第十五章 看门狗的应用

15.1 原理分析

15.1.1：看门狗的作用

看门狗定时器（WDT，Watch Dog Timer）是单片机的一个组成部分，它实际上是一个计数器。看门狗计时器是用来防止万一单片机程序出错造成重大损失的计时器。

防错的原理很简单，它在硬件上就是一个定时器，当它溢出的时候就会让单片机强制复位使程序重新开始执行。正常的情况下是不能让它溢出的，所以在程序上每隔一段时间要给他置一次值（俗称喂狗），只要程序中正常给它喂他就不会溢出。。因此可以用看门狗防止程序在跑飞的时候回不到正常模式。

看门狗可用于受到电气噪音、电源故障、静电放电等影响的应用，或需要高可靠性的环境。如果一个应用不需要看门狗功能，可以配置看门狗定时器为一个间隔定时器，这样可以用于在选定的时间间隔产生中断。

15.1.2：看门狗的运行

1. 看门狗溢出

nRF52xx 系统芯片的看门狗上是向下计数的定时器。使用低频时钟源（LFCLK）提供时钟。通过触发 START 任务来启动看门狗定时器。可以在低功耗应用、休眠或者调试的时候，通过配置 CONFIG 寄存器来运行看门狗或者暂停看门狗。

当看门狗定时器通过 START 任务启动时，看门狗计数器加载 CRV 寄存器中指定的值。然后看门狗向下计数器，当在向下计数到 0 后，会溢出时产生 TIMEOUT 事件。看门狗 TIMEOUT 事件会导致系统重新复位或者 TIMEOUT 超时中断。如果需要喂狗重新加载计数初值，此计数器重新加载 CRV 寄存器中指定的值。

看门狗的 TIMEOUT 超时时间由下式给出：

timeout[s] =（CRV + 1）/ 32768

启动时，只要没有其他 32.768 kHz 时钟源正在运行并产生 32.768 kHz 系统时钟，看门狗就会自动强制启动 32.768 kHz RC 振荡器。

2. 看门狗喂狗

看门狗喂狗实际上就是看门狗定时器重加载的过程。看门狗有 8 个独立的重载请求寄存器 RR[0]~RR[7]，用于使用 CRV 寄存器中指定的值重新加载到看门狗然后计数。如果需要重新加载看门狗计数器，需要将特殊值 0x6E524635 写入所有使能的重载寄存器 RR[n]。通过 RREN 寄存器决定单独使能哪一个 RR 寄存器或同时使用多个 RR 寄存器。

3. 看门狗复位与 TIMEOUT 中断

如果系统没有按时的进行喂狗操作，那么 TIMEOUT 事件就会自动导致看门狗复位。

如果我们通过寄存器 INTENSET 使能了看门狗中断，那么看门狗配置会在 TIMEOUT 事件上产生中断。在产生 TIMEOUT 事件后，看门狗将推迟两个 32.768 kHz 时钟周期后执行看门狗复位。

必须在启动前配置看门狗。启动后，看门狗的配置寄存器（包括寄存器 CRV，RREN 和 CONFIG）将被阻止以进行进一步配置，但当器件复位或从关闭模式唤醒后再次开始运行，看门狗配置寄存器将再次可用于配置看门狗。看门狗可以从多个复位源复位，但并不是所有的复位都可以复位看门狗定时器，如下图所示，x 表示可以可以复位看门狗定时器：

Reset source	WDT
CPU lockup [6]	
Soft reset	
Wakeup from System OFF mode reset	
Watchdog reset [9]	x
Pin reset	x
Brownout reset	x
Power on reset	x

4 临时暂停看门狗

默认情况下，看门狗将在 CPU 处于休眠状态会保持运行以及调试器停止时暂停运行。但是可以通过设置寄存器 CONFIG，来使看门狗在 CPU 处于休眠状态时以及调试器停止时都进行自动暂停。配置寄存器 CONFIG 如下图所示：

