第十章：内部温度传感与随机数产生

10.1 内部温度传感

在 nrf52xx 系列芯片内部，包含一个内部温度传感器。在一些恶劣的应用环境下面，可以通过检测芯片内部温度而感知设备的工作环境温度，如果温度过高或者过低了则马上睡眠或者停止运转. 可以保证您的设备工作的可靠性。

内部温度传感器的主要功能是测量芯片温度，但是如果外部应用需要测量温度也可以使用。因为芯片外接负载一定的情况下，那么芯片的发热也基本稳定，相对于外界的温度而言，这个偏差值也是基本稳定的，这时可以通过线性补偿来实现应用。

这里列出的是温度传感器的主要功能：

- 温度范围大于或等于设备的工作温度
- 分辨率为 0.25 度

通过触发 START 任务启动 TEMP 温度传感器。温度测量完成后，将生成 DATARDY 事件，并可从 TEMP 寄存器读取测量结果。要达到电气规范中规定的测量精度，必须选择外部晶体振荡器作为 HFCLK 源。

温度测量完成后，TEMP 这部分模拟电子设备掉电以节省电力。TEMP 仅支持一次性操作，这意味着必须使用 START 任务显式启动每个 TEMP 测量。

10.1.1 温度传感器寄存器

内部温度传感器作为外部设备，基础地址为 0x4000C000 ，其寄存器描述如下所示：

寄存器名称	地址偏移	功能描述
TASKS_START	0x000	开始测量温度
TASKS_STOP	0x004	停止温度测量
EVENTS_DATARDY	0x100	温度测量完成，数据准备就绪
INTENSET	0x304	启用中断
INTENCLR	0x308	禁用中断
TEMP	0x508	温度单位为°C（0.25°步长）
A0	0x520	第 1 块斜率线性函数的斜率
A1	0x524	第 2 块斜率线性函数的斜率
A2	0x528	第 3 块斜率线性函数的斜率
A3	0x52C	第 4 块斜率线性函数的斜率
A4	0x530	第 5 块斜率线性函数的斜率
A5	0x540	第 6 块斜率线性函数的斜率
B0	0x540	第 1 段线性函数的 y-截距
B1	0x544	第 2 段线性函数的 y-截距
B2	0x548	第 3 段线性函数的 y-截距
B3	0x54C	第 4 段线性函数的 y-截距
B4	0x550	第 5 段线性函数的 y-截距
B5	0x554	第 6 段线性函数的 y-截距
T0	0x560	第 1 段线性函数的终点
T1	0x564	第 2 段线性函数的终点
T2	0x568	第 3 段线性函数的终点
T3	0x56c	第 4 段线性函数的终点
T4	0x570	第 5 段线性函数的终点

寄存器说明：

1：INTENSET 寄存器：写 '1' 去使能 DATARDY 事件的中断

位数	Field	Value ID	Value	描述
第 0 位	DATARDY	Set	1	写：使能中断
		Event	0	读：判断中断已关闭
		Task	1	读：判断中断已使能

2：**INTENCLR** 寄存器：写 '1' 去禁止 DATARDY 事件的中断

位数	Field	Value ID	Value	描述
第 0 位	DATARDY	Set	1	写：关闭中断
		Event	0	读：判断中断已关闭
		Task	1	读：判断中断已使能

3：**TEMP** 寄存器：温度单位:°C(0.25°一个进位)

位数	Field	Value ID	Value	描述
第 0~31 位	TEMP			温度单位:°C(0.25°一个进位)

