第十六章 SAADC 采集

ADC 为 Analog-to-Digital Converter 的缩写，指模/数转换器或者模数转换器 。是指将连续变化的模拟信号转换为离散的数字信号的器件。真实世界的模拟信号，例如温度、压力、声音或者图像等，需要转换成更容易储存、处理和发射的数字形式。模/数转换器可以实现这个功能。在 nrf52832 中的 ADC 为一个逐次逼近（successive-approximation）模拟数字转换器。具体的属性如下所示（参考手册 P345 页）：

1: 8/10/12 分辨率，采用过采样可达到 14 位分辨率。

2：多达 8 个输入通道：

单端输入时时有 1 个通道，2 个通道组成差分输入。

单端和差分输入时可以配置成扫描模式。

3：满量程输入范围为 0 和 VDD

4：可以通过软件触发采样任务启动采样，也可以使用低功耗 32.768Khz 的 RTC 定时器 H 或者更加精确的的 1/16Mhz 定时器通过 PPI 来触发采样任务，使得 SAADC 具有非常灵活的采样频率。

5：单次的采集模式只使用一个采集通道。

6：扫描模式是按照顺序采样一系列通道。频道直接的采样延迟位 t ack + t conv。用户通过配置 t ack 可以使频道直接的间隔时间不同

7：可以通过 EasyDMA 可以直接将采样的结果保存到 RAM 内。

8：中断发生在单次采样和换成满事件时。

9： Samples stored as 16-bit 2’ s complement values for differential and single-ended sampling

10：无需外部定时器就可以实现连续采样。

11：可以配置通道输入负载电阻。

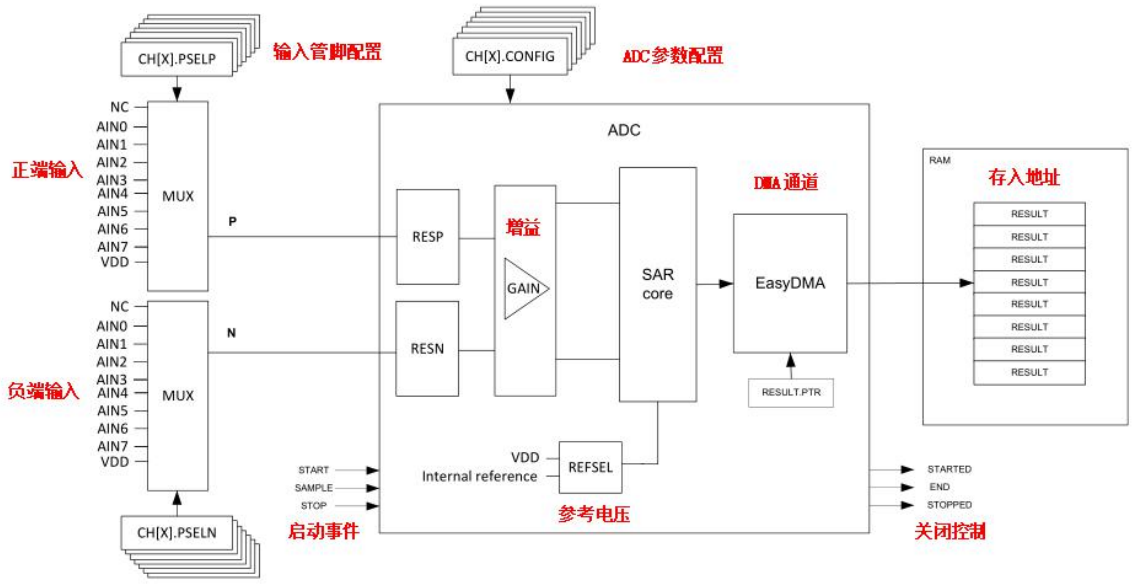
16.1 原理分析

下面来分析下 nrf52832 内部 ADC 的基本原理，其内部 SAADC 支持多达到 8 个外部模拟输入通道。我们需要关心的 ADC 的几个关键参数：

16.1.1：采样模式

nrf52832 内部 ADC 的采样信号输入可以通过 8 个外部模拟输入通道进行采集，但是其 ADC 内部事件上有 16 个通道和 VDD，如下图所示。其中 8 个通道为正端输入（N），另外 8 个通道为负端输入（P）。因此信号采样的模式可以分为单独输入和差分输入两种方式。

Channel input	Source	Connectivity
CH[n].PSELP	AIN0...AIN7	Yes(any)
CH[n].PSELP	VDD	Yes
CH[n].PSELN	AIN0...AIN7	Yes(any)
CH[n].PSELN	VDD	Yes



默认状态下，ADC 的配置模式为单端输入，因为在配置寄存器 CH[n].CONFIG 中设置的 MODE 为 0。需要配置为差分输入的时候，把 MODE 设置为 1。单端模式的时候，ADC 内部把负极接入到地，如下表所示：

F	RW	MODE		Enable differential mode
			SE	0	Single ended, PSELN will be ignored, negative input to ADC shorted to GND
			Diff	1	Differential

而差分模式则是把负极通过负向端输入，通过计算两端的差值来换算出采样结果。具体计算公式我们等下来讨论。

16.1.2：信号增益

如上面的 ADC 内部结构图，采样数据通过采集输入端进入后会通过一个增益 GAIN 对信号进行放大，这个增益在配置寄存器 CH[n].CONFIG 中设置的 GAIN 位进行设置，如下表所示，可以设置为几个值：1/6、1/5、1/4、1/3、1/2、1、2、4 这些参数。

C	RW	GAIN		Gain control
			Gain1_6	0	1/6
			Gain1_5	1	1/5
			Gain1_4	2	1/4
			Gain1_3	3	1/3
			Gain1_2	4	1/2
			Gain1	5	1
			Gain2	6	2
			Gain4	7	4

16.1.3：参考电压：

ADC 的参考电压可以采用两种方式，一个是内部参考电压，为 0.6V 大小。另一种采用 VDD/4 为参考电压。通过配置寄存器 CH[n].CONFIG 中设置的 RESEL 位进行配置，如下表所示：

D	RW	REFSEL		Reference control
		Internal	0	Internal reference (0.6 V)
		VDD1_4	1	VDD/4 as reference

输入采样电压范围：当采样内部电压作为参考时，范围为$^{\pm 0.6\ V}$；当使用 VDD 电压作为参考电压时，输入范围为$^{\pm VDD/4}$。Gain 增益参数可以来调整输入范围，如下图所示：

```
Input range = (+- 0.6 V or +-VDD/4)/Gain
```

比如，如果选择 VDD 作为参考电压，信号输入采样单端输入，同时增益为 1/4，那么输入信号范围为：

```
Input range = (VDD/4)/(1/4) = VDD
```

如果选择内部电压作为参考，信号为单端输入，放大增益为 1/6，那么输入电压范围为：

```
Input range = (0.6 V)/(1/6) = 3.6 V
```

但是要注意，AIN0-AIN7 输入范围不能超过 VDD，低于 VSS。

16.1.4：采样精度：

ADC 的分辨率对采样结果也是至关重要的，nrf52832 内部 ADC 可以配置为多种采样分辨率，一般情况下可以设置为 8/10/12 位，采用过采样可达到 14 位分辨率。通过设置 RESOULUTION 寄存器来配置，如下表所示：

