LE5010睡眠模式的应用说明

1）3种睡眠模式的介绍

	LP0	LP1	LP2	LP3
RAM Retention	On	On	Off	Off
32K	On	Off	On	Off
Power Consumption	~uA		<1uA	<500nA
Scenario	BLE ADV/CONN		Wakeup from RTC & Reboot	Wakeup from IO & Reboot
	Scheduled By SDK Automatically		Triggered by APP	Triggered by APP

有三种睡眠模式，LP0、LP2、LP3,各自的特点：

- LP0：蓝牙正常工作，空闲情况下进入睡眠状态：SDK软件自动进入睡眠模式，睡眠前的IO口保持动作SDK已经处理了，应用软件不用在处理；
- LP2：睡眠之后蓝牙不工作，可以通过RTC和外部中断引脚唤醒，每次唤醒相当于复位；
- LP3：睡眠之后蓝牙不工作，可以通过外部中断引脚唤醒，每次唤醒相当于复位。

  进入LP2和LP3需要软件配置睡眠唤醒的方式，然后调用接口进入睡眠模式：不能通过BLE事件唤醒。

  注：所有睡眠模式下，当使用串口打印时，需要将RX上拉（避免RX漏电导致功耗偏高。原因：UART RX 口为浮空输入时，会有漏电，所以要加个上拉，配置内部上拉就可以）,外挂32k晶振的情况下，只能在LP0模式下工作。

2）BLE应用

要进入睡眠状态需要将外设以及映射到的IO进行反初始化、软件定时器关掉。

例如：对串口进行反初始化接口

```
HAL_StatusTypeDef HAL_UART_DeInit(UART_HandleTypeDef *huart);
void uart1_io_deinit(void);
```

LP0模式

LP0的睡眠和广播间隔，发送功率还有外设的工作状态有关。BLE的例程在ble_loop()里调用了睡眠函数，只需在app_config.h调用如下宏：

```
#define SDK_DEEP_SLEEP_ENABLE 1
#define DEBUG_MODE 0
```

测试数据：

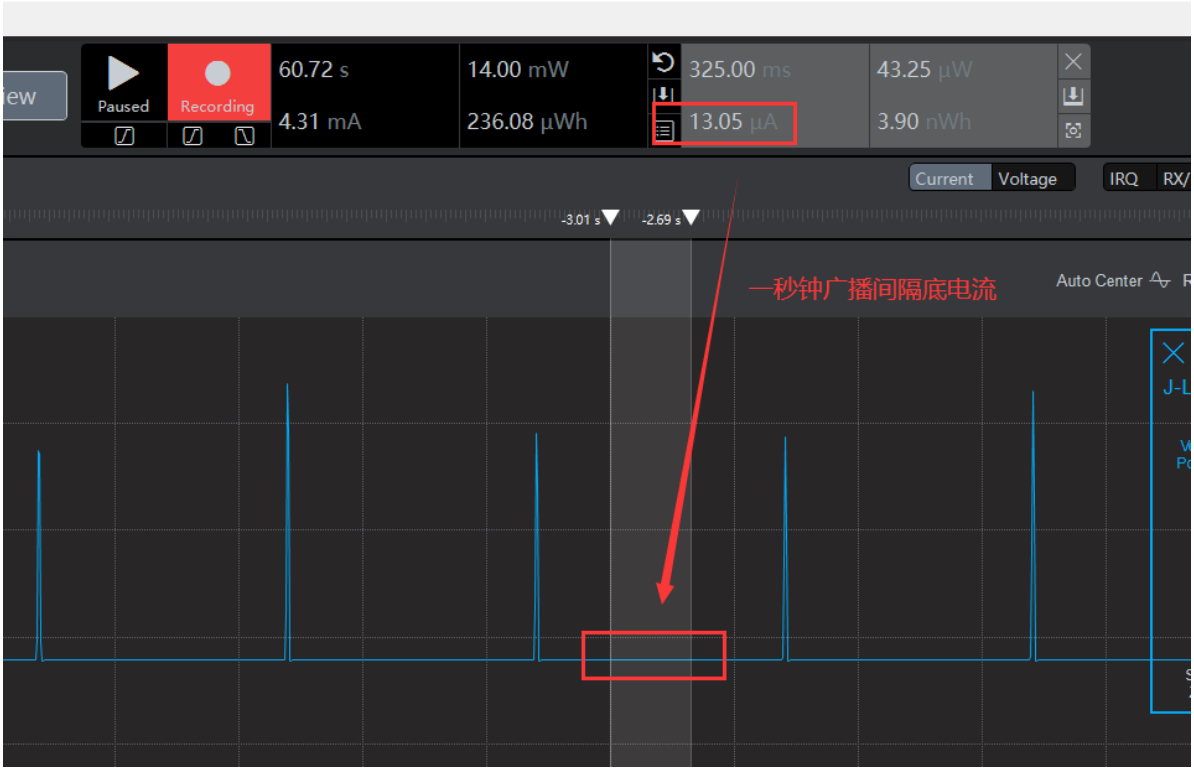
		LP0				LP2	LP3
		底电流	ADV 平均	CE 平均		底电流	底电流
DCDC_ON	16M	13.58uA	48.06uA	LOW(15ms)	1.76mA	1.24uA	755.99nA
				BALANCED(47.5ms)	594.28uA		
				HIGH(125ms)	240.49uA		
	64M	13.04uA	61.88uA	LOW(15ms)	2.12mA	1.49uA	765.79nA
				BALANCED(47.5ms)	691.35uA		
				HIGH(125ms)	275.71uA		
DCDC_OFF	16M	13.05uA	56.27uA	LOW(15ms)	1.35mA	1.28uA	767.1nA
				BALANCED(47.5ms)	469uA		
				HIGH(125ms)	199.45uA		
	64M	12.68uA	72.54uA	LOW(15ms)	2.19mA	1.51uA	775.98nA
				BALANCED(47.5ms)	719.47uA		
				HIGH(125ms)	304.32uA		

实测功耗:

