硬件中运行
- 打开串口助手小程序，设置好串口端口和波特率等参数，点击“打开串口”
- 按下NUCLEO-G431RB板上的B1按键，会看到串口助手的数据接收栏会有数据送上来，但每次送上的数据不同。这是因为还没有给ADC1的输入引脚施加信号，它是浮空的。浮空时测得的信号不会是0，可能是一个不确定的值
- 可以分别用跳线将PA0连接到GND和VCC（3.3V）上，并操作B1按键，可以看出在连接到GND时，送上的是0，连接到VCC时，会送上来一个接近4095的数

编译、下载，运行，查看结果


XCOM V2.0
ADCResult = 540
ADCResult = 712
ADCResult = 470
ADCResult = 504
ADCResult = 490
ADCResult = 796
ADCResult = 777
ADCResult = 785
ADCResult = 751
ADCResult = 831
ADCResult = 569
ADCResult = 0
ADCResult = 0
ADCResult = 0
ADCResult = 0
ADCResult = 0
ADCResult = 0
ADCResult = 4088
ADCResult = 4087
ADCResult = 4086
ADCResult = 4087
ADCResult = 4087
ADCResult = 4087
串口选择
COM4:STMicroelectroni
波特率
115200
停止位
1
数据位
8
奇偶校验
无
串口操作
关闭串口
保存窗口
清除接收
16进制显示
白底黑字
RTS
DTR
单条发送
多条发送
协议传输
帮助
发送
清除发送
定时发送
周期:
1000
ms
打开文件
发送文件
停止发送
16进制发送
发送新行
0%
开源电子网：www.openedv.com
www.openedv.com
S:0
R:1194
CTS=0 DSR=0 DCD=0

另一种实现方式

- 上面的实现中，把启动ADC、查询转换是否完成、读取AD转换结果等操作放到了按键B1中断的回调函数中，实际上，也可以将这些操作放到main函数的while(1)循环中。可以修改代码如下：

```
/* Infinite loop */
  while (1)
  {
    /* USER CODE BEGIN 3 */
          HAL_ADC_Start(&hadc1);
          HAL_ADC_PollForConversion(&hadc1, 10);
          ADC1ConvertedValue = HAL_ADC_GetValue(&hadc1);
          printf("ADCResult = %d \r\n", ADC1ConvertedValue);
          if (ADC1ConvertedValue > 2048)
             HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
          else
             HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
          HAL_Delay(500);
  }
  /* USER CODE END 3 */
```

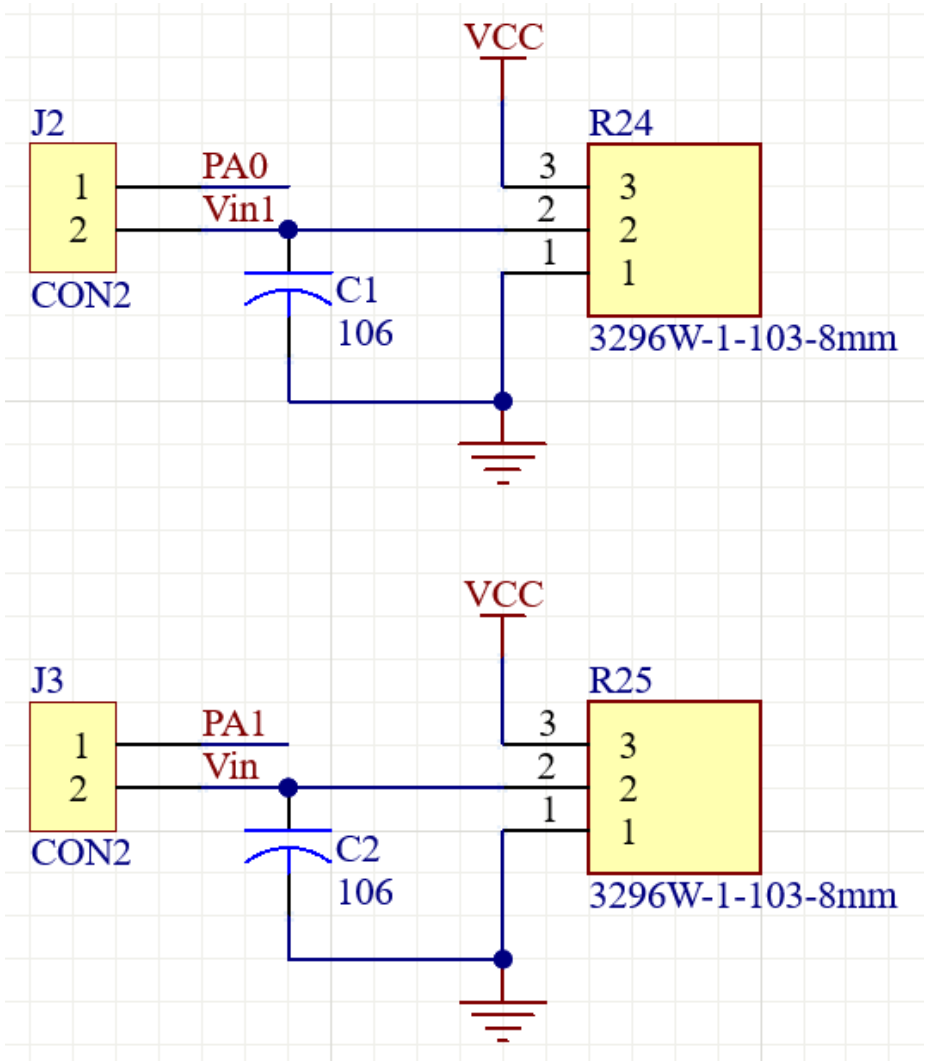
为什么加这一句

练习7：ADC

任务7.1、实现上述单通道ADC单次采样。

（1）分别测量将ADC的输入接GND和3.3V，查看结果；

（2）利用扩展板上的分压器，通过调节电位器旋钮，查看结果。




R24
R25
POWER

ADC的连续工作模式

ADC的连续工作模式

- 在上面的例子中，在每次操作按键B1后，都要重复进行启动ADC、等待转换，然后读取转换结果这些步骤
- 在ADC1的配置界面中，ADC设置（ADC_Settings）列表中的连续转换模式（Continuous Conversion Mode）参数，在默认时是被禁止（Disabled）的。此时ADC为单次转换模式，即ADC只会进行一次转换，所以在每次读取AD转换结果前，需要先启动转换

ADC的连续工作模式

- 在实际应用时，很多时候是让ADC连续进行采样。此时，可以使能ADC的连续转换模式，在此转换模式下，ADC完成一次转换后，会开始新的转换
- 如何使用ADC的连续转换模式？