Bit number				31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Id																																A	A	A
Reset 0x00000001				**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
Id	**RW**	**Field**	**Value Id**	**Value**																													
A	RW	VAL																															
			8bit	0																													
			10bit	1																													
			12bit	2																													
			14bit	3																													

通过这几个步骤，采集结果就可以出来了，那么输出结果按照下面公式进行计算：

$$RESULT = [V(P) - V(N)] * GAIN/REFERENCE * 2^{(RESOLUTION - m)}$$

公式中：
V(P)为：ADC 正端输入
V(N)为：ADC 负端输入
GAIN 为：为增益值
REFERENCE 为：为参考电压
RESOULUTION 为：采样精度
M 为：采样模式，当单端输入的时候为 0，差分输入的时候为 1。

16.1.5：工作模式：

SAADC 有三种工作模式，分别为：单次转换模式，连续转换模式，扫描模式。

16.1.5.1 单次转换模式 One-shot mode：

要配置 nRF52832 的 SAADC 为单次转换模式，通过配置 CH[n].PSELP,CH[n].PSELN 和 CH[n].CONFIG 寄存器，来使能一个 ADC 通道，使得 ADC 工作于单次模式。当触发采样任务后，ADC 开始采样输入电压，采样时间通过 CH[n].CONFIG.TACQ 来设置，当 DONE 事件发生表示一次采集完成。

在没有过采样发生的时候，RESULTDONE event 事件等同于 DONE 事件。在实际采样数据通过 EasyDMA 保存到 RAM 之前，这两个事件都会发生。

16.1.5.2 连续转换模式 Continuous mode：

连续采样模式能够通过内部定时器实现定时采样，或者触发 SAMPLE 任务通过 PPI 链接一个通用寄存器来实现。

SAMPLERATE 寄存器能够被用于用本地的定时器代替独立的 SAMPLE tasks。当设置 SAMPLERATE.MODE 位设置为 Timers, 单次的 SAMPLE task 任务来启动 SAADC。一个 STOP task 停止采样。在 SAMPLERATE.CC 来控制采样率.

回到普通采样，设置 SAMPLERATE.MODE 返回 Task。如下图所示：

Bit number				31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Id				B A A A A A A A A A
Reset 0x00000000				0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Id	**RW**	**Field**	**Value Id**	**Value**	**Description**
A	RW	CC		[80..2047]	Capture and compare value. Sample rate is 16 MHz/CC
B	RW	MODE			Select mode for sample rate control
			Task	0	Rate is controlled from SAMPLE task
			Timers	1	Rate is controlled from local timer (use CC to control the rate)

采样的频率，SAMPLERATE.CC 内定时时间越短，频率越快。同时，这个还和通道数量有关系，公式如下所示：

$$f_{SAMPLE} < 1/[CHANNELS \times (t_{ACQ} + t_{conv})]$$

在没有过采样发生的时候，RESULTDONE event 事件等同于 DONE 事件。在实际采样数据通过 EasyDMA 保存到 RAM 之前，这两个事件都会发生。

16.1.5.3 扫描采样

频道被使能如果 CH[n].PSELP 被设置。如果超过 1 个通道被使能，那么 ADC 就进入扫描模式。区别与单次模式，就是频道大于 1。

在扫描模式下，一个 SAMPLE task 任务触发每个被使能的通道进行转换，所有频道转换需要的时间按下面公式计算:

$$Total\ time < Sum(CH[x].t_{ACQ}+t_{CONV}),\ x=0..enabled\ channels$$

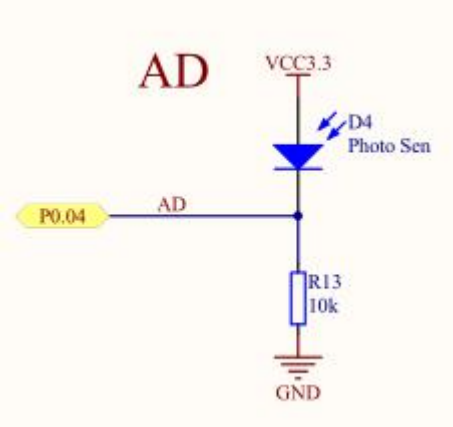
在没有过采样发生的时候，RESULTDONE event 事件等同于 DONE 事件。在实际采样数据通过

EasyDMA 保存到 RAM 之前，这两个事件都会发生。

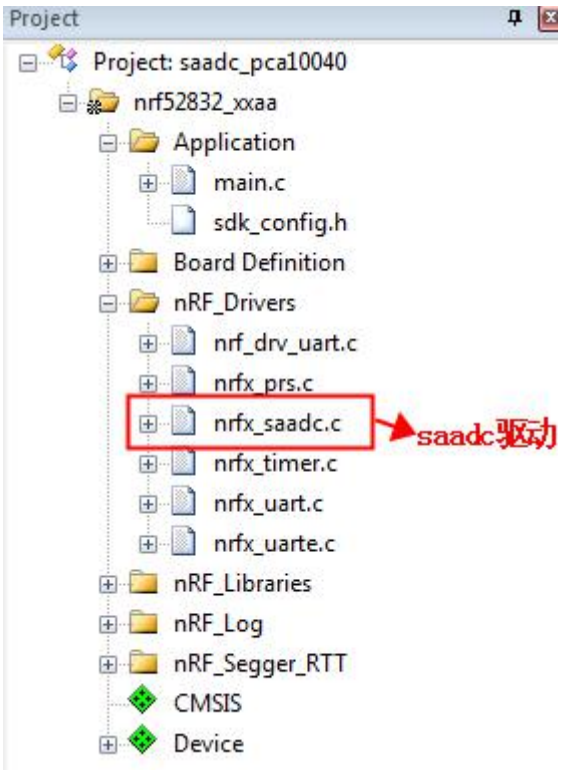
16.2 应用实例编写

16.2.1 ADC 的单次采样

如下图所示：青云 QY-nRF52832 开发板上，光敏电阻通过管脚 P0.04 接入到 nrf52832 中，P0.04 作为 ADC 采样管脚。



在代码文件中，实验建立了一个演示历程，我们打开看看需要那些库文件。打开 arm5 文件夹中的工程项目如下：



这个工程可以在串口例子中进行添加，在工程中，和 ADC 相关的驱动文件为：nrfx_saadc.c

驱动文件，这个函数需要加入工程项目中，同时注意配置驱动路径。

下面我们首先来进行 saadc 的初始化配置，设置代码如下 ：

```
01.    void saadc_init(void)
02. {
03.      ret_code_t err_code;
04.     //adc 通道配置
05.      nrf_saadc_channel_config_t channel_config =
06.      NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE(NRF_SAADC_INPUT_AIN2);
07.     //adc 初始化
08.      err_code = nrf_drv_saadc_init(NULL, saadc_callback);
09.      APP_ERROR_CHECK(err_code);
10.     //adc 通道初始化
11.      err_code = nrf_drv_saadc_channel_init(0, &channel_config);
12.      APP_ERROR_CHECK(err_code);
13. }
14.
```