				测温结果。2 的补码格式，0.25° C 一个进位级 判决点：DATARDY 域

4：**A[n] n=0~5** 寄存器：线性函数的斜率

位数	Field	Value ID	Value	描述
第 0~11 位	TEMP			第 n+1 段线性函数的斜率

5：**B[n] n=0~5** 寄存器：段线性函数的 y 轴截距

位数	Field	Value ID	Value	描述
第 0~13 位	TEMP			第 n+1 段线性函数的 y 轴截距

6：**T[n] n=0~4** 寄存器：分段线性函数的端点

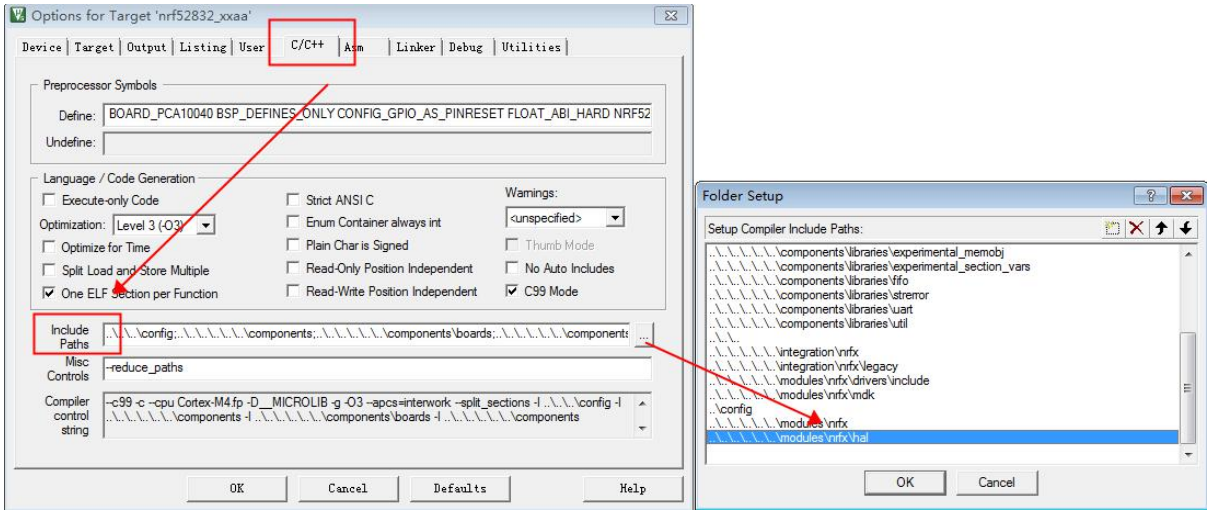
位数	Field	Value ID	Value	描述
第 0~7 位	TEMP			第 n+1 个分段线性函数的端点

10.1.2 温度传感器电气特征

功能描述	最小值	典型值	最大值	单位
温度测量所需的时间		**36**		**us**
温度传感器范围	**-40**		**85**	**° C**
温度传感器精度	**-5**		**5**	**° C**
温度传感器分辨率		**0.25**		**° C**
样品在恒定器件温度下的稳定性		**+/-0.25**		**° C**
在 25° C 的样品偏移	**-2.5**		**2.5**	**° C**

10.1.3 温度传感器库函数编程

对应温读采集的工程搭建，我们可以采用前面的串口组件库的工程例子进行修改。由于组件库中提供了一个 nrf_temp.h 驱动文件，因此我们需要关注在工程配置 Options for Target 中的 C/C++选项卡下：Include Paths 中添加\\\modules\nrfx\hal 路径，因为 nrf_temp.h 驱动文件位于 modules\nrfx\hal 文件夹内，具体如下图所示：



主函数中，需要在 main.c 文件最开头的文件包含中，包含添加#include "nrf_temp.h"已包含内部温度库头文件，如下图所示：

```
11  #include <stdbool.h>
12  #include <stdint.h>
13  #include <stdio.h>
14  #include "app_uart.h"
15  #include "app_error.h"
16  #include "nrf_delay.h"
17  #include "nrf.h"
18  #include "bsp.h"
19  #if defined (UART_PRESENT)
20  #include "nrf_uart.h"
21  #endif
22  #if defined (UARTE_PRESENT)
23  #include "nrf_uarte.h"
24  #endif
25  #include "nrf_temp.h"//添加内部温度库头文件
26
```

本例中，我们主要来实现了测试内部温度，然后通过串口输出这样一个过程。对应内部温度采样配置总结如下：

1：初始化温度采集，官方在 nrf_temp.h 文件中提供了一个初始化函数：

nrf_temp_init();

函数：static __INLINE void nrf_temp_init(void)
*功能：功能为准备温度测量的临时模块。此函数初始化临时模块并将其写入隐藏的配置寄存器。
*参数　无

温度偏移值必须手动加载到临时模块，可以采用该函数实现。

2：启动温度采集，等待采集过程完成。温湿度采集提供了寄存器 TASKS_START 来启动采集，当置位该寄存器，就可以启动内部温度采集模块进行温度采集了。

NRF_TEMP->TASKS_START = 1;

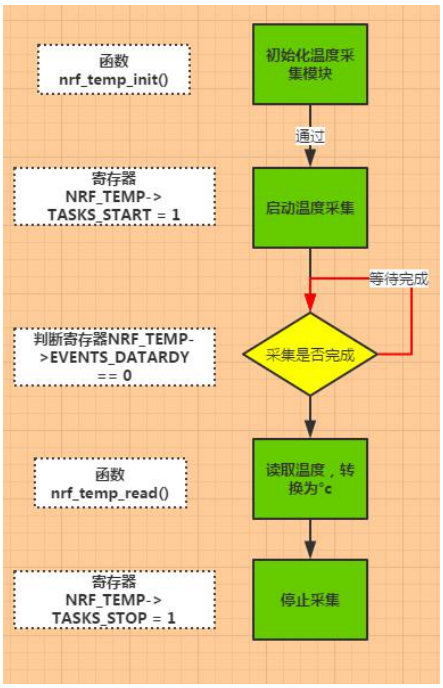
采集需要时间，这个过程并不能立即打断，因此需要等待 EVENTS_DATARDY 事件是否被置为，如果置位了，表明临时缓冲中温度数据已经采集存储就绪。或者在 INTENSET 寄存器中使能 DATARDY 事件的中断，采用中断方式进行判断。