建立新工程

时钟源和Debug模式配置

■选择时钟源和Debug模式

✓ System Core->RCC->将高速时钟（HSE）选择为Crystal/Ceramic Resonator

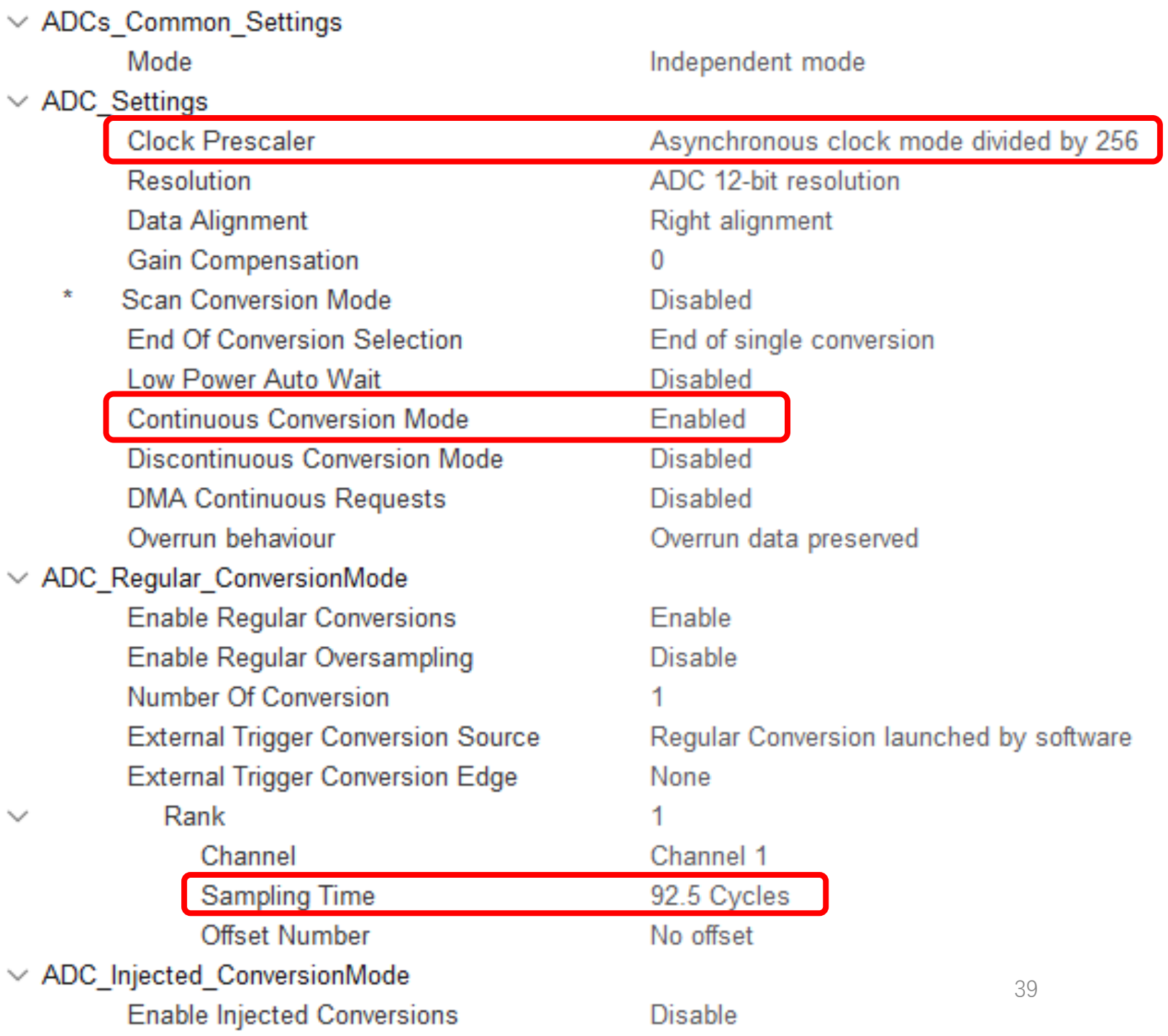
✓ SYS->Debug选择为Serial Wire

配置串口

- 选择“Connectivity”中的USART2，在其模式（Mode）栏选“异步”（Asynchronous），其他参数设置均保持默认（波特率115200），不开启中断。
- 将USART2的两个引脚PA2和PA3均设置为上拉（Pull-up）。

配置ADC

- 选择Analog中的ADC1
- 选择IN1为IN1 Single-ended
- 将ADC的Clock Prescaler选择为：Asynchronous clock mode divided by 256
- 将ADC_Settings参数栏中Continuous Conversion Mode选择为Enable
- 将位于Rank下的采样时间选择为92.5周期。这个参数决定着ADC的转换时间。



单选题 1分

设置

此时ADC的采样频率是多少?

Clock Prescaler	Asynchronous clock mode divided by 256
Sampling Time	92.5 Cycles

A 约1.26kHz

B 约12.6kHz

C 约126kHz

D 约126Hz

$f_s = f_{ADC}$ / [采样时间[周期]+位数[位]+0.5 Msps]