第 5 行：通过设置一个结构体，设置配置函数，官方给了两个结构体来配置单端输入和差分输入。在函数头文件 nrf_drv_saadc.h 文件中采用：

NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE：表示单端输入配置。

NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_DIFFERENTIAL：表示差分输入配置。

比如单端输入配置结构体如下表示所示：

```
01. #define NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE(PIN_P) \
02. {                                                       \
03.     .resistor_p = NRF_SAADC_RESISTOR_DISABLED,          \
04.     .resistor_n = NRF_SAADC_RESISTOR_DISABLED,          \
05.     .gain       = NRF_SAADC_GAIN1_6,                    \
06.     .reference  = NRF_SAADC_REFERENCE_INTERNAL,         \
07.     .acq_time   = NRF_SAADC_ACQTIME_10US,               \
08.     .mode       = NRF_SAADC_MODE_SINGLE_ENDED,          \
09.     .pin_p      = (nrf_saadc_input_t)(PIN_P),           \
10.     .pin_n      = NRF_SAADC_INPUT_DISABLED              \
11. }
```

上面的参数配置代码解释如下：

第 3 行：正端输入：SAADC 的旁路电阻关

第 4 行：负端输入：SAADC 的旁路电阻关

第 5 行：增益：SAADC 的增益为 1/6

第 6 行：参考电压值：采用芯片内部参考电压

第 7 行：采样时间：10us

第 8 行：模式：单端输入

第 9 行：正端输入引脚：输入管脚

第 10 行：负端输入引脚：输入管脚关闭。

根据硬件配置，管脚 P0.04 是可以配置为 AD 输入管脚 AIN2，可以查看数据手册 P27 页管脚配置，如下图所示：

5	P0.03	Digital I/O	General purpose I/O pin.
	AIN1	Analog input	SAADC/COMP/LPCOMP input.
6	P0.04	Digital I/O	General purpose I/O pin.
	AIN2	Analog input	SAADC/COMP/LPCOMP input.
7	P0.05	Digital I/O	General purpose I/O pin.
	AIN3	Analog input	SAADC/COMP/LPCOMP input.
8	P0.06	Digital I/O	General purpose I/O pin.

```
15.     //adc 初始化
16.     err_code = nrf_drv_saadc_init(NULL, saadc_callback);
```

nrf_drv_saadc_init 函数对 saadc 进行初始化，第一样形参 NULL,表示使用默认配置参数 NRFX_SAADC_DEFAULT_CONFIG，如下图所示：

```
01. #define NRFX_SAADC_DEFAULT_CONFIG
02. {                \
03.     .resolution    = (nrf_saadc_resolution_t)NRFX_SAADC_CONFIG_RESOLUTION, \\分辨率
04.     .oversample = (nrf_saadc_oversample_t)NRFX_SAADC_CONFIG_OVERSAMPLE, \\过采
    样     .interrupt_priority = NRFX_SAADC_CONFIG_IRQ_PRIORITY, \\saadc 中断优先级
05.     .low_power_mode = NRFX_SAADC_CONFIG_LP_MODE    \\低功耗模式
06. }
```

默认配置参数在 sdk_config.h 文件中，其中 ADC 的分辨率为 10bit，不发送过采样，中断优先级设为低，不开动低功耗模式。具体配置如下代码所示：

```
07. // <o> NRFX_SAADC_CONFIG_RESOLUTION    - Resolution
08.
09. // <0=> 8 bit
10. // <1=> 10 bit
11. // <2=> 12 bit
12. // <3=> 14 bit
13.
14. #ifndef NRFX_SAADC_CONFIG_RESOLUTION
15. #define NRFX_SAADC_CONFIG_RESOLUTION 1
16. #endif
17.
18. // <o> NRFX_SAADC_CONFIG_OVERSAMPLE    - Sample period
19.
20. // <0=> Disabled
21. // <1=> 2x
22. // <2=> 4x
23. // <3=> 8x
24. // <4=> 16x
25. // <5=> 32x
26. // <6=> 64x
27. // <7=> 128x
28. // <8=> 256x
29.
30. #ifndef NRFX_SAADC_CONFIG_OVERSAMPLE
31. #define NRFX_SAADC_CONFIG_OVERSAMPLE 0
```

```
#endif

// <q> NRFX_SAADC_CONFIG_LP_MODE  - Enabling low power mode


#ifndef NRFX_SAADC_CONFIG_LP_MODE
#define NRFX_SAADC_CONFIG_LP_MODE 0
#endif

// <o> NRFX_SAADC_CONFIG_IRQ_PRIORITY  - Interrupt priority

// <0=> 0 (highest)
// <1=> 1
// <2=> 2
// <3=> 3
// <4=> 4
// <5=> 5
// <6=> 6
// <7=> 7

#ifndef NRFX_SAADC_CONFIG_IRQ_PRIORITY
#define NRFX_SAADC_CONFIG_IRQ_PRIORITY 7
#endif
```

nrf_drv_saadc_init 函数第二个形参为 saadc_callback，设置为 saadc 回调函数，回调中断函数可以为空，什么都不执行。如果有中断任务需要执行，可以在 saadc_callback 内写中断函数。

```
err_code = nrf_drv_saadc_channel_init(0, &channel_config);
APP_ERROR_CHECK(err_code);
```

nrf_drv_saadc_channel_init 函数初始化 ADC 的通道函数，第一个形参设置为通道值，当设置为 0 的时候，表示选择通道 0。第二个形参为前面配置的通道参数结构体作为指针调入。

那么 adc 配置好后，直接可以在主函数里进行调用，同时采用串口输出，代码如下所示：

```
int main(void)
{   nrf_saadc_value_t  saadc_val;
    float  val;  //保存 SAADC 采样数据计算的实际电压值
    uart_config();//配置串口

    printf("\n\rSAADC HAL simple example.\r\n");
    saadc_init();//saadc 初始化
    while(1)
    {
        //启动一次 ADC 采样。
            nrf_drv_saadc_sample_convert(0,&saadc_val);
            //串口输出 ADC 采样值。
            val = saadc_val * 3.6 /1024;
            printf(" %.3fV\n",  val);
```

```
15.                 //延时 300ms，方便观察 SAADC 采样数据
16.                 nrf_delay_ms(500);
17.
18.         }
19. }
```

主函数中，因为我们没有采用中断采集，那么就通过一个循环扫描采集数据，在 while 循环下延迟 500ms 采集一次电压值。

实验下载到青云 nRF52832 开发板后，通过串口来观察采集的电压大小，为了方便计算，我们可以根据前面的计算公式计算出来电压，

$$RESULT = [V(P) - V(N)] * GAIN/REFERENCE * 2^{(RESOLUTION - m)}$$

VP 为 要采样的电压，VN 为 0，GAIN 为 1/6，REFERENCE 为 0.6，RESOLUTION 为 10，m 为 0，RESULT 是 saadc_val 结果。是那么公式可以化为：

val = saadc_val * 3.6 /1024;

的实验现象如下，可以和万用表对比测量结果：

16.2.2 ADC 的差分采样

在 ADC 采样中，单端输入容易受到外界信号的干扰，而采样差分输入，两个输入信号的相减的差值作为最后的结果，可以有效的互相抵消外界干扰。对比上面单端输入，改变如下位置：

```
01. void saadc_init(void)
02. {
03.         ret_code_t err_code;
04. //配置 ADC 输入频道（要改动的）
05.  nrf_saadc_channel_config_t channel_config =
    NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_DIFFERENTIAL
```

```
06.                             (NRF_SAADC_INPUT_AIN2,NRF_SAADC_INPUT_AIN0);
07.    //初始化 ADC
08.        err_code = nrf_drv_saadc_init(NULL, saadc_callback);
09.        APP_ERROR_CHECK(err_code);
10.    //初始化 ADC 通道配置
11.        err_code = nrf_drv_saadc_channel_init(0, &channel_config);
12.        APP_ERROR_CHECK(err_code);
13. }
```