3：读出采集温度的值，并且停止温度采集。当临时缓冲中温度数据已经准备就绪后，就可以读出数据，官方在 nrf_temp.h 文件中提供了一个读取函数:

nrf_temp_read();

函数：static __INLINE int32_t nrf_temp_read(void)
*功能：为温度测量读数功能。该函数读取 10 位 2 的补码值并将其转换为 32 位 2 的补码值。
*参数 无

该函数从 TEMP 寄存器中读取 10 位 2 的补码值并将其转换为 32 位 2 的补码值，注意 TEMP 寄存器中存储的值是以 0.25° C 一个进位级，如果要转换为° C 为单位，则实际的温度值应该是*0.25 或者除以 4。 当温度读取成功后，对寄存器 TASKS_STOP 置位以停止温度采集。



根据总结的设计步骤，主函数编写程序如下所示。程序中采用查询方式等待温度采集成功，成功后读取温度值，并且转换为° C 为单位，通过串口打印输出，进行观察：

```
01. int main(void)
02. {
03.
04.     int32_t volatile temp;
05.   //初始化 LED 灯
06.     LEDS_CONFIGURE(LEDS_MASK);
07.     LEDS_OFF(LEDS_MASK);
08.   //初始化串口
09.   uart_init();
10.   //初始化内部温度传感器
11.     nrf_temp_init();
12.     printf("Temperature example started.");
13.     //循环测试内部温度
14.     while (true)
15.     {   //开始温度测量
16.         NRF_TEMP->TASKS_START = 1;
17.         /*当温度测量尚未完成时，如果为 DATARDY 事件启用中断并在中断中读取结果，则
18. 可以跳过等待。 */
19.         while (NRF_TEMP->EVENTS_DATARDY == 0)
```

```
20.             {
21.                 //不做任何事
22.             }
23.             NRF_TEMP->EVENTS_DATARDY = 0;
24.
25.             //停止任务清除临时寄存器。
26.             temp = (nrf_temp_read() / 4);
27.
28.             /*TEMP:当 DATARDY 事件发生时，Temp 模块模拟前端不断电。*/
29.             NRF_TEMP->TASKS_STOP = 1; /**  停止温度测量。  */
30.
31.             printf("温度采样: %u\n\r", (uint8_t)(temp));
32.            nrf_gpio_pin_toggle(LED_1);
33.             nrf_delay_ms(500);
34.
35.     }
```

工程编译后下载到青风 nrf52832 开发板内。打开串口调试助手，选择开发板串口端号，设置波特率为 115200，数据位为 8，停止位为 1。每隔 500ms 会输出一次温度值，如果串口正确收到温度的数据，则 LED1 灯也会翻转。串口输入现象如下图所示：

10.2 随机数产生器

10.2.1 随机数发生器原理

随机数发生器（RNG）基于内部热噪声产生的真非确定性的随机数。经常用于通讯加密等场合。

该随机数发生器 RNG 通过触发 START 任务进行启动；通过触发 STOP 任务停止运行。在启动时，新的随机数连续产生，并在准备好时写入 VALUE 寄存器中。随着每次新的随机数写入到 VALUE 寄存器，都会触发一个 VALRDY 事件。这意味着生成 VALRDY 事件后，直到下一个 VALRDY 事件之前这段时间，CPU 有足够的时间在新的随机数覆盖旧的随机数之前，从 VALUE 寄

存器中读出的旧随机数的值。

10.2.1.1 随机数发生器的应用场合

随机数发生器常用于处理器和设备之间的通讯中进行信息加密和身份认证，其应用场合如下：

1、某设备发送操作请求给 MCU，MCU 利用随机数发生器产生一段随机序列（明文），MCU 采用加密方式对明文进行加密产生一段密文（明文 + 用户密码 = 密文）。然后 MCU 发送密文给某设备。

2、某设备利用用户密码，同时采用相同的解密方式对密文进行解密，还原出明文，并发送给 MCU。MCU 在 100ms 内等待设备端解密出来的明文，并与随机数发生器随机产生的明文做比较。若比较正确则认为设备端输入了正确的密码，如果错误则终止通讯。

整个通信过程都中，明文都是随机序列。如果设备端无正确的密码，通讯过程是无法正确进行的。因此实现了安全的密码验证授权！

10.2.2.2 偏差校正

在内部比特流上采用偏差校正算法以消除对“1”或“0”的任何偏差。然后将这些位排队到一个 8 位寄存器，以便从 VALUE 寄存器进行并行读出。可以在 CONFIG 寄存器中启用偏差校正。这将导致较慢的值生成，但将确保随机值的统计均匀分布。

生成速度：生成一个随机字节数据所需的时间是不可预测的，并且可能在一个字节到下一个字节之间变化。启用偏差校正时尤其如此。

10.2.2 随机数发生器寄存器

内部温度传感器作为外部设备，基础地址为 0x4000D000 ，其寄存器描述如下所示：

寄存器名称	地址偏移	功能描述
TASKS_START	0x000	任务启动随机数生成器
TASKS_STOP	0x004	任务停止随机数生成器
EVENTS_VALRDY	0x100	为写入 VALUE 寄存器的每个新随机数生成事件
SHORTS	0x200	快捷方式登记
INTENSET	0x304	启用中断
INTENCLR	0x308	禁用中断
CONFIG	0x504	配置寄存器
VALUE	0x508	输出随机数