提交

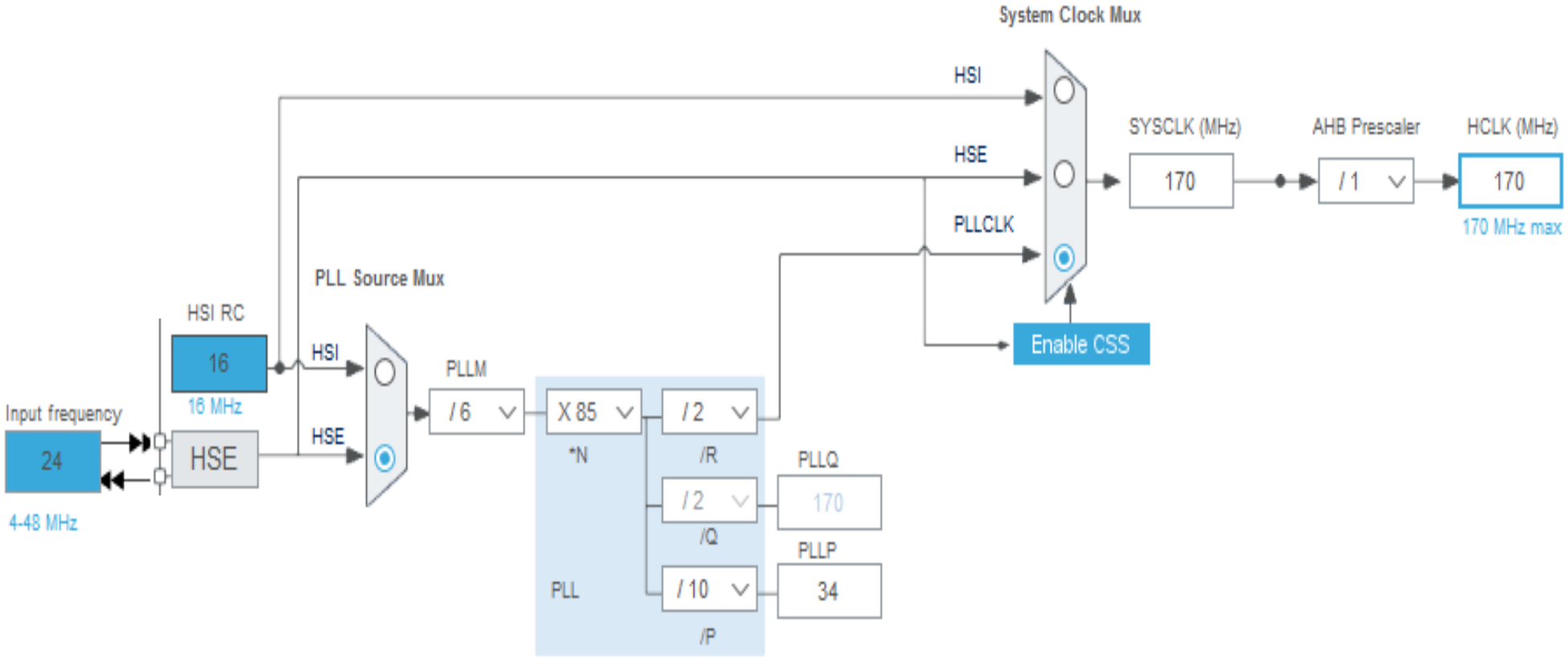
配置中断

- 打开ADC配置界面中的NVIC设置（NVIC Settings），使能ADC1的中断（ADC1 and AD2 global interrupt，ADC1与ADC2共用一个中断类型）
- 在“System Core”中的NVIC中，使能中断EXTI line[15:10] interrupts。将其抢占式优先级设置为2
- 将ADC1中断的优先级设置为1，此处将ADC的优先级设置得比外部中断的优先级高

时钟配置

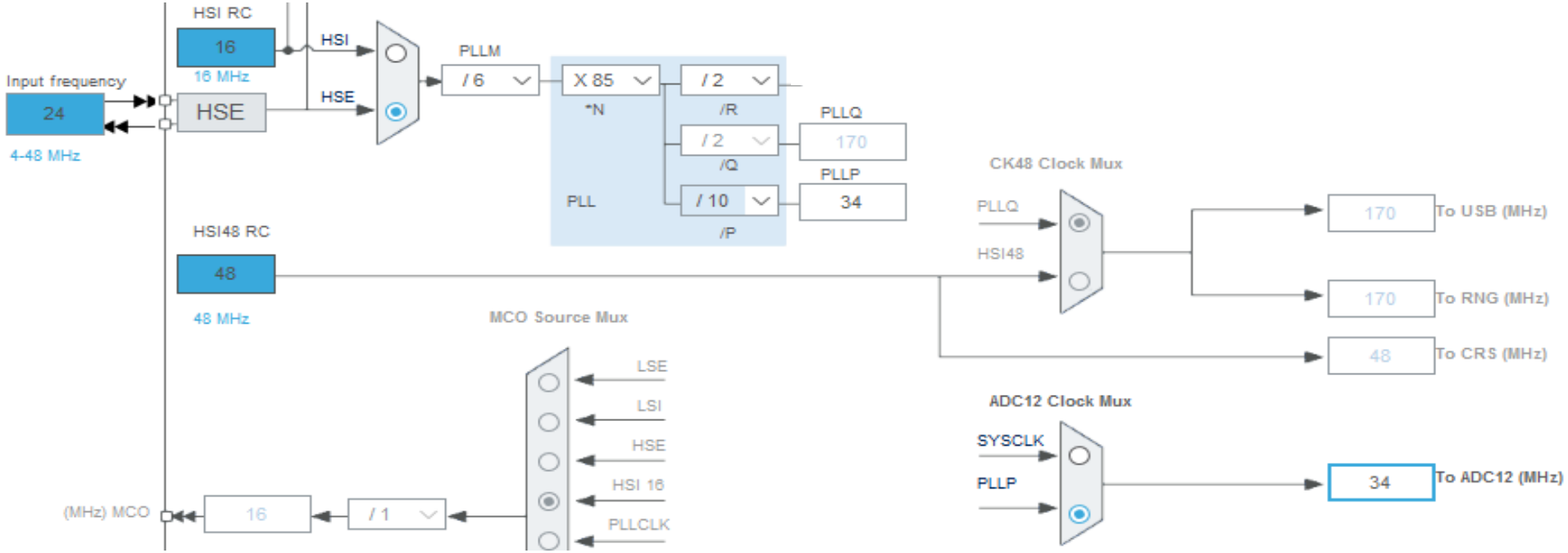
■配置系统时钟

✓ 在“Clock Configuration”中，将系统时钟（SYSCLK）配置为170Mhz



ADC时钟设置

■ 配置ADC的时钟为34MHz



■ 保存硬件配置界面（*.ioc），启动代码生成

ADC中断的使用

■ 前面介绍了外部中断、串口中断、定时器中断等的使用方法，关于ADC中断的使用，与它们也是类似的。关键点有两点：

✓ 1、重写回调函数，

✓ 2、在主程序初始化时开启中断

■ ADC中断相关的回调函数可以用：

HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)。

■ 启动ADC中断的库函数为：

HAL_ADC_Start_IT(ADC_HandleTypeDef *hadc)。

定义存储转换结果的数组

- 首先定义一个数组，用于存储AD转换结果。每执行一次回调函数，就将此次AD转换结果存入数组，直到存满
- 随后，在主程序中，通过串口送出数组中存储的数据

为此，需要在主程序中定义几个变量（放到注释对中）：

```
/* USER CODE BEGIN PV */
uint16_t ADC1ConvertedData[ADC_CONVERTED_DATA_BUFFER_SIZE];
uint16_t ADC1Data_index = 0;
uint8_t ADCSampleFlag = 0;
/* USER CODE END PV */
```