Address offset: 0x50C

Configuration register

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				C 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 0 0 1**
Id	**RW**	**Field**	**Value Id**	**Value**	**Description**
A	RW	SLEEP 休眠模式下			Configure the watchdog to either be paused, or kept running, while the CPU is sleeping 当CPU休眠的时候暂停看门狗
			Pause 暂停	0	Pause watchdog while the CPU is sleeping
			Run 运行	1	Keep the watchdog running while the CPU is sleeping 当CPU休眠的时候保持看门狗运行
C	RW	HALT 调试模式			Configure the watchdog to either be paused, or kept running, while the CPU is halted by the debugger 当处于调试停止的时候暂停看门狗
			Pause 暂停	0	Pause watchdog while the CPU is halted by the debugger
			Run 运行	1	Keep the watchdog running while the CPU is halted by the debugger 当CPU处于调试停止的时候运行看门狗

15.2 应用实例编写

15.2.1 WDT 寄存器的编程

15.2.1.1 看门狗寄存器介绍：

看门狗作为外部设备具有 1 个模块，其基础地址为下表所示：

基础地址	外设	模块	描述
0x40010000	WDT	WDT	看门狗定时器

看门狗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	启动看门狗
EVENTS_TIMEOUT	0x100	看门狗超时
INTENSET	0x304	启用中断
INTENCLR	0x308	禁用中断
RUNSTATUS	0x400	运行状态
REQSTATUS	0x404	请求状态
CRV	0x504	计数器重载值
RREN	0x508	启用重载请求寄存器的寄存器
CONFIG	0x50C	配置寄存器
RR[0]	0x600	重新加载请求 0
RR[1]	0x604	重新加载请求 1
RR[2]	0x608	重新加载请求 2
RR[3]	0x60C	重新加载请求 3
RR[4]	0x610	重新加载请求 4
RR[5]	0x614	重新加载请求 5
RR[6]	0x618	重新加载请求 6
RR[7]	0x61C	重新加载请求 7

下面几个寄存器详细说明：

1：INTENSETTIMEOUT 事件中断使能寄存器：

位数	复位值	Field	Value ID	Value	描述：可读可写
第 0 位	0x00000000	TIMEOUT	Set	1	写：使能 TIMEOUT 事件中断
			Disabled	0	读：事件中断关闭状态
			Enabled	1	读：事件中断使能状态

INTENCLRTIMEOUT 事件中断禁止寄存器：

位数	复位值	Field	Value ID	Value	描述：可读可写
第 0 位	0x00000000	TIMEOUT	Set	1	写：禁止 TIMEOUT 事件中断
			Disabled	0	读：事件中断关闭状态
			Enabled	1	读：事件中断使能状态

2：RUNSTATUS 看门狗状态寄存器：

位数	复位值	Field	Value ID	Value	描述：只读寄存器

第 0 位	0x00000000	RUNSTATUS			指示看门狗是否运行，
			NotRunning	0	读：看门狗没有运行
			Running	1	读：看门狗运行中