1： SHORTS 快捷寄存器：用于快速通过 VALRDY 事件触发 STOP 任务：

位数	复位值	Field	Value ID	Value	描述：可读可写寄存器
第 0 位	0x00000000	VALRDY_STOP			VALRDY 事件和 STOP 任务之间的快捷方式
			Disabled	0	禁用快捷方式
			Enabled	1	启用快捷方式

2： INTENSET 启用中断寄存器：使能随机数中断

位数	复位值	Field	Value ID	Value	描述：可读可写寄存器
第 0 位	0x00000000	VALRDY			将写入'1'为启用 VALRDY 事件中断
			Set	1	使能中断
			Disabled	0	读：中断禁止
			Enabled	1	读：中断使能

3：INTENCLR 关闭中断寄存器：禁止随机数中断

位数	复位值	Field	Value ID	Value	描述：可读可写寄存器
第 0 位	0x00000000	VALRDY			将写入'1'为禁止 VALRDY 事件中断
			Clear	1	禁止中断
			Disabled	0	读：中断禁止
			Enabled	1	读：中断使能

4：CONFIG 配置寄存器：配置是否开启偏差校正

位数	复位值	Field	Value ID	Value	描述：可读可写寄存器
第 0 位	0x00000000	DERCEN			偏差校正，
			Disabled	0	关闭校正
			Enabled	1	使能校正

5：VALUE 输出随机数寄存器：随机数的存储寄存器

位数	复位值	Field	Value ID	Value	描述：可读可写寄存器
第 0~7 位	0x00000000	VALUE		[0..255]	生成随机数的值

10.2.3 随机数发生库函数编程

10.2.3.1 组件库的编写流程

随机数发生器采用组件库编程会大大降低编程难度，优化数据处理过程。下面详细讲解组件库编写随机数发生器的过程，并且进入到库函数内部，探讨整个随机数产生以及读取的过程。

1.使用 RNG 模块之前，首先初始化 RNG 发生器模块，在 SDK 的组件库中，提供了 nrf_drv_rng.c 这样一个库文件，该文件中提供了一个 nrf_drv_rng_init()函数，用于初始化 RNG 模块功能。

关于函数 nrf_drv_rng_init，该函数介绍如下所示：

函数：ret_code_t nrf_drv_rng_init(nrf_drv_rng_config_t const * p_config);
*功能：函数用于初始化 nrf_drv_rng 模块。
*参数 p_config 初始配置。
* 返回值： NRF_SUCCESS 驱动程序初始化成功。 * 返回值：NRF_ERROR_MODULE_ALREADY_INITIALIZED 驱动程序已经初始化。

该函数的形参为 nrf_drv_rng_config_t const * p_config，也就是提供了一个结构体对初始化化参数进行了定义，我们找到这个结构体：

```
01. typedef struct
02. {
```

```
    bool        error_correction : 1;   /*误差修正的帧。*/
    uint8_t    interrupt_priority;      /*中断优先级 */
} nrfx_rng_config_t;
```

这个结构体定义了配置的参数有两个，一个帧标注是否有数据修正，一个设置了优先级。nrf_drv_rng_init 函数的结构看起来来非常简单，但是我们应该关注其函数内部到底做了什么工作？下面我们进入函数内部，具体代码如下所示：

```
01. ret_code_t nrf_drv_rng_init(nrf_drv_rng_config_t const * p_config)
02. {
03.     ret_code_t err_code = NRF_SUCCESS;
04.     if (m_rng_cb.state != NRFX_DRV_STATE_UNINITIALIZED)//RNC 状态，如果不等于未初始化
05.     {
06.         return NRF_ERROR_MODULE_ALREADY_INITIALIZED;//返回已经初始化模块
07.     }
08.     if (p_config == NULL)
09.     {
10.         p_config = &m_default_config;   //如果是 NULL，则使用默认配置
11.     }
12.     m_rng_cb.config = *p_config;//如果有配置，则复制给配置参数
13.
14.     NRF_DRV_RNG_LOCK();//RNC 锁定，主要是协议栈下
15.
16.     if (!NRF_DRV_RNG_SD_IS_ENABLED())
17.     {
18.         err_code = nrfx_rng_init(&m_rng_cb.config, nrfx_rng_handler);//初始化 RNC 配置，并且触发 RNC 事件
19.         if (err_code != NRF_SUCCESS)
20.         {
21.             return err_code;
22.         }
23.         nrfx_rng_start();//开始 RNC
24.     }
25.     m_rng_cb.state = NRFX_DRV_STATE_INITIALIZED;
26.     NRF_DRV_RNG_RELEASE();//RNC 释放
27.     return err_code;
28. }
```