- 数组长度ADC_CONVERTED_DATA_BUFFER_SIZE可以定义到main.h中：

```
/* USER CODE BEGIN Private defines */
#define ADC_CONVERTED_DATA_BUFFER_SIZE (uint16_t) 65
/* USER CODE END Private defines */
```

重定义回调函数

- 在main.c中定义回调函数HAL_ADC_ConvCpltCallback()

```
/* USER CODE BEGIN 4 */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
        ADC1ConvertedData[ADC1Data_index] = HAL_ADC_GetValue(&hadc1);
        if (ADCSampleFlag == 0)
                ADC1Data_index++;
        if (ADC1Data_index == ADC_CONVERTED_DATA_BUFFER_SIZE)
        {
                ADCSampleFlag = 1;
                ADC1Data_index = 0;
        }
}
/* USER CODE END 4 */
```

在主程序中写发送数据代码

- 变量ADCSampleFlag: 当数组存满后，该标志位置1，等待串口发送数组中的数据；一旦数据发送完毕，再将该变量赋值为0，继续更新数组中数据
- 将数据发送代码放置到main函数中的while(1)循环中:

```
while (1)
{
 /* USER CODE BEGIN 3 */
 if (ADCSampleFlag == 1)
 {
        for(uint16_t i = 1; i < ADC_CONVERTED_DATA_BUFFER_SIZE; i++)
        {
          printf("ADC1ConvertedData[%d] = %d\r\n", i, ADC1ConvertedData[i]);
        }
        ADCSampleFlag = 0;
 }
 HAL_Delay(1000);
}
/* USER CODE END 3 */
```

校验ADC与使能ADC中断

- 还要在主程序初始化中使能ADC中断。将HAL_ADC_Start_IT()函数与HAL_ADCEx_Calibration_Start()放到while(1)之前，MX_ADC1_Init()之后的注释对中

```
/* USER CODE BEGIN 2 */
  HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
  HAL_ADC_Start_IT(&hadc1);
/* USER CODE END 2 */
```

配置printf函数

- 由于上述代码中使用了printf函数通过串口发送数据，还要包含头文件stdio.h：

```
/* USER CODE BEGIN Includes */
#include "stdio.h"
/* USER CODE END Includes */
```

- 还要定义putchar函数，可将该函数与ADC的回调函数一起，放到main.c中的注释对中：

```
/* USER CODE BEGIN 4 */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
        ……
}
int __io_putchar(int ch)
{
  HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFF);
  return ch;
}
/* USER CODE END 4 */
```

查看结果

- 编译工程，下载代码到硬件中，并将程序运行起来。施加信号到ADC输入端PA0上。
- 打开串口助手，可以收到送上来的数据：

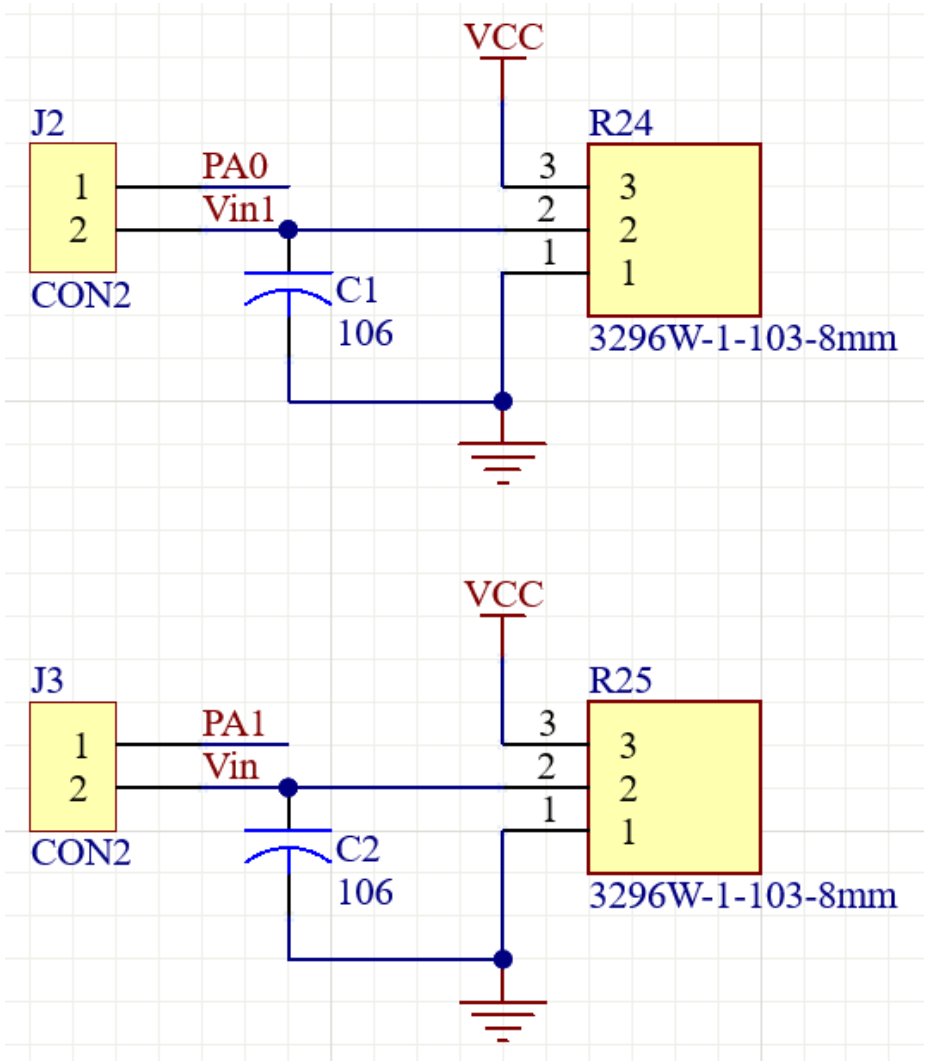
ADC1ConvertedData[1] = 1600
ADC1ConvertedData[2] = 1732
……
ADC1ConvertedData[64] = 2094

- 每次会送上来64个AD转换结果

练习7：ADC

任务7.2、实现上述单通道ADC连续采样。

利用扩展板上的分压器，通过调节电位器旋钮，利用串口助手，查看结果。



如何实测AD采样频率

- 在前面配置AD和时钟参数时，计算得到ADC的采样频率约为1.26kHz，具体是多少呢？是否可以通过实际测量进行验证呢？
- 这里介绍一种简单的方式。不过，需要另外配置一个IO（输出），譬如PC3，可以在回调函数HAL_ADC_ConvCpltCallback()加入控制PC3输出状态的语句，通过示波器测量就可以测得实际的采样频率了。
- 譬如，可以在AD的回调函数中加入如下语句：

```
HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_3);
```