用结构体 NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_DIFFERENTIAL 表示差分输入配置。同时配置两个输入通道 NRF_SAADC_INPUT_AIN2 和 NRF_SAADC_INPUT_AIN0，两个输入管脚更具实际需要进行选择，对照手册查找管脚数。配置结构体如下：

```
01. #define NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_DIFFERENTIAL(PIN_P, PIN_N) \
02. {                                                                       \
03.     .resistor_p = NRF_SAADC_RESISTOR_DISABLED,                          \
04.     .resistor_n = NRF_SAADC_RESISTOR_DISABLED,                          \
05.     .gain       = NRF_SAADC_GAIN1_6,                                    \
06.     .reference  = NRF_SAADC_REFERENCE_INTERNAL,                         \
07.     .acq_time   = NRF_SAADC_ACQTIME_10US,                               \
08.     .mode       = NRF_SAADC_MODE_DIFFERENTIAL,                          \
09.     .pin_p      = (nrf_saadc_input_t)(PIN_P),                           \
10.     .pin_n      = (nrf_saadc_input_t)(PIN_N)                            \
11. }
```

解释配置如下：

第 3 行：正端输入：SAADC 的旁路电阻关

第 4 行：负端输入：SAADC 的旁路电阻关

第 5 行：增益：SAADC 的增益为 1/6

第 6 行：参考电压值：采用芯片内部参考电压

第 7 行：采样时间：10us

第 8 行：模式：单端输入

第 9 行：正端输入引脚： 正端输入管脚

第 10 行：负端输入引脚：负端输入管脚。

根据硬件配置，管脚 P0.04 是可以配置为 AD 输入管脚 AIN2，管脚 P0.02 是可以配置为 AD 输入管脚 AIN0。

可以查看数据手册 P27 页管脚配置，如下图所示：

4	P0.02	Digital I/O	General purpose I/O pin.
	AIN0	Analog input	SAADC/COMP/LPCOMP input.
5	P0.03	Digital I/O	General purpose I/O pin.
	AIN1	Analog input	SAADC/COMP/LPCOMP input.
6	P0.04	Digital I/O	General purpose I/O pin.
	AIN2	Analog input	SAADC/COMP/LPCOMP input.
7	P0.05	Digital I/O	General purpose I/O pin.
	AIN3	Analog input	SAADC/COMP/LPCOMP input.
8	P0.06	Digital I/O	General purpose I/O pin.

主函数可以不做任何变化，在主函数里扫描采集数据。实验下载到青云 nRF52832 开发板后，通过串口来观察采集的电压大小，为了方便计算，我们可以根据前面的计算公式计算出来电压，

$$RESULT = [V(P) - V(N)] * GAIN/REFERENCE * 2^{(RESOLUTION - m)}$$

VP 为 要正端采样的电压，VN 为负向端采样电压，GAIN 为 1/6，REFERENCE 为 0.6，RESOLUTION 为 10，m 为 0，RESULT 是 saadc_val 结果。是那么公式可以化为：

差分电压 VP-VN=val = saadc_val * 3.6 /1024;

打开串口助手，输出的实验现象如下，可以和万用表对比测量结果：

16.2.3 EasyDMA 之单缓冲中断采样

一：原理分析：

官方 SDK 中，对 ADC 采样会提供软件缓冲存放转换，缓冲寄存器满了后就会触发中断，在中断内读取缓冲寄存器的值。也就是称为 EasyDMA 方式，这种方式的采样速度大大提高。当 ADC 的采样任务一旦被触发，那么 ADC 的转换结果可以通过 EasyDMA 存储到在 RAM 内的结果缓冲内。

结果缓冲 buffer 的地址位于 RESULT.PTR 寄存器中，RESULT.PTR 寄存器是双缓冲，当 STARTED 时间产生，下一个 START task 产生，该缓冲能够立即更新和做好数据准备。

在官方库函数中，提供函数 nrf_drv_saadc_buffer_convert 实现该功能，下面通过对该函数的详细介绍，让大家理解采用 EasyDMA 方式如何进行配置的。

:采用该函数的时候首先需要在主函数文件中定义几个宏定义参数，分别如下：

```
//设置缓冲的数量，决定要填满几个缓冲后启动中断
#define   SAMPLES_IN_BUFFER   5
//设置缓冲的组数，这里也就是对应使用的通道的数量，本例是单通道，我们只定义一个数值
static nrf_saadc_value_t          m_buffer_pool[SAMPLES_IN_BUFFER];
//采样转换次数
static uint32_t                    m_adc_evt_counter;
```

对应函数 ret_code_t nrf_drv_saadc_buffer_convert(nrf_saadc_value_t * p_buffer, uint16_t size)这样一个完整定义的函数，其中两个形势参数是必须详细理解的。

◆第一形参：nrf_saadc_value_t * p_buffer，在 adc 初始化设置，我们设置如下所示：

```
err_code = nrf_drv_saadc_buffer_convert(m_buffer_pool,SAMPLES_IN_BUFFER);
    APP_ERROR_CHECK(err_code);
```

也就是说，第一个形参用的 m_buffer_pool，这个 m_buffer_pool 是一个数组，数组的长度为 SAMPLES_IN_BUFFER 这么长。这样一个数组，对应一个 adc 的转换通道，你有几个通道，你就是设置几个数组的长度加倍，例如你如果用 2 个通道的，可以把数组设置为：

m_buffer_pool[SAMPLES_IN_BUFFER*2]

这样一二个数组长度放置两个通道的数据。

◆第二个形参 uint16_t size，这个表示数组大小，也就是在 RAM 内给的缓存大小，如果大小为 N，表示 N 个字节大小的缓存，当然这个 N 并不是没有限制的，它取决与寄存器 RESULT.MAXCNT 内的设置。

这两个才是实际是在函数内部的 nrf_saadc_buffer_init 中调用了，这个函数直接配置的是寄存器，非常方便大家理解：

```
01. __STATIC_INLINE void nrf_saadc_buffer_init(nrf_saadc_value_t * buffer, uint32_t num)
02. {
03.     NRF_SAADC->RESULT.PTR = (uint32_t)buffer;
04.     NRF_SAADC->RESULT.MAXCNT = num;
05. }
```

RESULT.PTR 寄存器：

Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Id	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
Reset 0x00000000	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Id	RW	Field	Value Id	Value	Description
A	RW	PTR			Data pointer

RESULT.MAXCNT 寄存器：

Bit number	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Id																	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
Reset 0x00000000	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Id	RW	Field	Value Id	Value	Description
A	RW	MAXCNT			Maximum number of buffer words to transfer