3：REQSTATUS 请求状态寄存器：

位数	复位值	Field	Value ID	Value	描述：只读寄存器
第 0 位	1				RR [0]寄存器的请求状态
		RR0	DisabledOrRequested	0	RR [0]寄存器未启用，或者已经请求重新加载
			EnabledAndUnrequested	1	RR [0]寄存器未启用，或者已经请求重新加载
第 1 位	0	RR1			RR [1]寄存器的请求状态
			DisabledOrRequested	0	RR [1]寄存器未启用，或者已经请求重新加载
			EnabledAndUnrequested	1	RR [1]寄存器未启用，或者已经请求重新加载
第 2 位	0	RR2			RR [2]寄存器的请求状态
			DisabledOrRequested	0	RR [2]寄存器未启用，或者已经请求重新加载
			EnabledAndUnrequested	1	RR [2]寄存器未启用，或者已经请求重新加载
第 3 位	0	RR3			RR [3]寄存器的请求状态
			DisabledOrRequested	0	RR [3]寄存器未启用，或者已经请求重新加载
			EnabledAndUnrequested	1	RR [3]寄存器未启用，或者已经请求重新加载
第 4 位	0	RR4			RR [4]寄存器的请求状态
			DisabledOrRequested	0	RR [4]寄存器未启用，或者已经请求重新加载
			EnabledAndUnrequested	1	RR [4]寄存器未启用，或者已经请求重新加载
第 5 位	0	RR5			RR [5]寄存器的请求状态
			DisabledOrRequested	0	RR [5]寄存器未启用，或者已经请求重新加载
			EnabledAndUnrequested	1	RR [5]寄存器未启用，或者已经请求重新加载
第 6 位	0	RR6			RR [6]寄存器的请求状态
			DisabledOrRequested	0	RR [6]寄存器未启用，或者已经请求重新加载
			EnabledAndUnrequested	1	RR [6]寄存器未启用，或者已经请求重新加载
第 7 位	0	RR7			RR [7]寄存器的请求状态
			DisabledOrRequested	0	RR [7]寄存器未启用，或者已经请求重新加载
			EnabledAndUnrequested	1	RR [7]寄存器未启用，或者已经请求重新加载

4：CRV 计数器重载值：

位数	复位值	Field	Value	描述
第 0~31 位	0xFFFFFFFF	CRV	0x0000000F~0xFFFFFFFF	计数器重新加载 32.768 kHz 时钟周期数的值

5：RREN 启用重载请求寄存器的寄存器:

位数	复位值	Field	Value ID	Value	描述：可读可写寄存器
第 0 位	1	RR0			启用或禁用 RR [0]寄存器
			Disabled	0	禁用 RR [0]寄存器
			Enabled	1	启用 RR [0]寄存器
第 1 位	0	RR1			启用或禁用 RR [1]寄存器
			Disabled	0	禁用 RR [1]寄存器
			Enabled	1	启用 RR [1]寄存器

第 2 位	0	RR2			启用或禁用 RR [2]寄存器
			Disabled	0	禁用 RR [2]寄存器
			Enabled	1	启用 RR [2]寄存器
第 3 位	0	RR3			启用或禁用 RR [3]寄存器
			Disabled	0	禁用 RR [3]寄存器
			Enabled	1	启用 RR [3]寄存器
第 4 位	0	RR4			启用或禁用 RR [4]寄存器
			Disabled	0	禁用 RR [4]寄存器
			Enabled	1	启用 RR [4]寄存器
第 5 位	0	RR5			启用或禁用 RR [5]寄存器
			Disabled	0	禁用 RR [5]寄存器
			Enabled	1	启用 RR [5]寄存器
第 6 位	0	RR6			启用或禁用 RR [6]寄存器
			Disabled	0	禁用 RR [6]寄存器
			Enabled	1	启用 RR [6]寄存器
第 7 位	0	RR7			启用或禁用 RR [7]寄存器
			Disabled	0	禁用 RR [7]寄存器
			Enabled	1	启用 RR [7]寄存器

6：RR[n] n=0~7 重新加载请求寄存器：

位数	复位值	Field	Value ID	Value	描述：可写
第 0~31 位	0x00000000	RR	Reload		重新加载请求
				0x6E524635	请求重新加载监视程序计时器的值