- 通过PC3测得的信号频率应该是采样频率的一半。

可用示波器测量

用定时器控制ADC采样

如何控制ADC的采样频率

- 上面的例子中，通过使能AD配置参数中的Continuous Conversion Mode，结合ADC中断，实现了连续采样。但这种方式有个缺点，就是不能灵活配置ADC的采样频率
- 上例中，ADC的采样频率约为1.26kHz，此频率是通过设置ADC的时钟频率和采样时间得到的
- 实际中，有时希望ADC以给定的采样频率转换数据，譬如1kHz。在这种情况下，靠配置AD时钟频率和采样时间的方法，就会比较不方便
- 是不是可以采用前面介绍的定时器呢？

建立新工程

时钟源与Debug模式配置

■选择时钟源和Debug模式

✓ System Core->RCC->将高速时钟（HSE）选择为Crystal/Ceramic Resonator

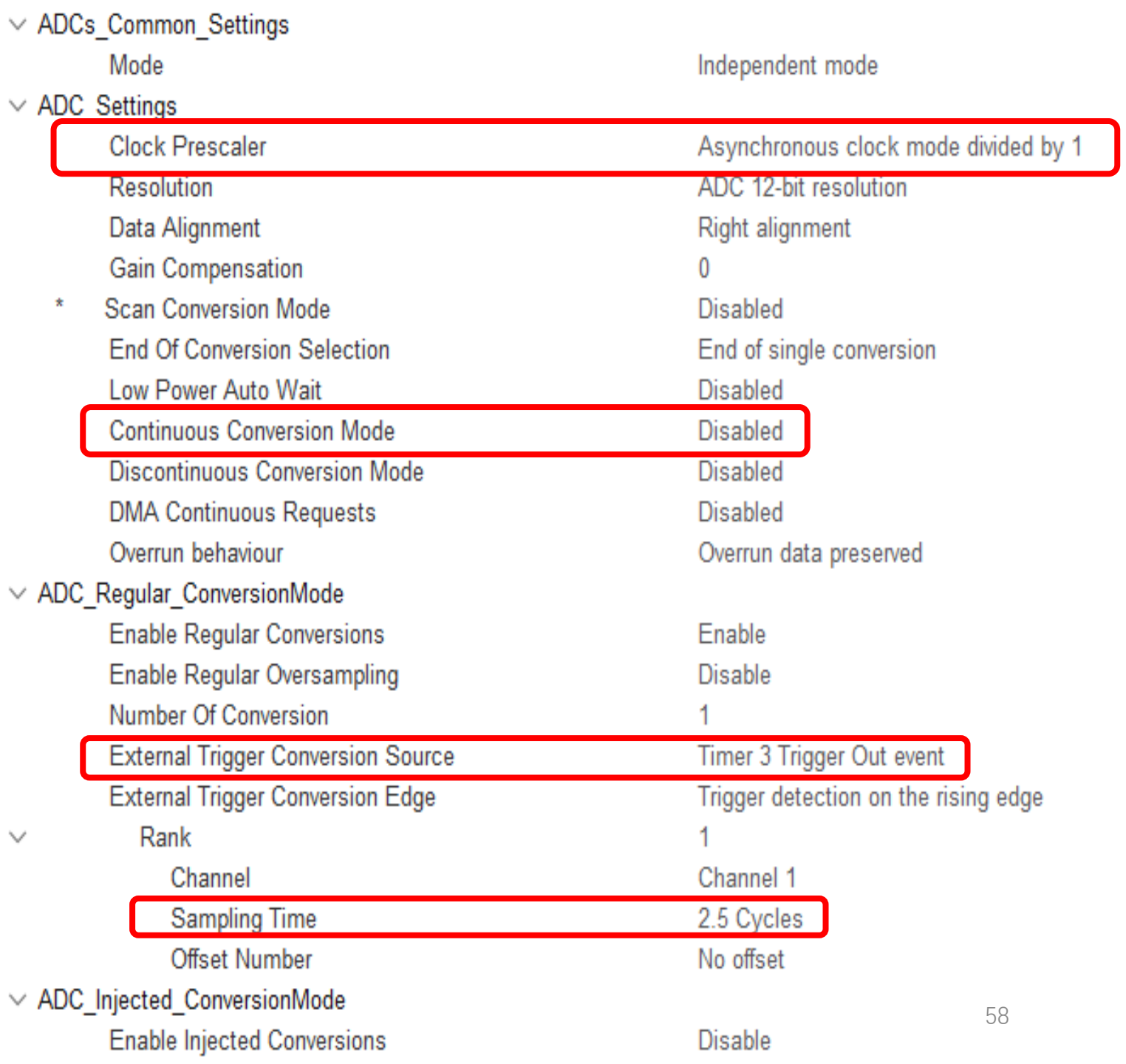
✓ SYS->Debug选择为Serial Wire

配置串口

- 选择“Connectivity”中的USART2，在其模式（Mode）栏选“异步”（Asynchronous），其他参数设置均保持默认（波特率115200），不开启中断。
- 将USART2的两个引脚PA2和PA3均设置为上拉（Pull-up）。

配置ADC

- 选择Analog中的ADC1
- 选择IN1为IN1 Single-ended
- 将ADC的Clock Prescaler选择为：Asynchronous clock mode divided by 1
- 将ADC_Settings参数栏中Continuous Conversion Mode选择为Disabled
- 在ADC_Regular_Conversion Mode栏，将External Trigger Conversion Source选择为：Timer 3 Trigger Out event
- 将位于Rank下的采样时间选择为2.5周期。这个参数决定着ADC的转换时间