那么对的 EasyDMA 采样方式整个过程可以这样表述出来：

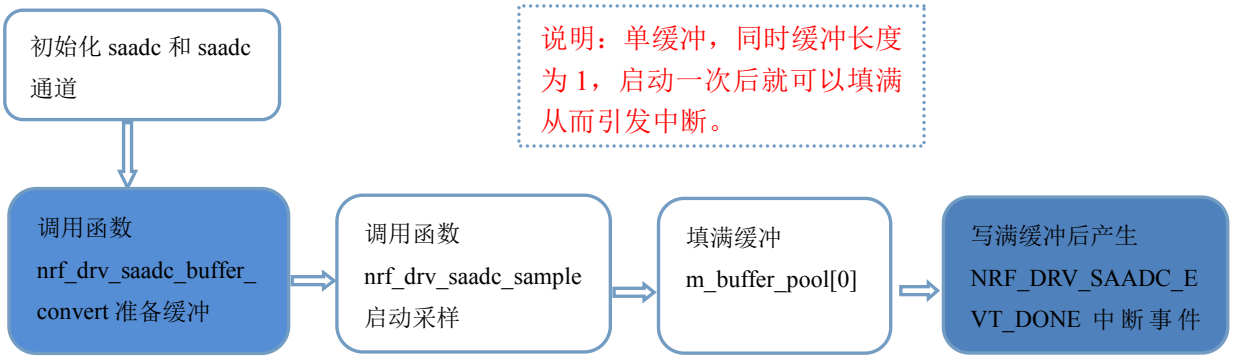
1：EasyDMA 作为 adc 采样的专用通道，负责把采样的数据发送给 RAM 内的结果寄存器 RESULT。

2：根据结果寄存器 RESULT 内的 RESULT.PTR 寄存器和 RESULT.MAXCNT 寄存器，分别决定了数据指针和这个数据指针的大小。

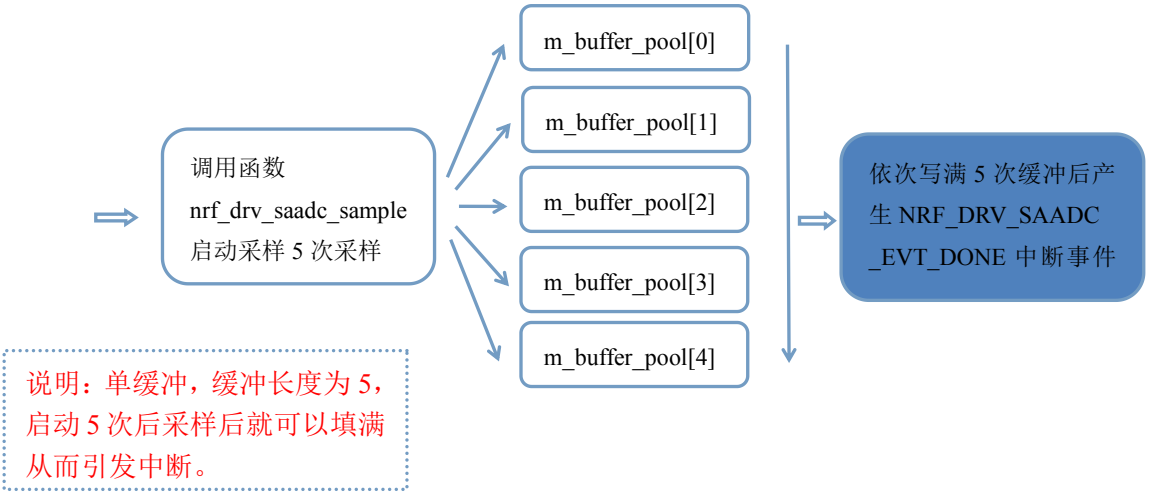
3：只有填满这个数据指针内所有的空间，才能触发中断把转换数据读出，因此整个转换次数=通道数*buff 缓冲大小。

例 1：单通道采集

① 假设采样数据缓冲为 1，通道为 1：

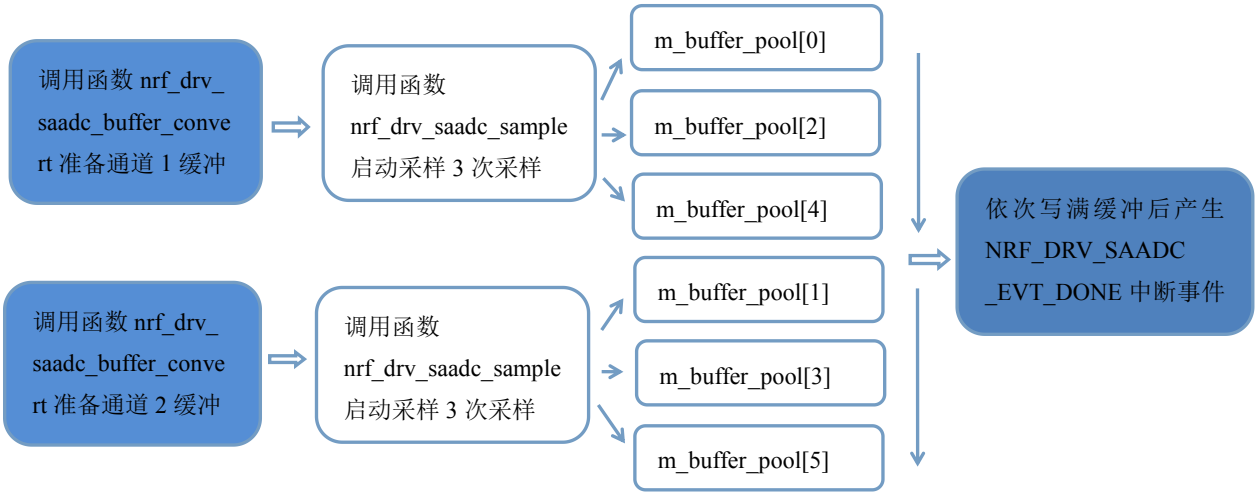


② 假设采样数据缓冲为 5，通道为 1：



例 2：多通道采集

③ 假设采样数据缓冲为 3，通道为 2：



二：实例编程

例 1：单通道采集

已单端输入为例，文件开头做一个宏定义：

```
01. //定义缓冲大小为 1.
02. #define SAMPLES_IN_BUFFER 1
03. //定义缓冲数值
04. static nrf_saadc_value_t          m_buffer_pool[SAMPLES_IN_BUFFER];
05. //定义采样次数
06. static uint32_t                   m_adc_evt_counter;
```

下面我们首先来进行 saadc 的初始化配置，设置代码，初始化通道的部分和前面的配置相同，只是需要多添加缓冲配置函数，如下所示：

```
07.    void saadc_init(void)
08. {
09.       ret_code_t err_code;
10.     //adc 通道配置
11.       nrf_saadc_channel_config_t channel_config =
12.       NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE(NRF_SAADC_INPUT_AIN2);
13.     //adc 初始化
14.       err_code = nrf_drv_saadc_init(NULL, saadc_callback);
15.       APP_ERROR_CHECK(err_code);
16.     //adc 通道初始化
17.       err_code = nrf_drv_saadc_channel_init(0, &channel_config);
18.       APP_ERROR_CHECK(err_code);
19.       //添加缓冲配置函数：
20.         err_code = nrf_drv_saadc_buffer_convert(m_buffer_pool,SAMPLES_IN_BUFFER);
21.       APP_ERROR_CHECK(err_code);
22. }
```

```
23.
```