第 8~10 行：如果配置参数为 null，则配置使用默认配置 &m_default_config，我们可以找到默认的定义；

```
29. #define NRFX_RNG_DEFAULT_CONFIG                                 \
30.     {                                                           \
31.         .error_correction     = NRFX_RNG_CONFIG_ERROR_CORRECTION, \
32.         .interrupt_priority = NRFX_RNG_CONFIG_IRQ_PRIORITY,     \
33.     }
```

默认配置参数在文件 sdk_config.h 文件中，具体配置如下所示：

```
34. // 错误修正，是否启动错误修正，启动就设置为 1
```

```
35. #ifndef NRFX_RNG_CONFIG_ERROR_CORRECTION
36. #define NRFX_RNG_CONFIG_ERROR_CORRECTION 1
37. #endif
38.
39. // RNG 的中断优先级设置
40. // <0=> 0 (highest)
41. // <1=> 1
42. // <2=> 2
43. // <3=> 3
44. // <4=> 4
45. // <5=> 5
46. // <6=> 6
47. // <7=> 7
48. #ifndef NRFX_RNG_CONFIG_IRQ_PRIORITY
49. #define NRFX_RNG_CONFIG_IRQ_PRIORITY 6
50. #endif
```

第 18 行，调用函数 nrfx_rng_init 初始化 RNC 配置，初始化配置中的两个参数。如下图所示。首先开球错误修正功能。然后配置 RNC 中断优先级，开启 RNC 中断。这个中断执行后，会触发一个 nrfx_rng_evt_handler 的回调事件。在 18 行中把这个事件定义为 nrfx_rng_handler。

```
60  nrfx_err_t nrfx_rng_init(nrfx_rng_config_t const * p_config, nrfx_rng_evt_handler_t handler)
61  {
62      NRFX_ASSERT(p_config);
63      NRFX_ASSERT(handler);
64      if (m_rng_state != NRFX_DRV_STATE_UNINITIALIZED)
65      {
66          return NRFX_ERROR_ALREADY_INITIALIZED;
67      }
68
69      m_rng_hndl = handler;
70
71      if (p_config->error_correction)
72      {
73          nrf_rng_error_correction_enable();
74      }
75      nrf_rng_shorts_disable(NRF_RNG_SHORT_VALRDY_STOP_MASK);
76      NRFX_IRQ_PRIORITY_SET(RNG_IRQn, p_config->interrupt_priority);//优先级设置
77      NRFX_IRQ_ENABLE(RNG_IRQn);//使能RNC中断
78
79      m_rng_state = NRFX_DRV_STATE_INITIALIZED;
80
81      return NRFX_SUCCESS;
82  }
```

开启错误修正

设置中断优先级，并且开启中断

第 23 行：调用函数 nrfx_rng_start()，开始 RNC 随机数发生器运行。一旦开始运行，新的随机数连续产生，并在准备好时写入 VALUE 寄存器中。随着每次新的随机数写入到 VALUE 寄存器，都会触发一个 VALRDY 中断事件。中断函数在 nrfx_rng.c 进行了定义，前面已经使能了中断，当中断事件发生后，我们进入中断。中断中首先清除 VALRDY 事件标志，然后通过 nrf_rng_random_value_get 函数读出 VALUE 寄存器中的值，最后就去执行 nrfx_rng_evt_handler 的回调事件。

```
01.     void nrfx_rng_irq_handler(void)
02. {
03.     nrf_rng_event_clear(NRF_RNG_EVENT_VALRDY);//清除中断标志
04.     uint8_t rng_value = nrf_rng_random_value_get();//读出 VALUE 寄存器中的值
05.     m_rng_hndl(rng_value);//执行 nrfx_rng_evt_handler 的回调事件。
06.     NRFX_LOG_DEBUG("Event: NRF_RNG_EVENT_VALRDY.");
07. }
```

rfx_rng_evt_handler 的回调事件主要就是需要把从 VALUE 寄存器中读出的值放入到队列文件中声明的一个专用放置随机数的池 m_rand_pool 中，如果这个池防满了，避免溢出，则停止 RNC 模块的运行，具体代码如下所示：

```
01. static void nrfx_rng_handler(uint8_t rng_val)
02. {
03.     NRF_DRV_RNG_LOCK();
04.     if (!NRF_DRV_RNG_SD_IS_ENABLED())
05.     {     //把从 VALUE 寄存器中读出的值放入到 m_rand_pool 池中
06.         UNUSED_RETURN_VALUE(nrf_queue_push(&m_rand_pool, &rng_val));
07.
08.         if (nrf_queue_is_full(&m_rand_pool))
09.         {   //如果队列池已经满了，则停止 RNG 模块运行
10.             nrfx_rng_stop();
11.         }
12.         NRF_LOG_DEBUG("Event: NRF_RNG_EVENT_VALRDY.");
13.     }
14.     NRF_DRV_RNG_RELEASE();
15. }
```