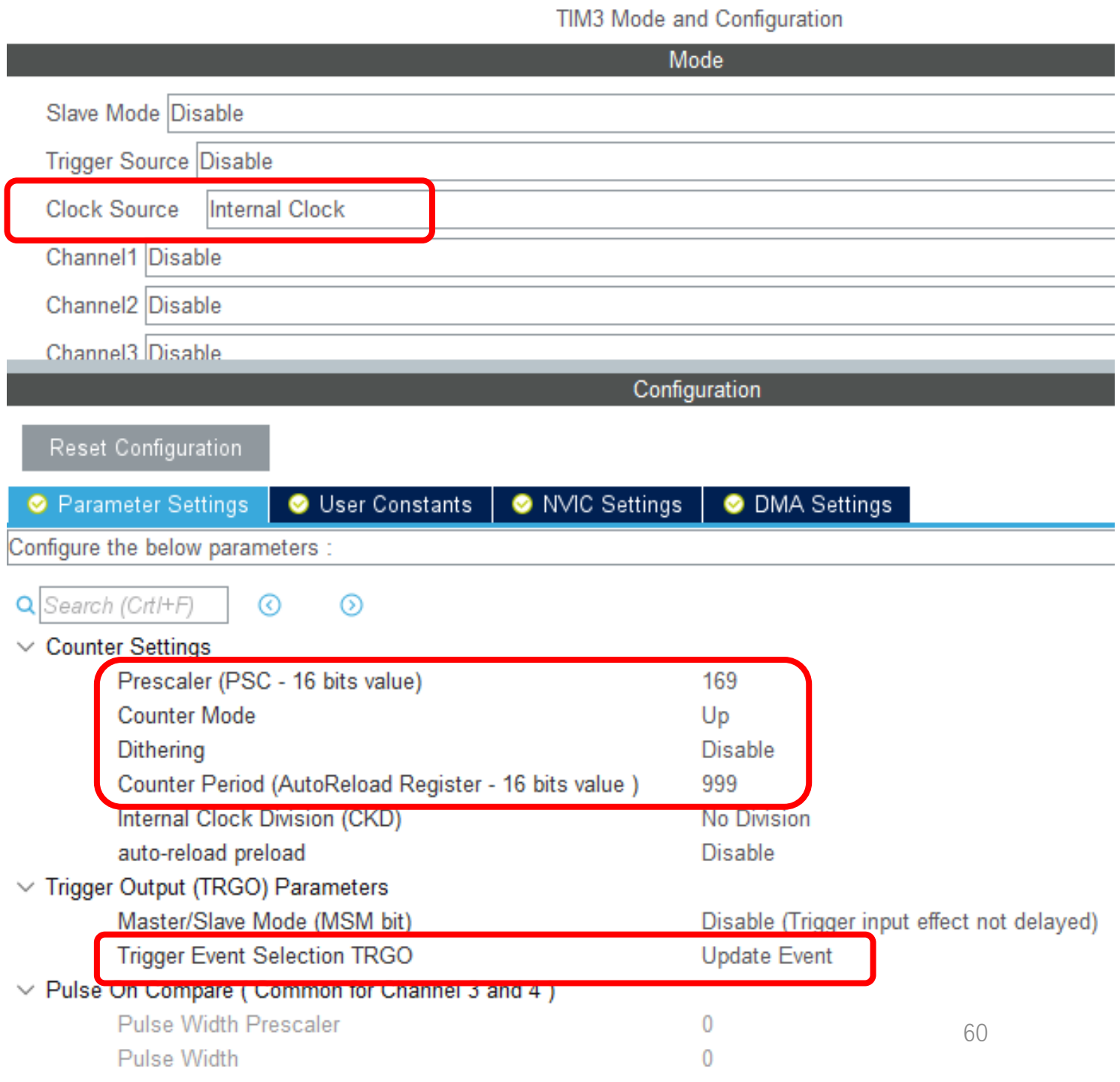


配置中断

- 打开ADC配置界面中的NVIC设置（NVIC Settings），使能ADC1的中断（ADC1 and AD2 global interrupt，ADC1与ADC2共用一个中断类型）
- 在“System Core”中的NVIC中，将ADC1中断的优先级设置为1。由于仅用此一个中断，所以优先级也可用默认值0

配置定时器TIM3

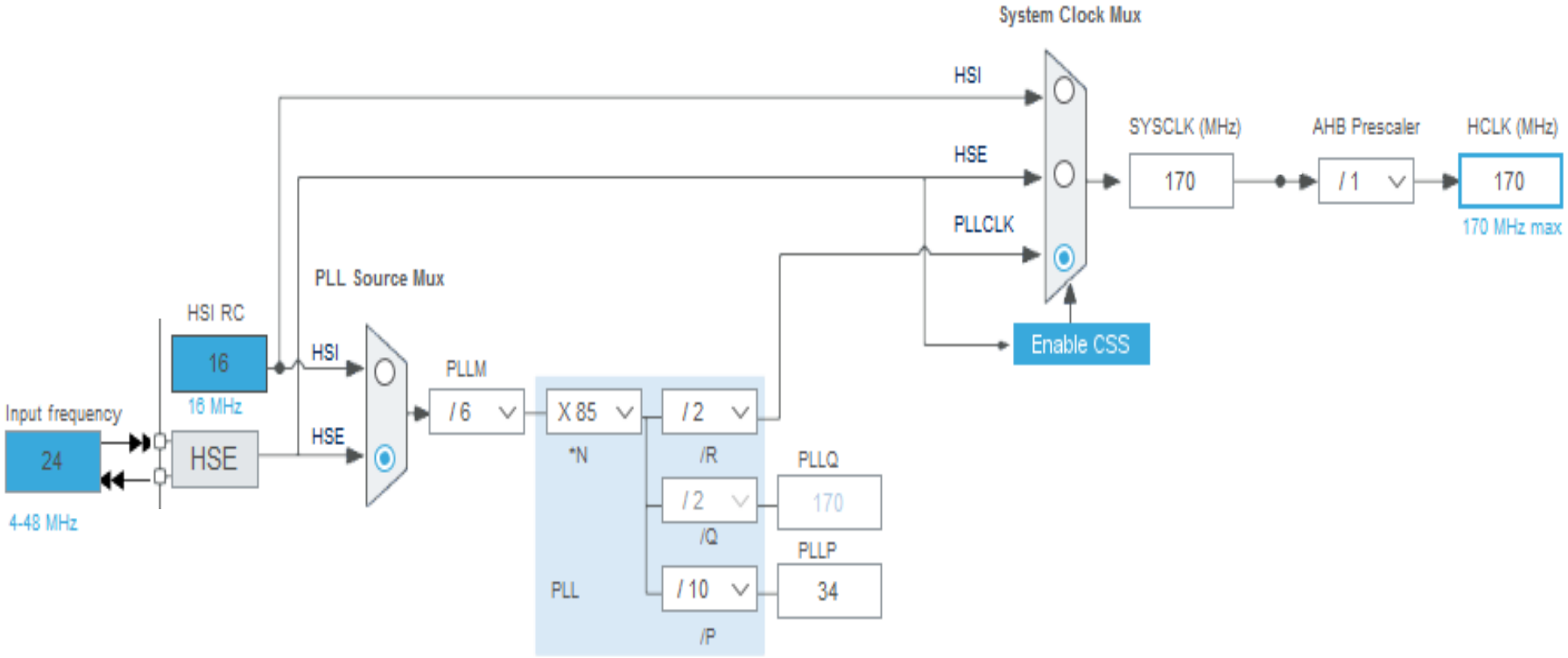
- 在TIM3的Mode栏，只需选择Clock Source为Internal Clock
- 在下面的配置栏，设置计数器的预分频因子为169
- 计数器周期设置为999
- 在Trigger Output参数栏，将Trigger Event Selection TRGO选为Update Event