添加单缓冲如红色字体显示内容，之后在主函数中启动采样，调用 nrf_drv_saadc_sample()函数，开始采样。

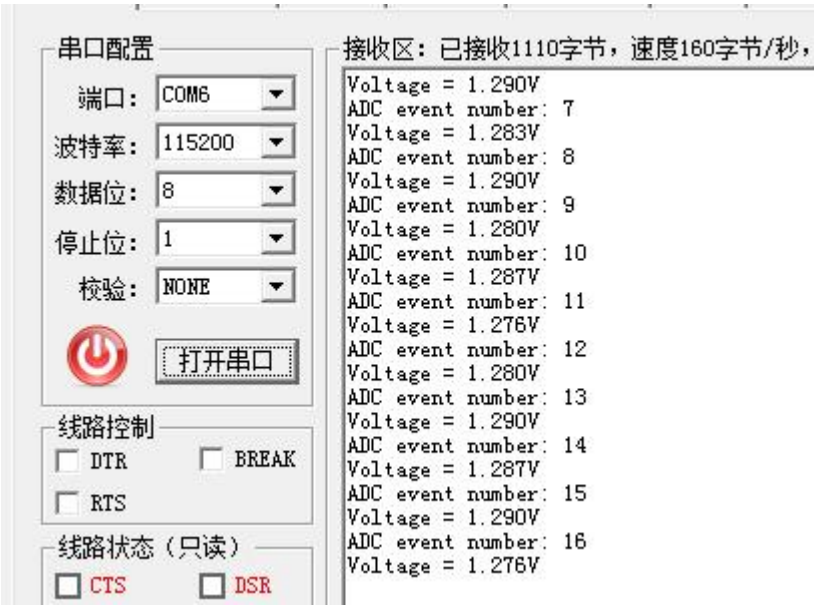
```
24. int main(void)
25. {
26.     uart_config();
27.     printf("\n\rSAADC HAL simple example.\r\n");
28.     saadc_init();

29.     while(1)
30.     {
31.   //启动一次 ADC 采样。
32.         nrf_drv_saadc_sample();
33.         //启动一次 ADC 采样。
34.         //延时 300ms，方便观察 SAADC 采样数据
35.         nrf_delay_ms(300);
36.     }
37. }
```

启动后，触发 adc 采样中断，在中断中进行判断是否缓冲填满，缓冲填满后会启动 NRF_DRV_SAADC_EVT_DONE 事件，同时串口输出转换后的采样电压，内容如下所示：

```
01. void saadc_callback(nrf_drv_saadc_evt_t const * p_event)
02. {
03.     float    val;
04.       //判断是否发送填满缓冲事件，如何发送表示本次采样完成
05.     if (p_event->type == NRF_DRV_SAADC_EVT_DONE)
06.     {
07.         ret_code_t err_code;
08.       //设置好缓存，为下次转换预备缓冲
09.         err_code =
    nrf_drv_saadc_buffer_convert(p_event->data.done.p_buffer,SAMPLES_IN_BUFFER);
10.         APP_ERROR_CHECK(err_code);
11.
12.         int i;
13.       //打印输出采样次数
14.         printf("ADC event number: %d\r\n",(int)m_adc_evt_counter);
15.         for (i = 0; i < SAMPLES_IN_BUFFER; i++)
16.         {
17.       //打印输出，通过前面的转换公式进行电压的转换计算。
18.             val = p_event->data.done.p_buffer[i] * 3.6 /1024;
19.                 printf("Voltage = %.3fV\r\n", val);
20.         }
21.       //采样次数加 1
22.         m_adc_evt_counter++;
```

```
23.         }
24. }
```

打开串口助手，输出的实验现象如下，可以和万用表对比测量结果，同时采样次数也跟着计数：



如果我们改变下最开头的宏定义，把缓冲大小改成 5，那么根据上面的分析，要采样 5 次才会有事件输出，如下变动：

```
38. //定义缓冲大小为 5
39. #define SAMPLES_IN_BUFFER 1
40. //定义缓冲数值
41. static nrf_saadc_value_t          m_buffer_pool[SAMPLES_IN_BUFFER];
42. //定义采样次数
43. static uint32_t                   m_adc_evt_counter;
```

打开串口助手，输出的实验现象如下，采样次数每次都会依次输出 5 个采样电压值，如下所示：

例 2：多通道采集

下面来实现多通道的采集，首先假设我们才有 2 个输入通道，每个通道分配 3 个缓冲大小，首先头文件设置如下：

```
//定义缓冲大小为 6
#define SAMPLES_IN_BUFFER  6
//定义缓冲数值
static nrf_saadc_value_t          m_buffer_pool[SAMPLES_IN_BUFFER];
//定义采样次数
static uint32_t                   m_adc_evt_counter;
```

2 通道，每个通道 3 个缓冲，所以定义的缓冲总数为 2*3=6，然后再 adc 的定义初始化，初始化中需要定义两个通道，如下所示：

```
void saadc_init(void)
{
    ret_code_t err_code;
  //配置通道 0，输入管脚为 AIN2
    nrf_saadc_channel_config_t channel_0_config =
        NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE(NRF_SAADC_INPUT_AIN2);
  //配置通道 1，输入管脚为 AIN0
     nrf_saadc_channel_config_t channel_1_config =
        NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE(NRF_SAADC_INPUT_AIN0);
    //adc 初始化
    err_code = nrf_drv_saadc_init(NULL, saadc_callback);
    APP_ERROR_CHECK(err_code);
    //adc 通道初始化，带入前面的通道配置结构体
    err_code = nrf_drv_saadc_channel_init(0, &channel_0_config);
    APP_ERROR_CHECK(err_code);
    err_code = nrf_drv_saadc_channel_init(1, &channel_1_config);
    APP_ERROR_CHECK(err_code);
  //添加缓冲配置函数：
   err_code = nrf_drv_saadc_buffer_convert(m_buffer_pool,SAMPLES_IN_BUFFER);
    APP_ERROR_CHECK(err_code);
}
```

对比单通道配置和多通道配置，区别就如上红色标示部分，需要配置多个通道，进行多通道的初始化过程。配置好后，主函数的设置没有任何变化，直接调用启动 adc 采样函数，代码如下：

```
int main(void)
{
    uart_config();
    printf("\n\rSAADC HAL simple example.\r\n");
    saadc_init();//调用 adc 初始化函数
    while(1)
    {
  //启动一次 ADC 采样。
            nrf_drv_saadc_sample();
            //启动一次 ADC 采样。
```