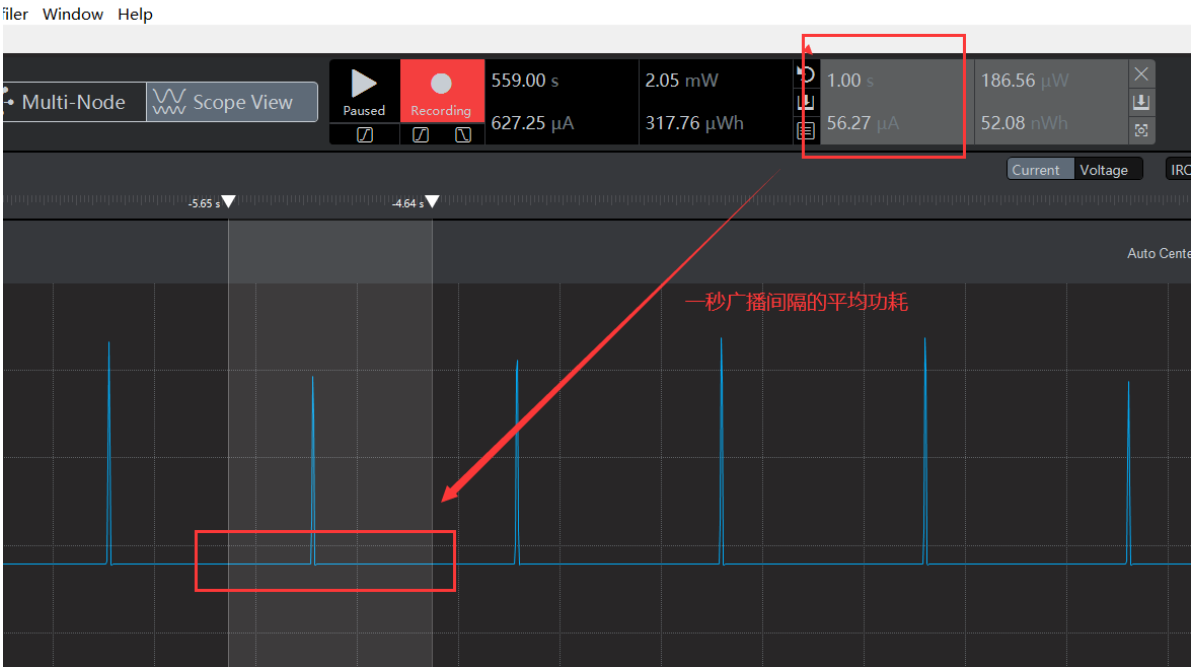
16M主频一秒钟广播间隔DCDC_OFF 底电流

SDK默认64M主频，修改主频需要在app_config.h配置宏：

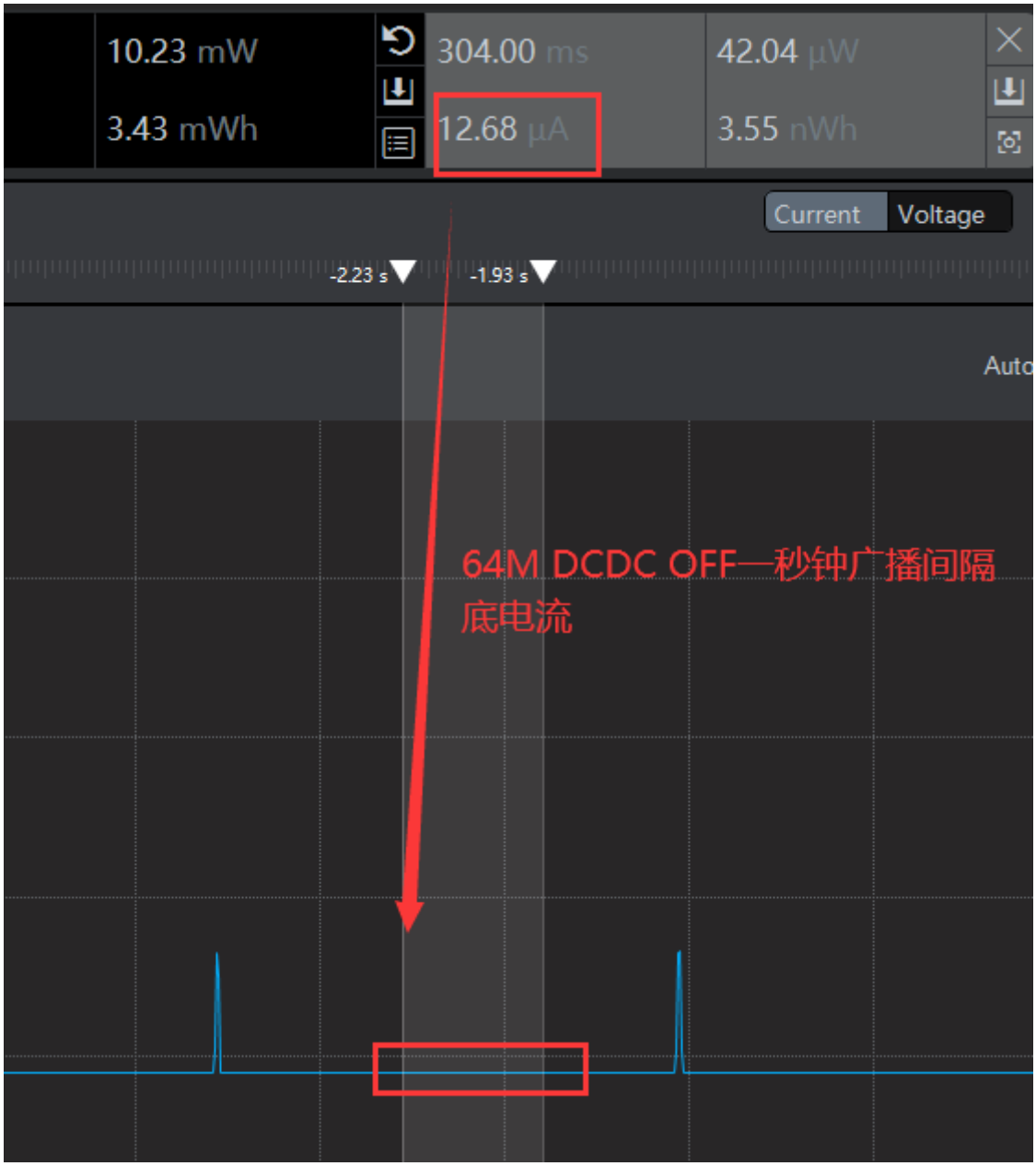