15.2.1.2 看门狗寄存器配置：

看门狗上是使用低频时钟源（LFCLK）提供时钟，因此使用之前，需要配置配置低速时钟。我们选择外部低速时钟振荡作为时钟源。首先用寄存器器 TASKS_LFCLKSTART 开低速时钟振荡任务，打开后就会触发低速时钟开始事件，一旦低速时钟开始事件 EVENTS_LFCLKSTARTED 被置位，就开启了低速时钟，这时候就可以跳出等待循环。这里就把低速时钟源设置完成了。

```
01.  void lfclk_config(void)
02. {    //选择时钟源
03.      NRF_CLOCK->LFCLKSRC = (CLOCK_LFCLKSRC_SRC_Xtal
04.                                          <<CLOCK_LFCLKSRC_SRC_Pos);
05.      //打开低速时钟开始任务，等待触发低速时钟开始事件
06.      NRF_CLOCK->EVENTS_LFCLKSTARTED   = 0;
07.      NRF_CLOCK->TASKS_LFCLKSTART      = 1;
08.      while (NRF_CLOCK->EVENTS_LFCLKSTARTED == 0)
09.      {
10.          //等待低速时钟事件，如果触发成功表示配置完成，跳出循环
11.      }
12.      NRF_CLOCK->EVENTS_LFCLKSTARTED = 0;
```

```
13. }
```

然后是本例编程的核心，初始化看门狗，具体代码如下所示：

```
14.    //*配置看门狗*//
15.     //配置看门狗重载值
16.       NRF_WDT->CRV=65536;
17.     //配置看门狗休眠下运行
18.        NRF_WDT->CONFIG=0x01;
19.     //申请喂狗通道，也就是使用哪个 RR
20.        NRF_WDT->RREN=0x01;
21.    //启动 WDT
22.       NRF_WDT->TASKS_START=1;
```

第 16 行：配置 CRV 计数器重载值，CRV 寄存器的重载值范围为：0x0000000F~0xFFFFFFFF，本例设置为 65536，根据公式 timeout[s] =（CRV + 1）/ 32768，超时时间 timeout[s] =2[s]，也就是说 2s 钟内不进行喂狗就会产生复位。

第 18 行: 配置 CONFIG 寄存器，在 CPU 处于休眠状态会保持运行以及调试器停止时暂停运行，赋值 0x01，其实 CONFIG 寄存器默认复位的时候就为 0x01。

第 20 行：申请喂狗通道，也就是启用哪个 RR 重新加载请求寄存器，使用 RR[0] ，则赋值为 0x01。

第 22 行：使能 TASKS_START 寄存器，开启看门狗。

如果我们需要看门狗超时中断，则需要对看门狗中断进行使能，需要使能看门狗中断和看门狗超时事件，具体代码如下所示：

```
23.    //使能看门狗定时器超时事件
24.     NRF_WDT->EVENTS_TIMEOUT=1;
25.       //使能看门狗中断
26.      NRF_WDT->INTENSET=1;
27.      //使能看门中断嵌套
28.     NVIC_EnableIRQ(WDT_IRQn);
29.
```

如果系统正常运行下，我们需要进行定时喂狗。如果在超时时间 2 秒内进行喂狗，那么 cpu 就不会重启，也不会触发超时事件中断了，正常运行下 LED 灯会依次闪亮一次，然后全部关闭。下面我们通过按键按下，来触发喂狗。喂狗就是向 RR 重新加载请求寄存器中进行赋值，赋值为 0x6E524635UL 就可以实现喂狗。具体代码如下所示。因此，如果我们 2s 内不停的按下按键 1，那么 cpu 就不会重新启动。

```
30.        //初始化后运行状态，led 闪亮一次
31.       for (uint32_t i = 0; i < LEDS_NUMBER; i++)
32.       {
33.             bsp_board_led_on(i);
34.                nrf_delay_ms(200);
35.       }
36.         bsp_board_leds_off();//然后全部关闭
37.
38.    while (1)
39.       {      //按键按下后
40.             if(nrf_gpio_pin_read(BUTTON_1) == 0)
```

```
41.             {
42.                 //实现喂狗
43.                 NRF_WDT->RR[0]=0x6E524635UL;
44.             }
45.     }
```

假如我们没有按下按键，就没有进行喂狗，看门狗超时就会参数超时溢出中断。在看门狗超时中断事件回调事件中，在 WDT 会花费的最大时间是两个 32768[Hz]时钟周期。在此之后，将发生 cpu 复位。我们在这个时间内打开全部 Led 灯，具体代码如下所示：

```
46. void wdt_event_handler(void)
47. {
48.
49.     if ((NRF_WDT->EVENTS_TIMEOUT!= 0) &&
50.         ((NRF_WDT->INTENSET) != 0))
51.     {
52.         bsp_board_leds_on();//打开所有 led 灯。
53.     }
54. }
```