执行完回调后，在 m_rand_pool 池中，就有 RNC 模块参数的随机数了。

注意事项：

关于这个 m_rand_pool 池的说明：

在 SDK 中提供了一个 nrf_queue.c 文件，专门来提供类似堆栈的缓冲区，相当一个存储池。随机数这个工程中需要调用这个函数，并且在 nrf_drv_rng.c 库文件中，调用了这个队列的定义，声明了一个 m_rand_pool 池，代码如下所示：

NRF_QUEUE_DEF(uint8_t,m_rand_pool,RNG_CONFIG_POOL_SIZE,NRF_QUEUE_MODE_OVERFLOW);

两个参数，一个是池子的大小，单位为字节数，最大可以声明为 64 字节：

```
01. #define RNG_CONFIG_POOL_SIZE 64
```

一个参数就是队列的模式：溢出模式或者覆盖模式：

```
02. typedef enum
03. {
04.     NRF_QUEUE_MODE_OVERFLOW,          //如果队列已满，新数据将覆盖老的数据。
05.     NRF_QUEUE_MODE_NO_OVERFLOW,       //如果队列已满，则不接受新数据。
06. } nrf_queue_mode_t;
```

默认配置为覆盖模式，新的数据过来就会覆盖老的数据。因此前面我们会判断 pool 池子是否满了，如果满了就停止 RNG 产生新的随机数，避免数据被覆盖。

2. 第二步从 pool 池中读出对应字节的随机数，并且把读出的随机数放入一个缓冲向量中，然后通过串口在 PC 机上打印进行观察。这时候我们可以调用两个库函数 nrf_drv_rng_bytes_available 和 nrf_drv_rng_rand，这两个函数介绍如下：

函数 nrf_drv_rng_bytes_available：

函数：void nrf_drv_rng_bytes_available(uint8_t * p_bytes_available);
*功能：用于获取当前可用的随机字节数。

*参数 p_bytes_available 当前随机数存储池中可用的字节数。

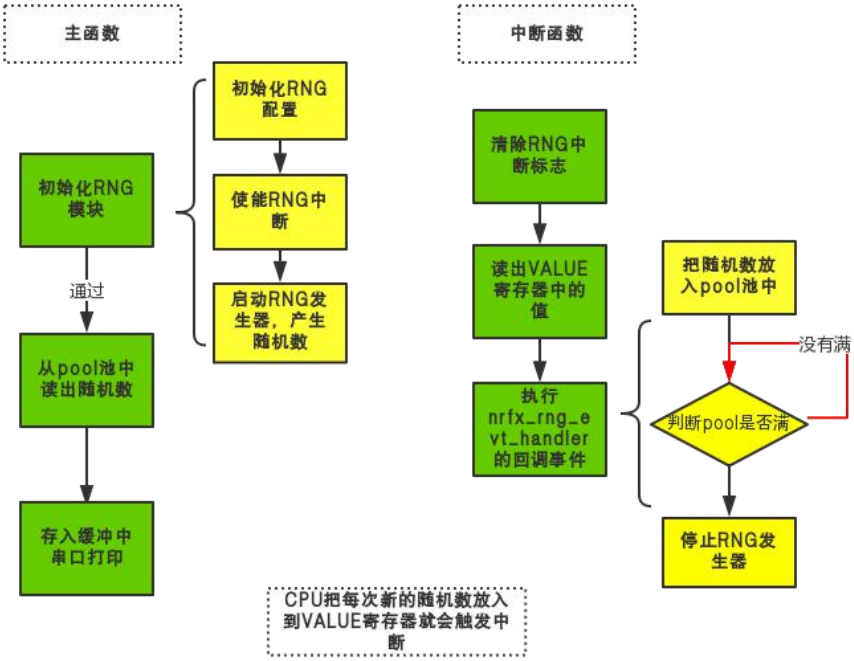
函数 nrf_drv_rng_rand：

函数：ret_code_t nrf_drv_rng_rand(uint8_t * p_buff, uint8_t length);
*功能：用于把随机数放入对应的缓冲向量中。
* 参数 p_buff 指向存储字节的 uint8_t 缓冲区的指针。 * 参数 length 从随机数存储池中取出并放入 p_buff 中的字节数。
* 返回值： NRF_SUCCESS 如果请求的字节被写入 p_buff。 * 返回值： NRF_ERROR_NOT_FOUND 如果没有向缓冲区写入字节，因为池中没有足够的可用字节。

编写代码如下所示，首先获取 pool 池中随机数的大小，比较下我们设置的 buff 缓冲的大小，选择最小的值作为我们存入 buff 的长度，以避免数据丢失。

```
01.  //获取随机数发生器 pool 池中产生的随机数字节大小
02.    nrf_drv_rng_bytes_available(&available);
03.    //比较设置的存放空间和随机数发送池中随机数的大小，取最小的作为写入长度
04.    uint8_t length = MIN(size, available);
05.    //把对应长度的随机数写入到缓冲中
06.    err_code = nrf_drv_rng_rand(p_buff, length);
```