```
1 #define SDK_HCLK_MHZ    (16)
```



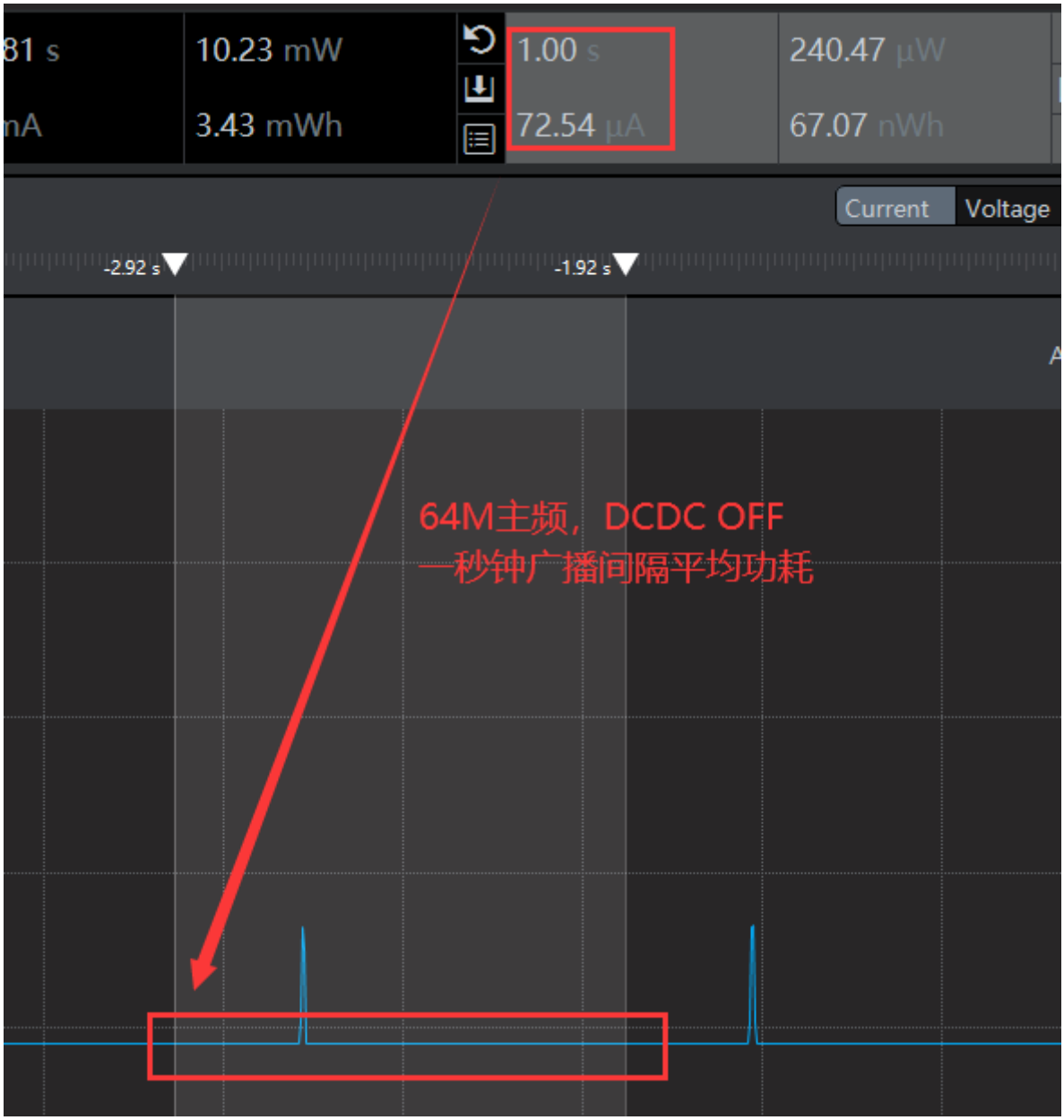
16M主频一秒钟广播间隔DCDC_OFF平均功耗：



64M主频一秒钟广播间隔DCDC_OFF 底电流:

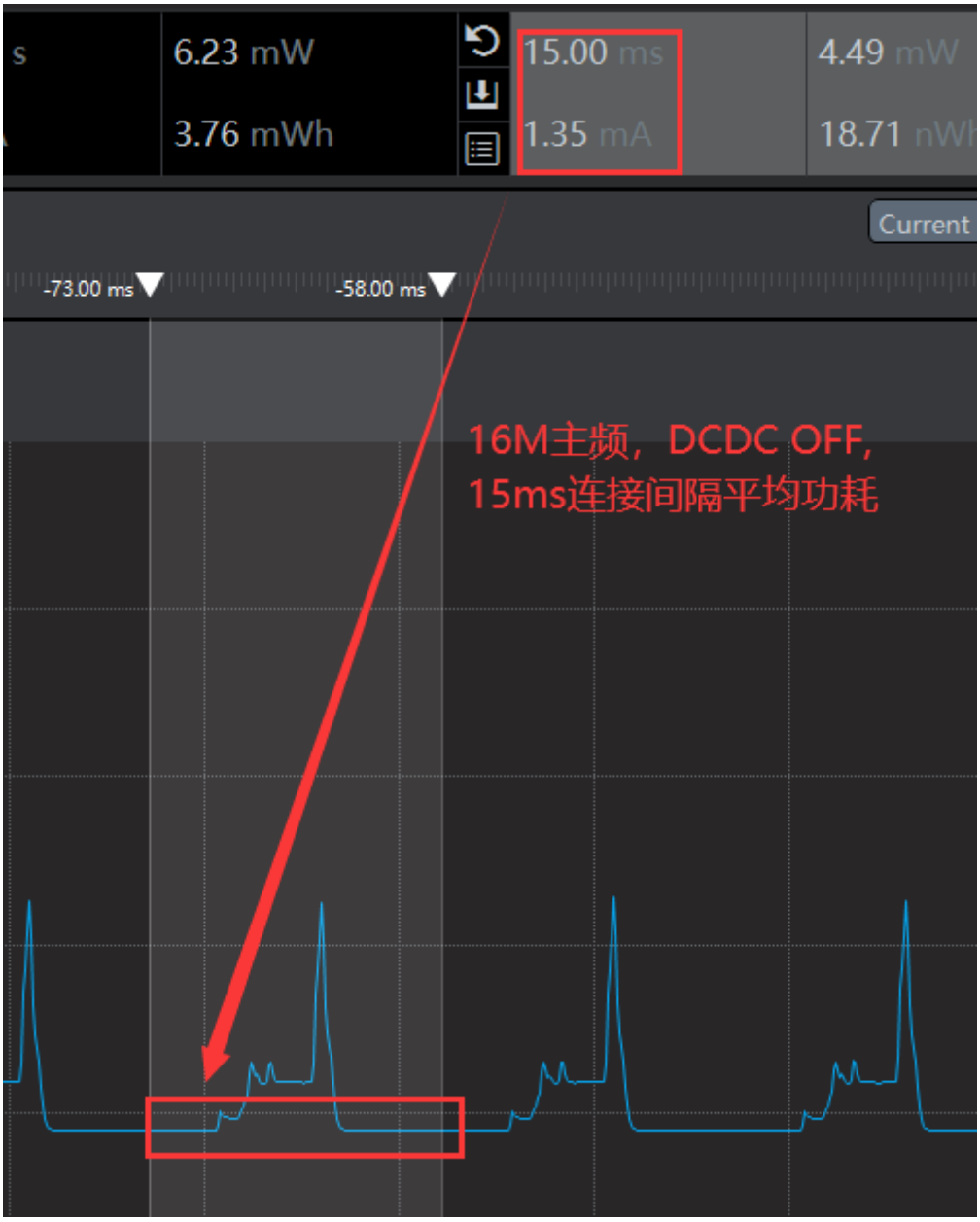


64M主频一秒钟广播间隔DCDC_OFF 平均功耗:

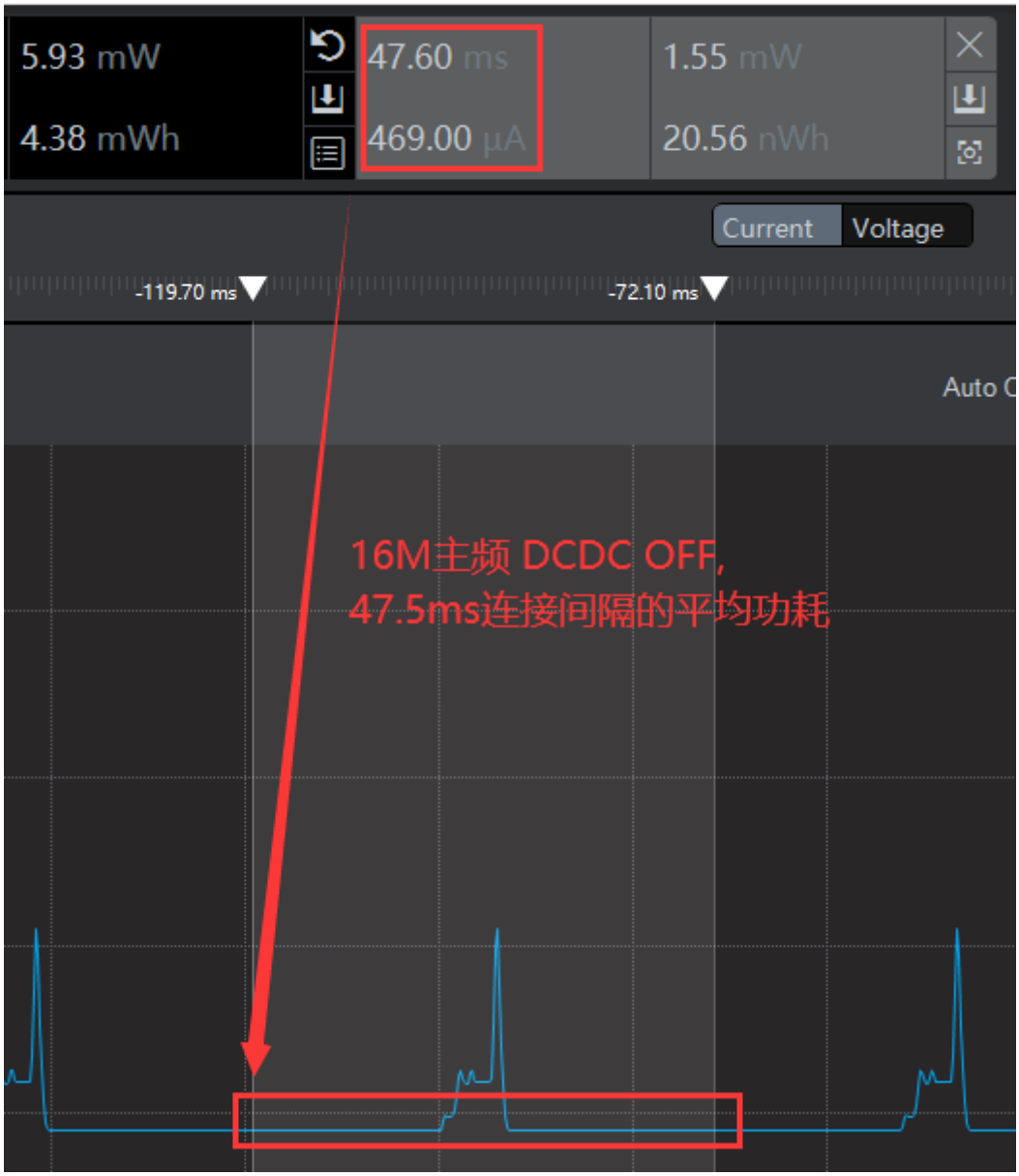


16M主频不同连接间隔下的平均功耗：

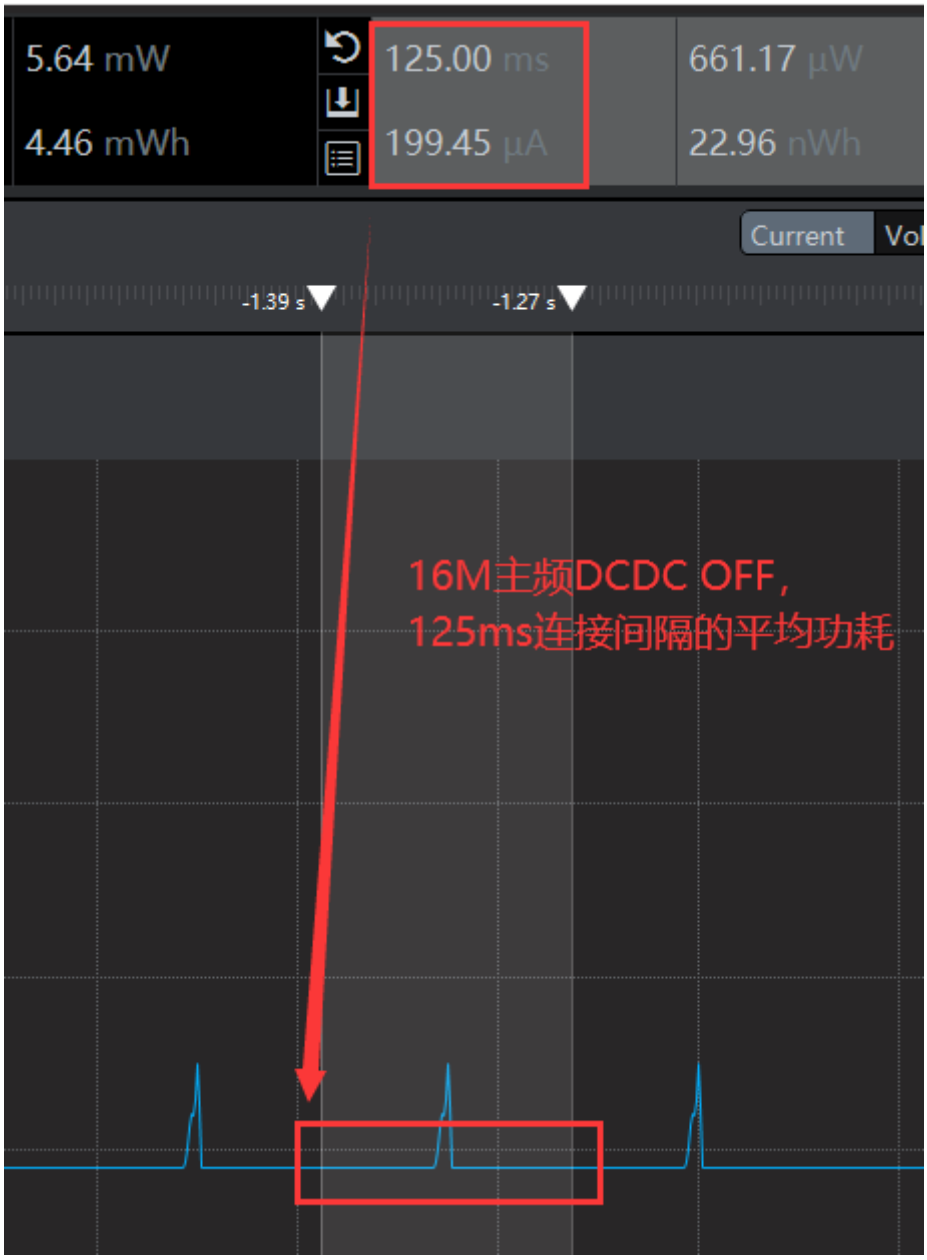
- LOW:


s
6.23 mW
3.76 mWh
15.00 ms
1.35 mA
4.49 mW
18.71 nWh
Current
-73.00 ms
-58.00 ms