因此实验下载到青云 nRF52832 开发板后的实验现象如下：

首先 LED 依次点亮，然后全部关闭，如果我们没有按下按键，首先会触发看门狗超时中断，全部 LED 灯会打开，然后 CPU 复位。由于打开的时间是两个 32768[Hz]时钟周期，到 CPU 复位的时间所有非常短，LED 灯光只能微弱全部开始闪一下。复位后，然后又开始 LED 依次点亮，重复上面的效果。

如果我们 2s 内按下了按键，CPU 就不会发生复位，LED 灯将全部保持熄灭状态。观察的时候需要不停的按下按键 1，以实现 2S 内不停的喂狗，避免 CPU 重新启动。

15.2.2 WDT 库函数的编程

15.2.2.1 看门狗的库函数 API 介绍

官方提供了一套看门狗的库函数来首先看门狗的编程，首先介绍下需要使用的看门狗 API 函数，如下所示：

首先需要初始化看门狗的参数，同时使能看门狗超时中断，配置 wdt_event_handler 看门狗中断回调函数，可以调用 nrf_drv_wdt_init 看门狗初始化函数，函数 API 如下所示：

●nrf_drv_wdt_init 函数：看门狗初始化函数：

函数：__STATIC_INLINE ret_code_t nrf_drv_wdt_init(nrf_drv_wdt_config_t const * p_config, nrf_wdt_event_handler_t	
*功能：此函数初始化 watchdog。	
* 参数： p_instance	指向本实例的指针,如果为空，则使用默认配置。
* 参数： wdt_event_handler	指定用户提供的事件处理程序。
* 返回值 NRF_SUCCESS	如果初始化成功

看门狗初始化函数中，配置了一个 NRFX_WDT_DEAFULT_CONFIG 默认参数，如下所示：

```
01. __STATIC_INLINE ret_code_t nrf_drv_wdt_init(nrf_drv_wdt_config_t const * p_config,
02.                                             nrf_wdt_event_handler_t   wdt_event_handler)
```

```
{
    if (p_config == NULL)
    {
        static const nrfx_wdt_config_t default_config = NRFX_WDT_DEAFULT_CONFIG;
        p_config = &default_config;
    }
    return nrfx_wdt_init(p_config, wdt_event_handler);
}
```

这个默认配置参数定义如下结构体，包含三个参数：

```
#define NRFX_WDT_DEAFULT_CONFIG
{       \CPU 的状态
    .behaviour = (nrf_wdt_behaviour_t)NRFX_WDT_CONFIG_BEHAVIOUR,
    .reload_value    = NRFX_WDT_CONFIG_RELOAD_VALUE,     \看门狗重导值
    .interrupt_priority = NRFX_WDT_CONFIG_IRQ_PRIORITY,   \看门狗中断的优先级
}
```

在 sdk_config.h 文件中，定义了看门狗配置的参数，如下所示，可以根据自己的需求进行选择：

```
// NRFX_WDT_CONFIG_BEHAVIOUR 在 CPU 休眠或停止模式下的 WDT 的状态
// <1=> 休眠的时候运行，调试停止时暂停
// <8=> 休眠的时候暂停，调试停止时运行
// <9=> 休眠的时候运行，调试停止时运行
// <0=> 休眠的时候暂停，调试停止时暂停

#ifndef NRFX_WDT_CONFIG_BEHAVIOUR
#define NRFX_WDT_CONFIG_BEHAVIOUR 1
#endif

//   NRFX_WDT_CONFIG_RELOAD_VALUE 重导值，范围在<15-4294967295>


#ifndef NRFX_WDT_CONFIG_RELOAD_VALUE
#define NRFX_WDT_CONFIG_RELOAD_VALUE 2000
#endif

// NRFX_WDT_CONFIG_IRQ_PRIORITY 中断优先级

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