时钟配置

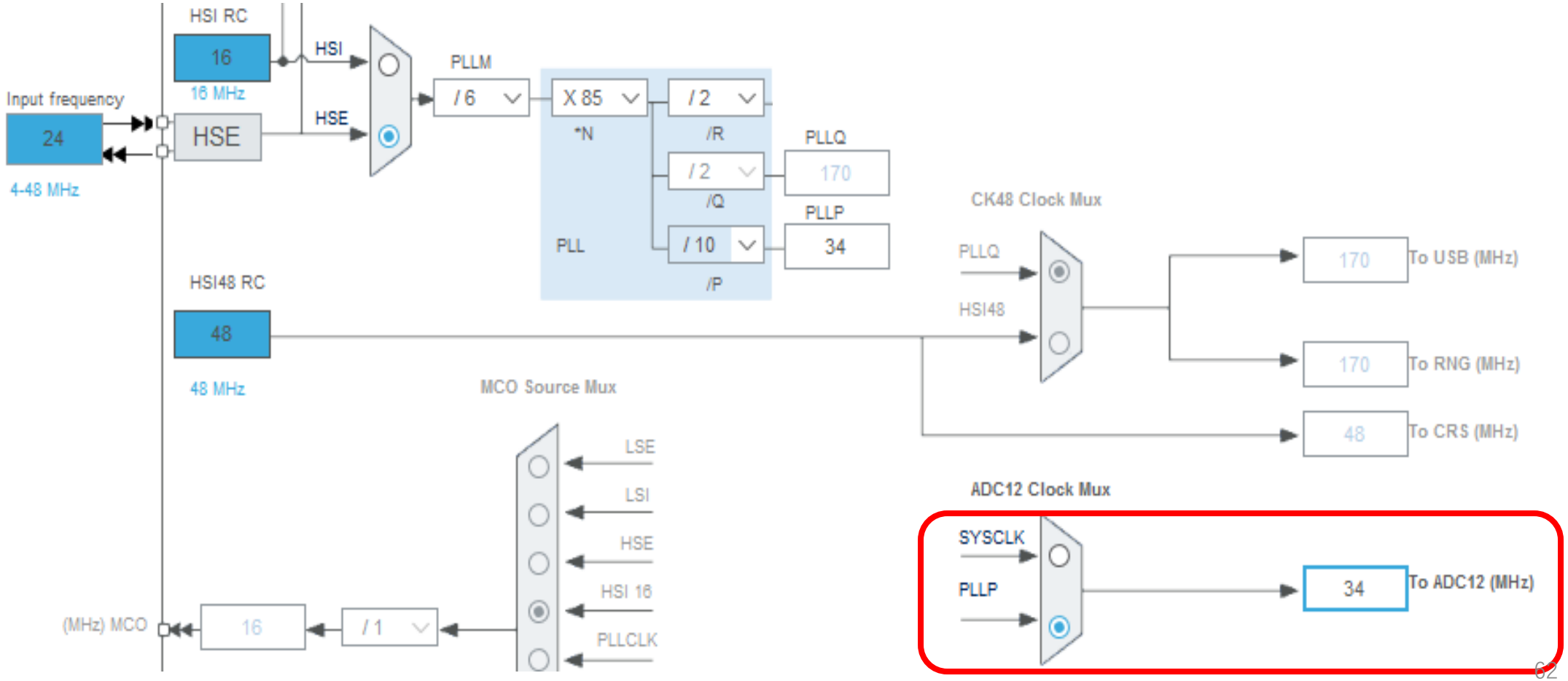
■配置系统时钟

✓ 在“Clock Configuration”中，将系统时钟（SYSCLK）配置为170Mhz



ADC时钟设置

■ 配置ADC的时钟为34MHz



修改代码

- 保存硬件配置界面（*.ioc），启动代码生成
- 打开main.c

重新定义ADC回调函数

- 在主程序中重写回调函数HAL_ADC_ConvCpltCallback()和串口发送数据的putchar函数，将它们放到main.c中的注释对中

```
/* USER CODE BEGIN 4 */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
        ADC1ConvertedData[ADC1Data_index] = HAL_ADC_GetValue(&hadc1);
        if (ADCSampleFlag == 0)
                ADC1Data_index++;
        if (ADC1Data_index == ADC_CONVERTED_DATA_BUFFER_SIZE)
        {
                ADCSampleFlag = 1;
                ADC1Data_index = 0;
        }
}
int __io_putchar(int ch)
{
    HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFF);
    return ch;
}
/* USER CODE END 4 */
```

在主程序中写发送数据代码

- 将数据发送代码放置到main函数中的while(1)循环中:

```
while (1)
{
    /* USER CODE BEGIN 3 */
   if (ADCSampleFlag == 1)
   {
         for(uint16_t i = 1; i < ADC_CONVERTED_DATA_BUFFER_SIZE; i++)
         {
          printf("ADC1ConvertedData[%d] = %d\r\n", i, ADC1ConvertedData[i]);
         }
         ADCSampleFlag = 0;
   }
   HAL_Delay(1000);
}
/* USER CODE END 3 */
```

变量定义

- 在主程序中定义变量（放到注释对中）：

```
/* USER CODE BEGIN PV */
uint16_t ADC1ConvertedData[ADC_CONVERTED_DATA_BUFFER_SIZE];
uint16_t ADC1Data_index = 0;
uint8_t ADCSampleFlag = 0;
/* USER CODE END PV */
```

- 数组长度ADC_CONVERTED_DATA_BUFFER_SIZE可以定义到main.h中：

```
/* USER CODE BEGIN Private defines */
#define ADC_CONVERTED_DATA_BUFFER_SIZE (uint16_t) 65
/* USER CODE END Private defines */
```

- 在main.c中，包含头文件stdio.h

```
/* USER CODE BEGIN Includes */
#include "stdio.h"
/* USER CODE END Includes */
```