那么整个随机数的获取，以及随机数读出过程。可以总结如下图所示的流程。主程序和中断程序相互独立。如果使能了 RNG 中断，当然 RNG 启动后，CPU 就会把新产生的随机数放入到 VALUE 寄存器中就会触发对应中断，在中断中完成把 VALUE 寄存器中的随机数放入到 pool 队列池的过程。当 pool 队列池满了后，会停止 RNG 模块，退出中断。主函数则是以循环扫描的方式读取 pool 队列池中的随机数，放入到缓冲数组中，通过串口打印输出进行观察。



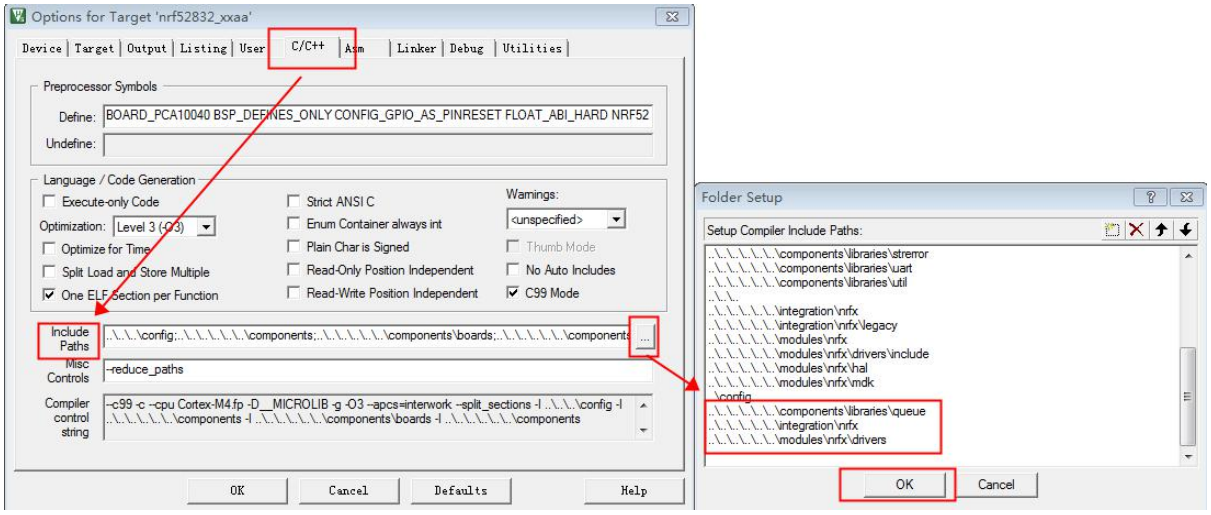
10.2.3.2 RNG 工程的搭建

我们先来搭建 RNC 工程目录，随机数发生器的演示工程可以在串口例程上进行修改，如下图所示。需要添加是三个驱动库函数，分别为：nrfx_rng.c、nrf_drv_rng.c 和 nrf_queue.c 函数。

添加驱动文件库后，这些文件库的路径也需要进行链接。在工程配置 Options for Target 中的 C/C++选项卡下：Include Paths 中添加：

\components\libraries\queue 路径	nrf_queue.c 文件的路径
\integration\nrfx 路径	nrf_drv_rng.c 文件的路径
\modules\nrfx\drivers 路径	nrfx_rng.c 文件的路径

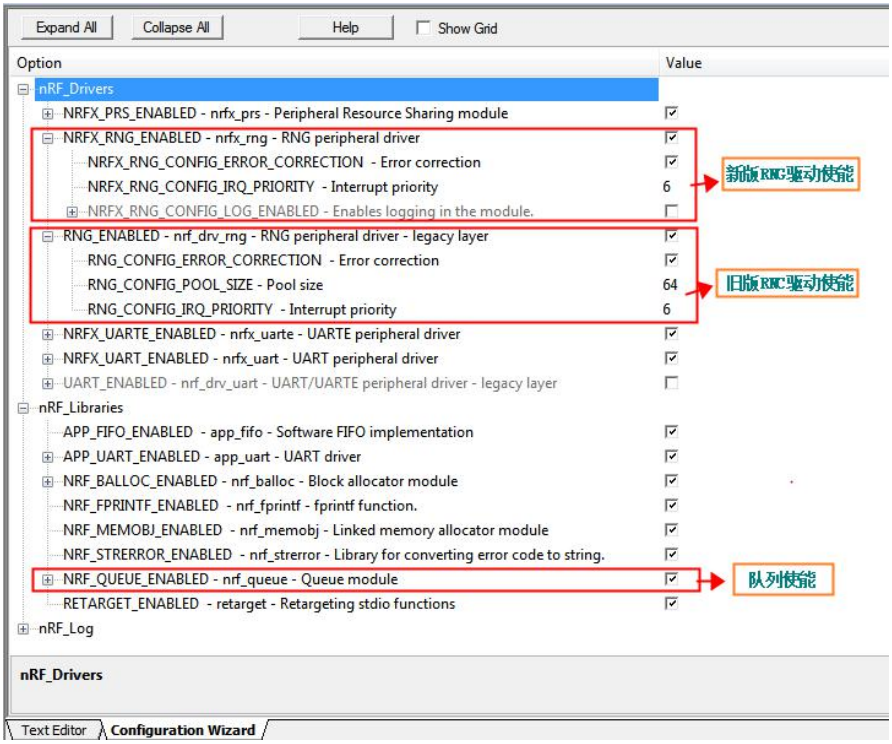
具体如下图所示：



由于 nrf_queue.h 的头文件是在 nrf_drv_rng.c 文件中被引用、nrfx_rng.h 的头文件在 nrf_drv_rng.h 文件中被引用。所以主函数中只需要引用一个头文件 nrf_drv_rng.h 就可以了对这三个函数进行调用。

```
07.  #include "nrf_drv_rng.h"
```