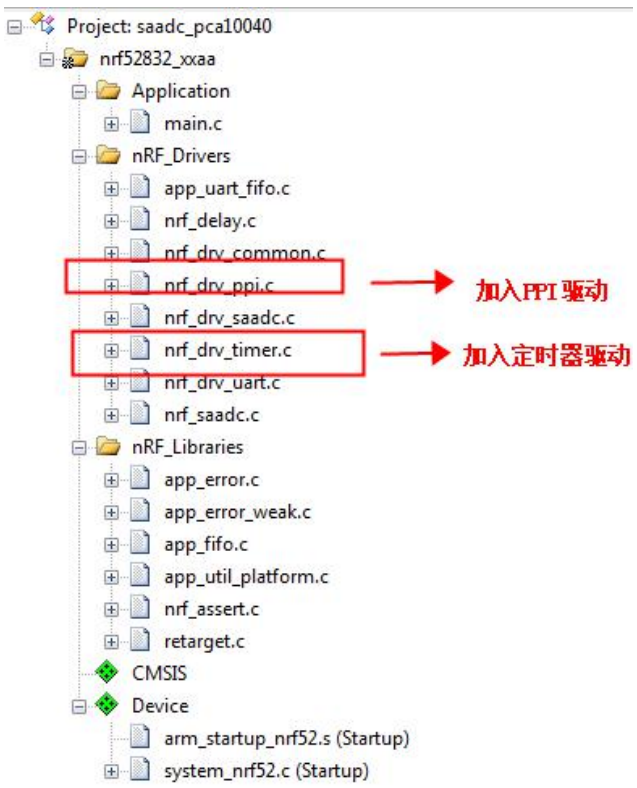
```
11.                 //延时 300ms，方便观察 SAADC 采样数据
12.                 nrf_delay_ms(300);
13.         }
14. }
```

打开串口助手，输出的实验现象如下，缓冲大小为 6，所以采样次数每次都会依次输出 6 个采样电压值，如下所示，而两个通道的采样值依次交错输出：

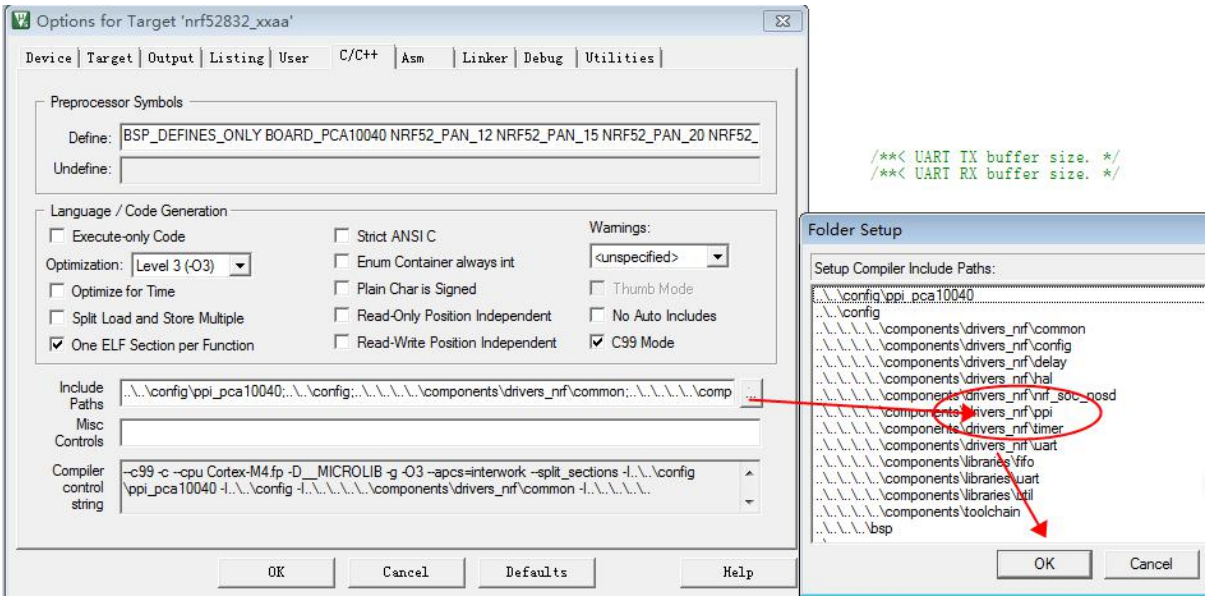
16.2.4 PPI 启动双缓冲中断采样

最后我们来讨论一个综合应用，结合了前面的定时器和 PPI 的功能，减少 CPU 的参与，提供转换效率的方法，也就是这讲的基于 PPI 启动的双缓冲中断采样法。我们前面的几种方法，都需要在主函数 main 中启动 adc 的采样，事件上是一种占用了 CPU 的，为了提搞我们的系统工作效率，我们把启动 adc 采样事件的这个工作丢给定时器和 PPI 通道来完成，CPU 不需要去参与。同时我们还采样官方推荐的双缓冲方式来存储与输出采样结果。

作为一个综合应用，我们用到了前面讲的 PPI 和定时器的知识，大家需要提前温习下前面的内容，首先是工程目录树如下：



图中，需要在之前的工程目录中添加 PPI 的驱动文件和定时器的驱动文件，然后再 main 主函数的头文件中调用#include "nrf_drv_ppi.h"和#include "nrf_drv_timer.h"，然后再路径中添加这两个启动的路径：



完成上面的工作后，我们来配置代码。配置代码要实现几个功能：

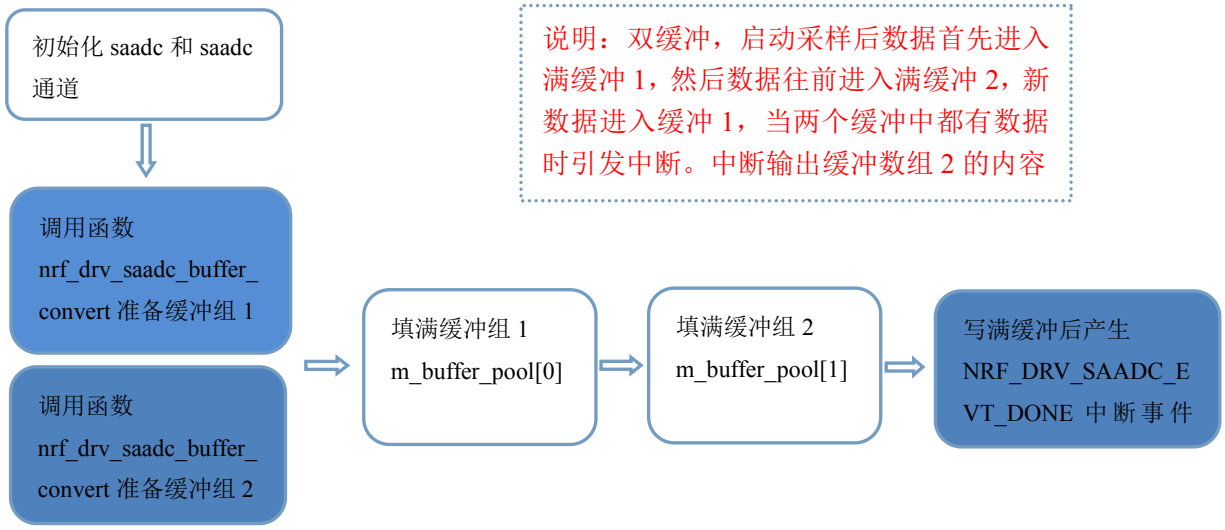
1：adc 的初始化，配置好 adc 的通道，缓冲大小和缓冲个数。

2：启动 adc 采样，本讲的核心是把启动工作丢给 PPI 和定时器处理，所以这里好配置定时器定时事件和 PPI 的触发通道。

3：最后就是 adc 采样完成后触发中断，中断中输出采集的数据。

下面就按照这 3 步来配置代码。

首先谈一下双缓冲 adc 的设置，双缓冲是在前面单缓冲的基础上实现的一个工作机制，也就是把缓冲的数组变成两个，数据依次进入缓冲数组 1 和缓冲数组 2，当两个数组内都有数据就会触发中断事件发生，中断后输出缓冲数组 2 内的内容：