```

```
29. #ifndef NRFX_WDT_CONFIG_IRQ_PRIORITY
30. #define NRFX_WDT_CONFIG_IRQ_PRIORITY 7
31. #endif
```

看门狗喂狗具有 RR[0]~RR[7] 8 个通道，可以通过 nrfx_wdt_channel_alloc 函数通道分配函数选择使用哪一个看门狗通道，然后使用函数 nrfx_wdt_channel_feed 在对应的通道内进行喂狗，具体函数 API 如下：

●#define nrf_drv_wdt_channel_alloc nrfx_wdt_channel_alloc 函数：看门狗通道分配函数

函数：nrfx_err_t nrfx_wdt_channel_alloc(nrfx_wdt_channel_id * p_channel_id);
*功能：此函数分配看门狗通道。这个函数不能在 nrfx_wdt_start(void)之后调用。
* 参数： p_channel_id 配置的通道 ID。
* 返回值 NRF_SUCCESS 如果初始化成功

● nrfx_wdt_channel_feed 函数：看门狗喂狗函数

函数：void nrfx_wdt_channel_feed(nrfx_wdt_channel_id channel_id);
*功能：函数功能为对看门狗通道进行喂狗。
*参数： p_channel_id 配置的通道 ID。
*返回值 无

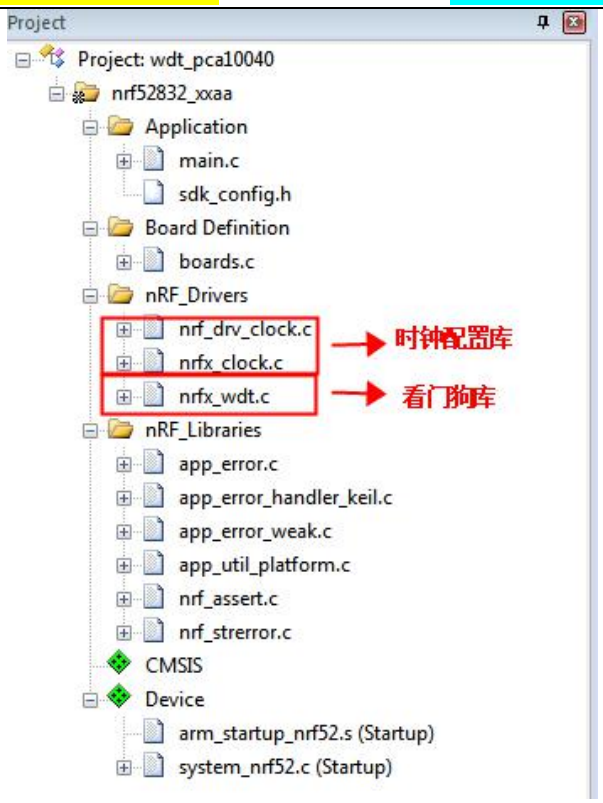
最后配置完成后，需要启动看门狗，调用 nrfx_wdt_enable 看门狗使能函数：