16M主频，DCDC OFF,
15ms连接间隔平均功耗

- BALANCED:

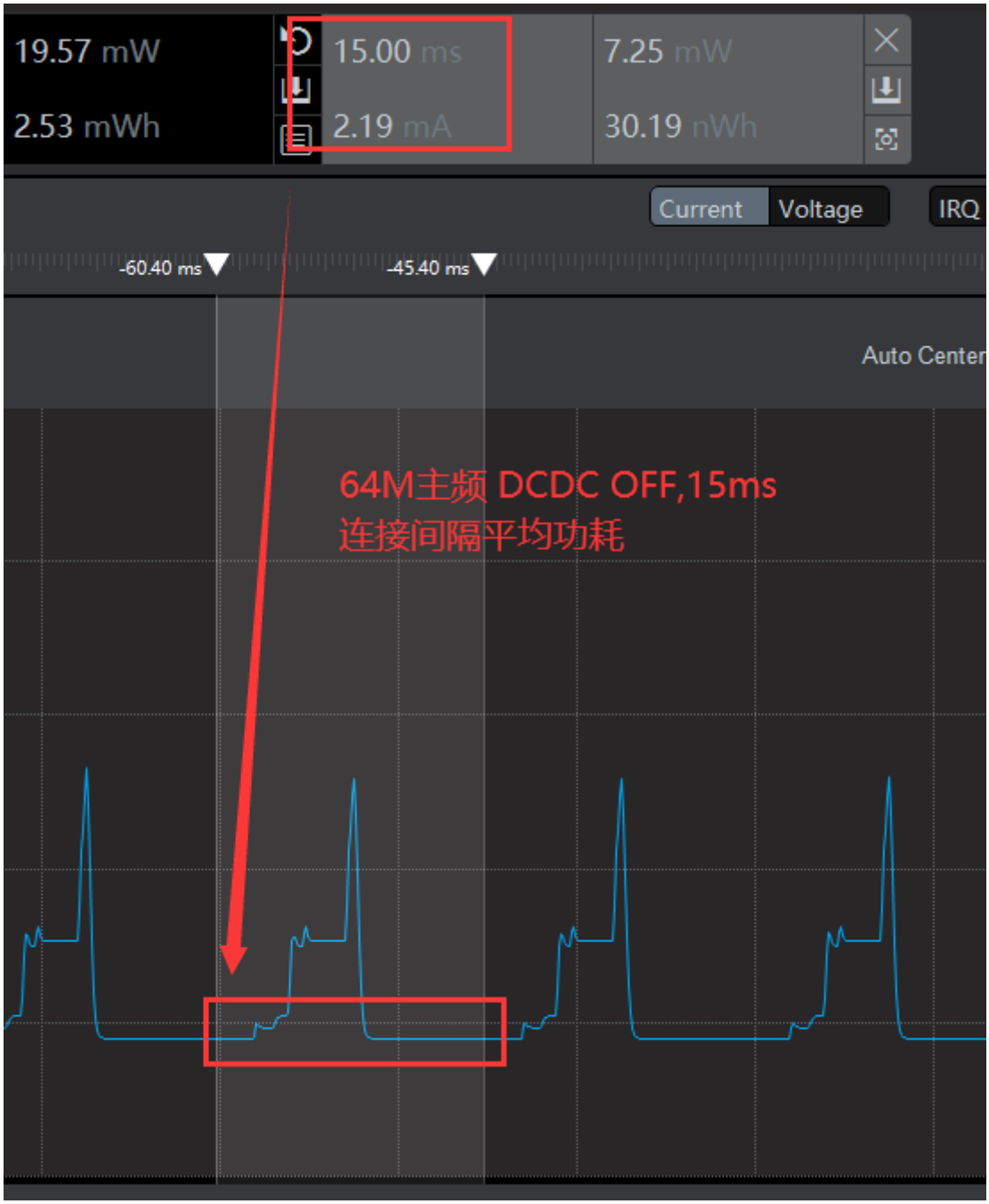

5.93 mW
4.38 mWh
47.60 ms
469.00 µA
1.55 mW
20.56 nWh
Current
Voltage
-119.70 ms
-72.10 ms
16M主频 DCDC OFF,
47.5ms连接间隔的平均功耗