同时需要在 sdk_config.h 文件中，添加配置 RNG 相关的配置，注意串口的 sdk_config.h 文件中是没有 RNG 相关的配置的，需要自己手动添加的，具体添加内容请例程参考代码。如果添加成功，切换到配置导航选项卡 configuarton wizard 上，会出现对应配置被勾选，如下图所示的。



配置完成后，开始编写代码。首先在 main.c 文件中编写一个获取数据数放入缓冲向量的子函数，调用上面两个库函数，具体代码如下所示：

```
08.   #define   RANDOM_BUFF_SIZE      16        /*存放随机数的缓冲大小. */
09.
10.  /**  函数功能是用于获取随机数，把随机数放入缓冲向量。
11.    *
```

```
12.  * 参数  p_buff          指向 unit8_t 缓冲区的指针，用于存储随机数。
13.  * 参数  length          从 pool 池中取出并放入 p_buff 中的字节数。
14.  *
15.  * 返回值        实际放置在 p_buff 中的字节数。
16.  */
17. static uint8_t random_vector_generate(uint8_t * p_buff, uint8_t size)
18. {
19.     uint32_t err_code;
20.     uint8_t  available;
21.     //获取随机数发生器 pool 池中产生的随机数字节大小
22.     nrf_drv_rng_bytes_available(&available);
23.     //比较设置的存放空间和随机数发送池中随机数的大小，取最小的作为写入长度
24.     uint8_t length = MIN(size, available);
25.     //把对应长度的随机数写入到缓冲中
26.     err_code = nrf_drv_rng_rand(p_buff, length);
27.     APP_ERROR_CHECK(err_code);
28.
29.     return length;
30. }
```

主函数中，首先初始化串口，然后调用 RNG 初始化库函数，根据上面的分析，初始化库函数里包含了初始化 RNG 配置、使能 RNG 中断、开始 RNG 等功能，当随机数生成后，会产生 RNG 中断，中断中把产生的随机随机数放入到 pool 队列池中。

最后就调用 random_vector_generat 函数，配合串口打印功能，把 pool 队列池中的随机数放入到一个缓冲数组中，然后再 pc 机上的串口助手上打印出来进行观察。

```
01. /**
02.  *主函数：循环输出随机数
03.  */
04. int main(void)
05. {
06.     LEDS_CONFIGURE(LEDS_MASK);
07.     LEDS_OFF(LEDS_MASK);
08.     uint32_t err_code;
09.     const app_uart_comm_params_t comm_params =
10.       {
11.           RX_PIN_NUMBER,
12.           TX_PIN_NUMBER,
13.           RTS_PIN_NUMBER,
14.           CTS_PIN_NUMBER,
15.           APP_UART_FLOW_CONTROL_DISABLED,
16.           false,
17.           UART_BAUDRATE_BAUDRATE_Baud115200
18.       };
19.     //初始化串口
20.     APP_UART_FIFO_INIT(&comm_params,
```

```
                                UART_RX_BUF_SIZE,
                                UART_TX_BUF_SIZE,
                                uart_Interrupt_handle,
                                APP_IRQ_PRIORITY_LOW,
                                err_code);

    APP_ERROR_CHECK(err_code);
     //初始化 RNG 模块，并且打开 RNG 模块，使能中断
    err_code = nrf_drv_rng_init(NULL);
    APP_ERROR_CHECK(err_code);
    printf("RNG example started.");

    while (true)
    {    //配置一个缓冲，把 pool 中的随机数放入缓冲数组中
        uint8_t p_buff[RANDOM_BUFF_SIZE];
        uint8_t length = random_vector_generate(p_buff,RANDOM_BUFF_SIZE);
        printf("Random Vector:");
        //串口打印该数组
          for(uint8_t i = 0; i < length; i++)
        {
            printf(" %2x",(int)p_buff[i]);
        }
        printf("\n\r");
        nrf_gpio_pin_toggle(LED_1);
        nrf_delay_ms(1000);
    }
}
```

工程编译后下载到青风 nrf52832 开发板内。打开串口调试助手，选择开发板串口端号，设置波特率为 115200，数据位为 8，停止位为 1。每隔 500ms 会输出一次温度值，如果串口正确打印出随机数，则 LED1 灯也会翻转。串口输出入下图所示，输出 16 字节的随机数：