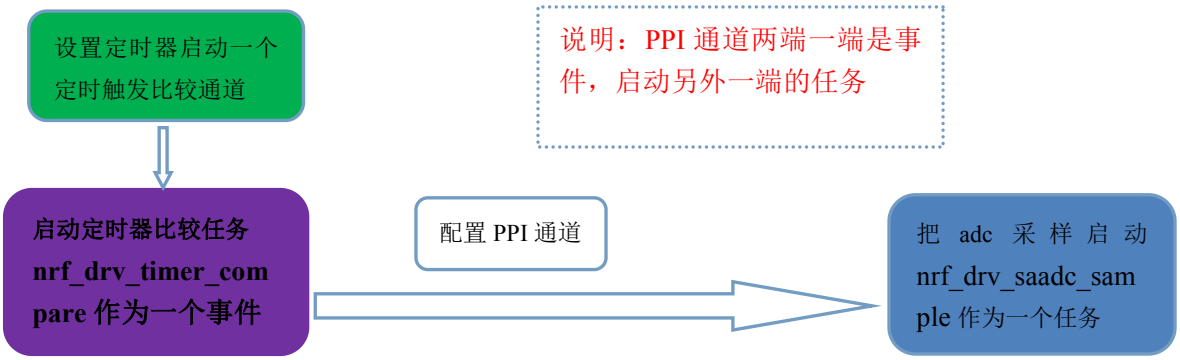


那么代码配置和单缓冲的区别如下所示，多一个缓冲数组配置：

```
15. void saadc_init(void)
16. {
17.     ret_code_t err_code;
18.     nrf_saadc_channel_config_t channel_config =
19.      //配置通道参数
    NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE(NRF_SAADC_INPUT_AIN2);
20.     //adc 初始化
21.     err_code = nrf_drv_saadc_init(NULL, saadc_callback);
22.     APP_ERROR_CHECK(err_code);
23.     //带入通道配置参数
24.     err_code = nrf_drv_saadc_channel_init(0, &channel_config);
25.     APP_ERROR_CHECK(err_code);
26.     //配置第一个缓冲
27.     err_code = nrf_drv_saadc_buffer_convert(m_buffer_pool[0],SAMPLES_IN_BUFFER);
28.     APP_ERROR_CHECK(err_code);
29.     //配置第二个缓冲
30.     err_code = nrf_drv_saadc_buffer_convert(m_buffer_pool[1],SAMPLES_IN_BUFFER);
31.     APP_ERROR_CHECK(err_code);
32. }
33.
```

再谈下第二个问题，adc 的采样启动，如何通过 PPI 启动 adc 的采样。前面的学习我们知道 PPI 实际上是一个通道，通道不能启动任何外设，打个比方：比如你在马路上，并不能自动的走，马路

只是一个通道，要车或者步行才能动。PPI 一样的是这个功能，我们要通过其他的外设来启动 adc 采集，一般采用定时器定时来启动 adc，原理如下图所示：



其基本原理还是比较简单的，我们首先配置一个定时器，定一个时间，比如 400ms，触发一次定时器比较发生。然后设置一个 PPI 通道，把定时器比较这个作为一个事件，作为 PPI 通道的一端。另外一端把启动 adc 的采样作为任务，作为 PPI 的另外一端。当 400ms 后，会通过定时器比较事情启动 adc 的采样开始。代码具体配置如下

```
34. //使能 PPI 通道
35. void saadc_sampling_event_enable(void)
36. {
37.     ret_code_t err_code = nrf_drv_ppi_channel_enable(m_ppi_channel);
38.     APP_ERROR_CHECK(err_code);
39. }
40.
41. void saadc_sampling_event_init(void)
42. {
43.     ret_code_t err_code;
44.     err_code = nrf_drv_ppi_init();
45.     APP_ERROR_CHECK(err_code);
46.     //定时器初始化
47.     err_code = nrf_drv_timer_init(&m_timer, NULL, timer_handler);
48.     APP_ERROR_CHECK(err_code);
49.
50.     //设置 每 400ms 发送一次 m_timer 比较事件
51.     uint32_t ticks = nrf_drv_timer_ms_to_ticks(&m_timer, 400);
52.   //设置定时 ，捕获/比较通道，比较值，清除比较任务，关掉比较器中断
53.     nrf_drv_timer_extended_compare(&m_timer, NRF_TIMER_CC_CHANNEL0, ticks,
    NRF_TIMER_SHORT_COMPARE0_CLEAR_MASK, false);
54.     nrf_drv_timer_enable(&m_timer);
55.
56.    //是设置 PPI 两端的通道，一个作为任务，一个作为事件
57.     uint32_t timer_compare_event_addr = nrf_drv_timer_compare_event_address_get(&m_timer,
    NRF_TIMER_CC_CHANNEL0);
58.     uint32_t saadc_sample_event_addr = nrf_drv_saadc_sample_task_get();
```

```
59.         //分频一个 PPI 通道
60.         err_code = nrf_drv_ppi_channel_alloc(&m_ppi_channel);
61.         APP_ERROR_CHECK(err_code);
62.         //分频 PPI 通道地址，一端是比较事件，一端是 adc 采样事件
63.         err_code = nrf_drv_ppi_channel_assign(m_ppi_channel, timer_compare_event_addr,
    saadc_sample_event_addr);
64.         APP_ERROR_CHECK(err_code);
65. }
```

第三步就是 adc 中断触发后进行数据输出，这个和前面的 adc 中断采集的内容一致，没有做任何变化，代码如下：

```
66. //adc 中断输出
67. void saadc_callback(nrf_drv_saadc_evt_t const * p_event)
68. {           float    val;
69.         if (p_event->type == NRF_DRV_SAADC_EVT_DONE)
70.         {
71.             ret_code_t err_code;
72.          //设置好缓存，为下次转换预备缓冲
73.             err_code = nrf_drv_saadc_buffer_convert(p_event->data.done.p_buffer,
    SAMPLES_IN_BUFFER);
74.             APP_ERROR_CHECK(err_code);
75.
76.             int i;
77.              //打印输出采样次数
78.             printf("ADC event number: %d\r\n",(int)m_adc_evt_counter);
79.              //输出采样值
80.             for (i = 0; i < SAMPLES_IN_BUFFER; i++)
81.             {
82. //                  printf("%d\r\n", p_event->data.done.p_buffer[i]);
83.                                 val = p_event->data.done.p_buffer[i] * 3.6 /1024;
84.                   printf("Voltage = %.3fV\r\n", val);
85.             }
86.             m_adc_evt_counter++;
87.         }
88. }
```

在主函数中，CPU 得到了解放，可以不做任何操作，初始化完成后直接等待结果输出，这种方式比较适合移植到协议栈下，代码如下：

```
89. int main(void)
90. {
91.         uart_config();
92.
93.         printf("\n\rSAADC HAL simple example.\r\n");
```

```
94.         saadc_sampling_event_init();
95.   //adc 初始化
96.         saadc_init();
97.         saadc_sampling_event_enable();
98.
99.         while(1)
100.         {
101.
102.         }
103. }
```

打开串口助手，输出的实验现象如下，缓冲大小为 5，如下所示，每次采样输出 5 个采样值：