- HIGH:

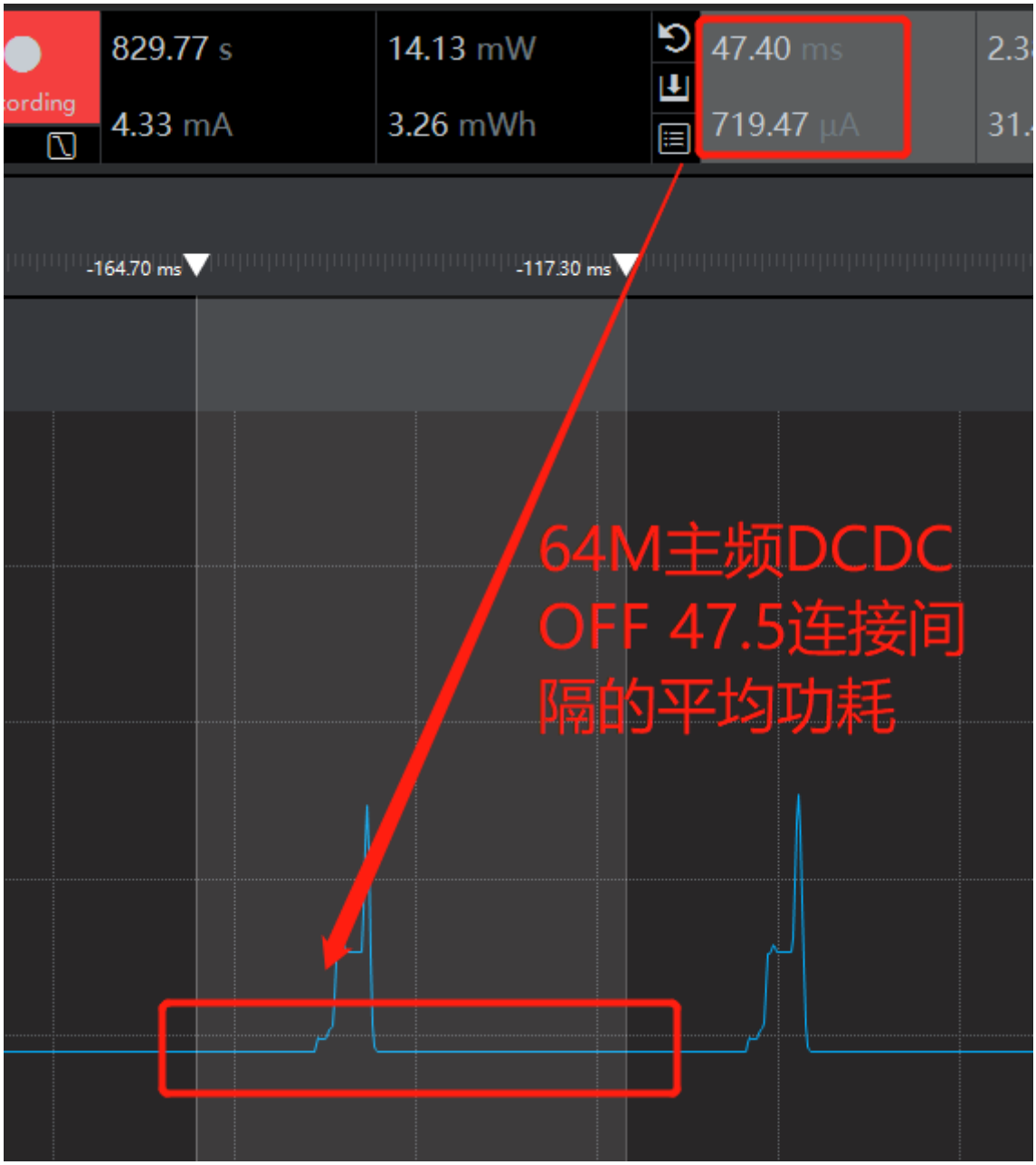


64M主频不同连接间隔下的平均功耗：

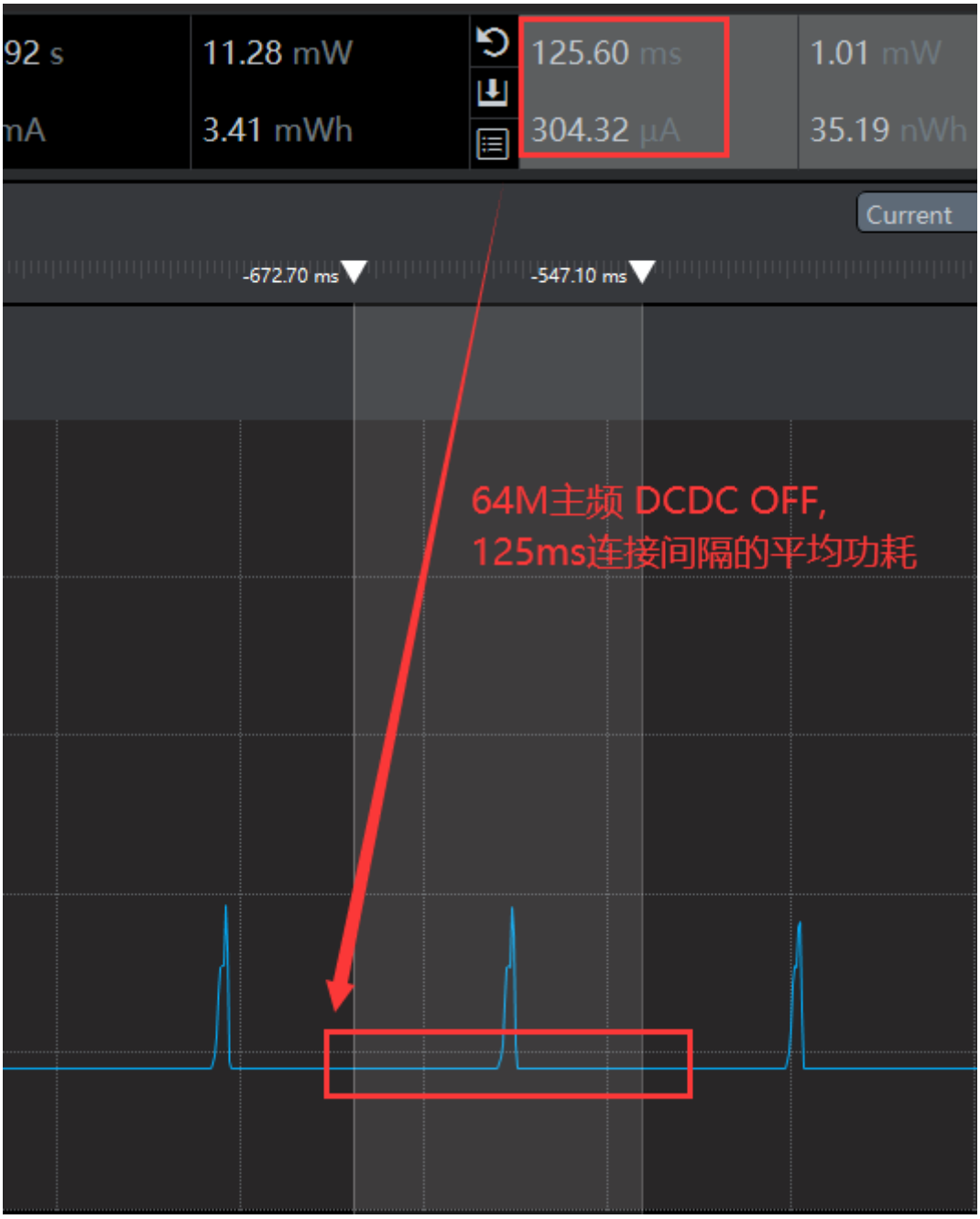
- LOW:


19.57 mW
2.53 mWh
15.00 ms
2.19 mA
7.25 mW
30.19 nWh
Current
Voltage
IRQ
-60.40 ms
-45.40 ms
Auto Center
64M主频 DCDC OFF,15ms
连接间隔平均功耗

- BALANCED:


829.77 s
14.13 mW
47.40 ms
4.33 mA
3.26 mWh
719.47 µA
-164.70 ms
-117.30 ms
64M主频DCDC OFF 47.5连接间隔的平均功耗

- HIGH:



LP2模式

进入睡眠状态需要将外设以及映射到的IO进行反初始化、软件定时器关掉LP2模式有RTC和GPIO唤醒，可同时使用；有四个外部中断唤醒io口：PA00,PA07,PB11,PB15;添加头文件：

```
#include "lsrtc.h"
#include "field_manipulate.h",
#include "sleep.h"
```

同样要配置两个宏：

```
#define SDK_DEEP_SLEEP_ENABLE 1
#define DEBUG_MODE 0
```

配置参数：PB15上升沿和RTC唤醒

```c
static void  ls_sleep_enter_lp2(void)
{
      struct deep_sleep_wakeup wakeup;
      memset(&wakeup,0,sizeof(wakeup));
      wakeup.pb15 = 1 ;                              //选择PB15作为唤醒io
      wakeup.pb15_rising_edge = 1;                   //选择上升沿唤醒
      wakeup.rtc = 1 ;                               //选择LP2模式
      enter_deep_sleep_mode_lvl2_lvl3(&wakeup);  //调用睡眠函数
}
```

io上升沿唤醒和RTC唤醒函数的配置：

```c
void exit_iowkup_init(void)
{
    io_cfg_input(PB15);
    io_pull_write(PB15, IO_PULL_DOWN);
    io_exti_config(PB15,INT_EDGE_RISING);
    io_exti_enable(PB15,true);
}
void RTC_WKUP_INIT(void )
{
     HAL_RTC_Init(RTC_CKSEL_LSI);
     RTC_wkuptime_set(1);
}
```