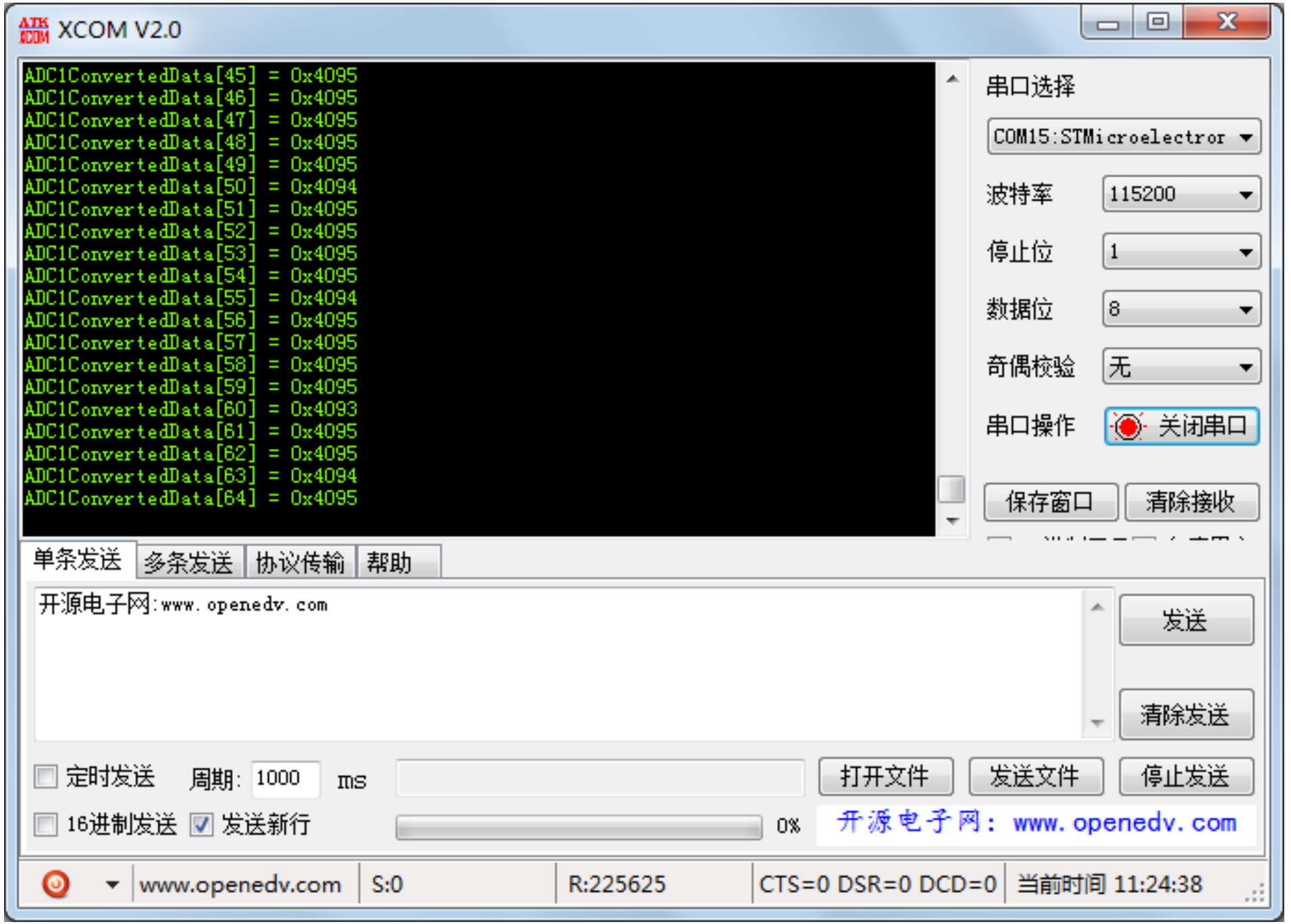
校验ADC、使能ADC中断和开启定时器

- 最后，在主程序初始化中使能ADC中断，并开启定时器TIM3
- 将HAL_ADC_Start_IT()、HAL_ADCEx_Calibration_Start()和HAL_TIM_Base_Start()放到while(1)之前，MX_ADC1_Init()之后的注释对中：

```
/* USER CODE BEGIN 2 */
  HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
  HAL_ADC_Start_IT(&hadc1);
  HAL_TIM_Base_Start(&htim3);
 /* USER CODE END 2 */
```

查看结果

- 编译工程，下载代码到硬件中，并将程序运行起来。施加信号到ADC输入端PA0上
- 打开串口助手，查看结果


XCOM V2.0
ADC1ConvertedData[45] = 0x4095
ADC1ConvertedData[46] = 0x4095
ADC1ConvertedData[47] = 0x4095
ADC1ConvertedData[48] = 0x4095
ADC1ConvertedData[49] = 0x4095
ADC1ConvertedData[50] = 0x4094
ADC1ConvertedData[51] = 0x4095
ADC1ConvertedData[52] = 0x4095
ADC1ConvertedData[53] = 0x4095
ADC1ConvertedData[54] = 0x4095
ADC1ConvertedData[55] = 0x4094
ADC1ConvertedData[56] = 0x4095
ADC1ConvertedData[57] = 0x4095
ADC1ConvertedData[58] = 0x4095
ADC1ConvertedData[59] = 0x4095
ADC1ConvertedData[60] = 0x4093
ADC1ConvertedData[61] = 0x4095
ADC1ConvertedData[62] = 0x4095
ADC1ConvertedData[63] = 0x4094
ADC1ConvertedData[64] = 0x4095
串口选择
COM15:STMicroelectror
波特率
115200
停止位
1
数据位
8
奇偶校验
无
串口操作
关闭串口
保存窗口
清除接收
单条发送
多条发送
协议传输
帮助
开源电子网:www.openedv.com
发送
清除发送
定时发送
周期:
1000
ms
打开文件
发送文件
停止发送
16进制发送
发送新行
0%
开源电子网: www.openedv.com
www.openedv.com
S:0
R:225625
CTS=0 DSR=0 DCD=0
当前时间 11:24:38

练习7：ADC

任务7.3、（1）实现上述单通道ADC连续采样（用定时器TIM3控制采样频率）。

（2）修改代码，接收到串口命令，再送出数据。给发送数据加上帧头和帧尾（ 0x53和0x45 ）。

利用扩展板上的分压器，通过调节电位器旋钮，利用串口助手，查看结果。

练习7：ADC

任务7.4、在7.3的基础上修改程序，实现下述功能：

运行程序后，得到一组ADC采样值：

（1）AD采样值数据中，查找等于特定值XXX（如4095）的采样值的数量；

（2）AD采样值数组中，找到所有相同的采样值及数量。

通过串口助手，送出上述查询结果。特定值XXX，通过串口助手传递给单片机。

提交网络学堂：每个子任务的工程文件（压缩），代码有简单注释

谢　谢!