●#define nrf_drv_wdt_enable nrfx_wdt_enable 函数：看门狗启动函数

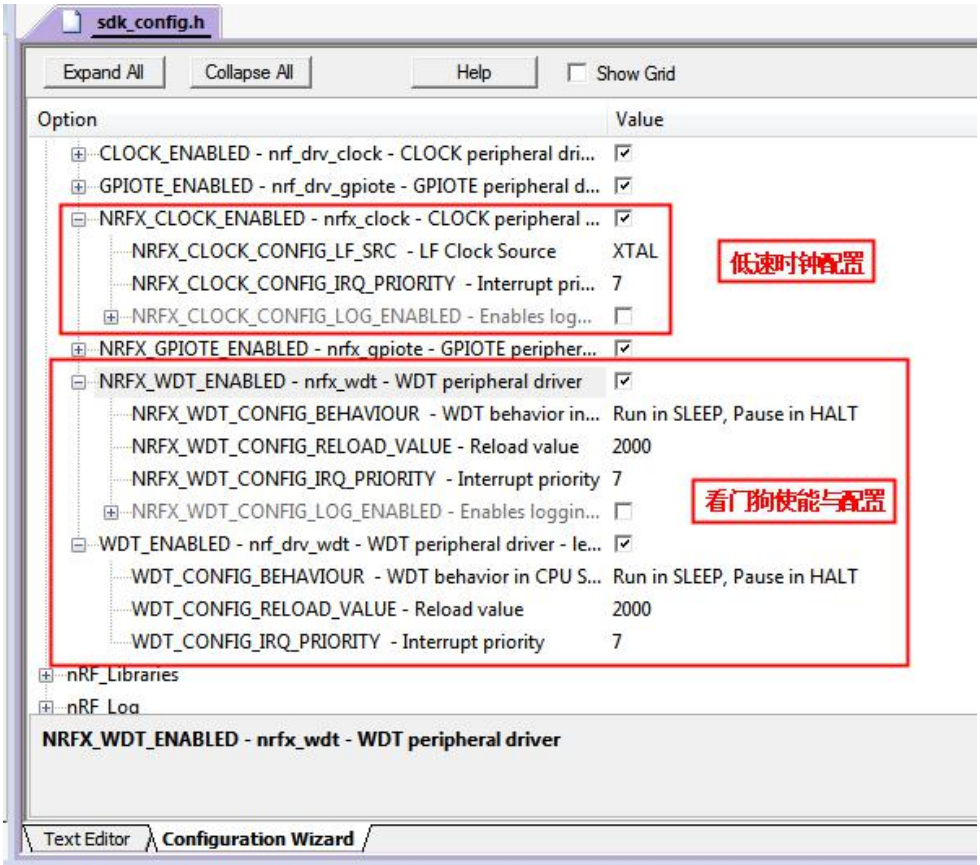
函数：void nrfx_wdt_enable(void);
*功能：调用此函数后，看门狗将启动。因此用户需要去喂狗所有分配的看门狗通道以避免复位。注意至少要分配一个看门狗通道。
* 参数：无
* 返回值 无

15.2.2.1 看门狗的库函数配置：

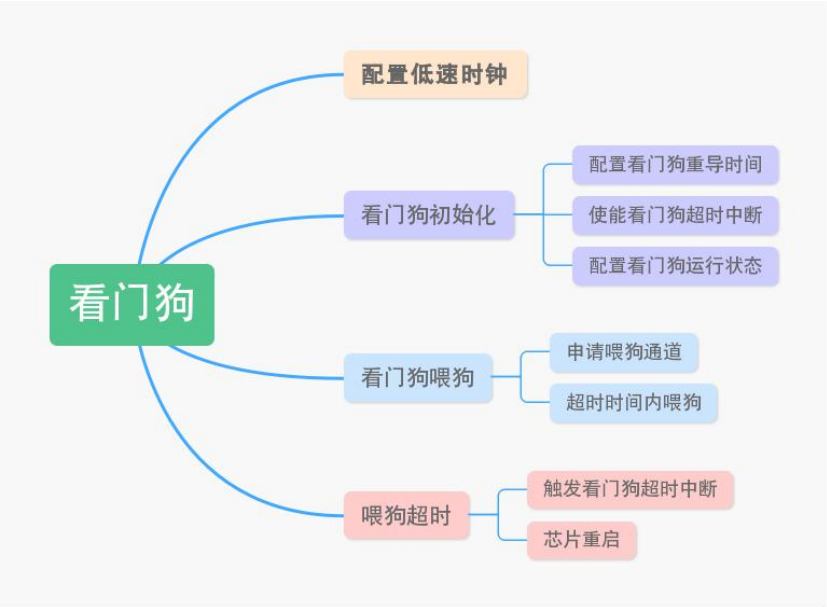
看门狗的库函数工程目录树可以以 GPIOTE 工程的目录树为基础进行修改，删除掉 GPIOTE 外设，添加如下图所示的三个驱动库文件：nrf_drv_clock.c、nrfx_clock.c、nrfx_wdt.c。其中驱动文件 nrf_drv_clock.c、nrfx_clock.c 是实在配置库函数，nrfx_wdt.c 为看门狗的驱动库函数。



同时需要在 sdk_config.h 文件中配置如下图的选项：低速时钟的配置和看门狗的三个默认配置参数，如果配置成功，在 sdk_config.h 文件的配置导航框 Cofiguration Wizard 中会出现如下图所示的勾选选项和参数值：



根据寄存器的例子，总结实验的看门狗配置结构可以如下框图所示：



根据配置步骤，首先我们来配置看门狗初始化，并且使能超时中断，声明超时中断处理回调函数 wdt_event_handler，然后申请看门狗的喂狗通道，最后对看门狗进行使能，具体代码如下所示：

```
//配置看门狗
nrf_drv_wdt_config_t config = NRF_DRV_WDT_DEAFULT_CONFIG;
//初始化看门狗和看门狗中断
err_code = nrf_drv_wdt_init(&config, wdt_event_handler);
APP_ERROR_CHECK(err_code);
 //分配看门狗通道
err_code = nrf_drv_wdt_channel_alloc(&m_channel_id);
APP_ERROR_CHECK(err_code);
//看门狗使能
nrf_drv_wdt_enable();
```

注意配置的时候把默认重导值 NRFX_WDT_CONFIG_RELOAD_VALUE 配置为 65536，也就是和配置寄存器 CRV 的值相同，2s 为喂狗超时时间。

喂狗之前把 LED 灯依次点亮依次，然后关闭所示 LED 灯，代码如下所示：

```
//Indicate program start on LEDs.
for (uint32_t i = 0; i < LEDS_NUMBER; i++)
{
    bsp_board_led_on(i);
    nrf_delay_ms(200);
}
 bsp_board_leds_off();
```

如果系统正常运行下，我们需要进行定时喂狗。如果在超时时间 2 秒内进行喂狗，那么 cpu 就不会重启，也不会触发超时事件中断了。下面我们通过按键按下，来触发喂狗，需要注意要在 LED 全部熄灭后 2S 之内按下按键才能够实现喂狗。

```
while (1)
{
    if(nrf_gpio_pin_read(BUTTON_1) == 0)
    {
```

```
22.                     //D1 点亮指示按键 S1 按下
23.                       nrf_gpio_pin_clear(LED_2);
24.                     //喂狗
25.                     nrfx_wdt_channel_feed(m_channel_id);
26.             }
27.     }
```

假如我们没有按下按键，就没有进行喂狗，看门狗超时就会参数超时溢出中断。在看门狗超时中断事件回调事件中，在 WDT 会花费的最大时间是两个 32768[Hz]时钟周期。在此之后，将发生 cpu 复位。我们在这个时间内打开全部 Led 灯，具体代码如下所示：

```
28. /**
29.  * 看门狗事件回调
30.  */
31. void wdt_event_handler(void)
32. {
33.     bsp_board_leds_on();
34.     nrf_gpio_pin_set(12);
35. }
```

因此实验下载到青云 nRF52832 开发板后的实验现象和寄存器方式相同：

首先 LED 依次点亮，然后全部关闭，如果我们没有按下按键，首先会触发看门狗超时中断，全部 LED 灯会打开，然后 CPU 复位。由于打开的时间是两个 32768[Hz]时钟周期，到 CPU 复位的时间所有非常短，LED 灯光只能微弱全部开始闪一下。复位后，然后又开始 LED 依次点亮，重复上面的效果。如果我们 2s 内按下了按键，CPU 就不会发生复位，LED 灯将全部保持熄灭状态。观察的时候需要不停的按下按键 1，以实现 2S 内不停的喂狗，避免 CPU 重新启动。