配置唤醒参数：PB15下降沿唤醒

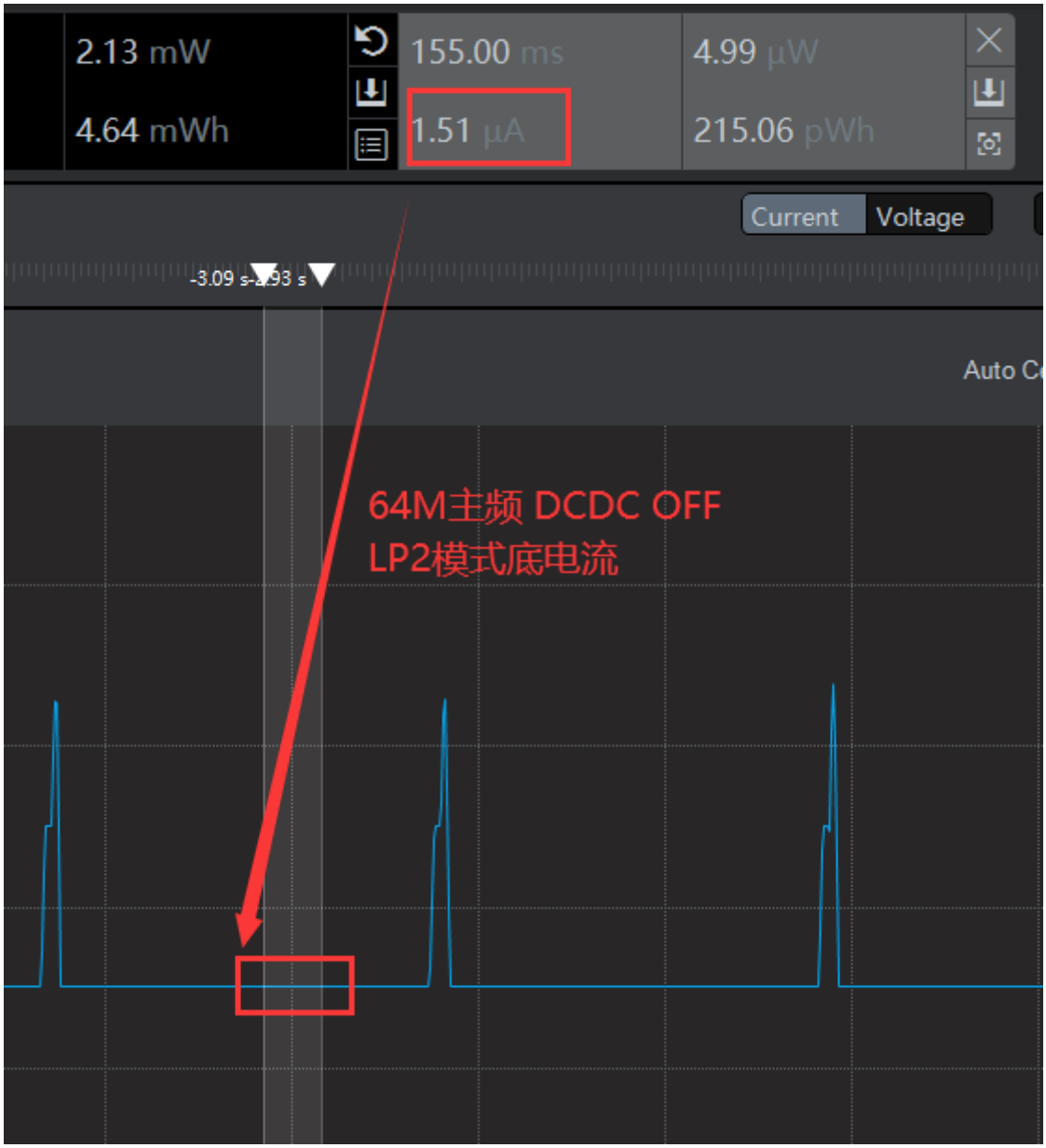
```c
static void  ls_sleep_enter_lp2(void)
{
    struct deep_sleep_wakeup wakeup;
    memset(&wakeup,0,sizeof(wakeup));
    wakeup.pb15 = 1 ;
    wakeup.pb15_rising_edge = 0;
    wakeup.rtc  = 1 ;
    enter_deep_sleep_mode_lvl2_lvl3(&wakeup);//调用唤醒函数
}
```

io下降沿唤醒函数

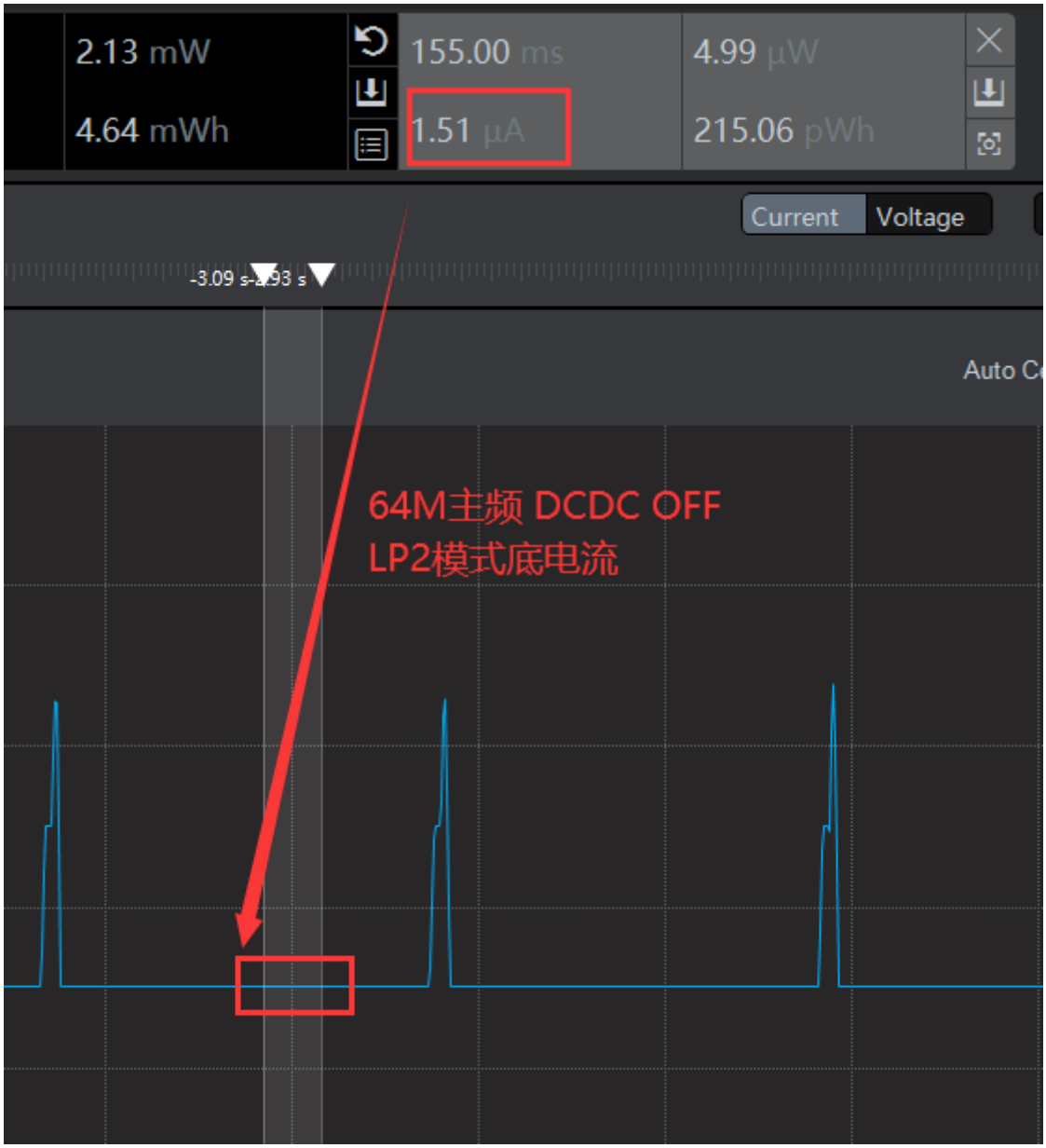
```c
void exitpb15_iowkup_init(void)
{
    io_cfg_input(PB15);
    io_pull_write(PB15, IO_PULL_UP);
    io_exti_config(PB15,INT_EDGE_FALLING);
    io_exti_enable(PB15,true);
}
```

实测功耗

- 16M主频DCDC_OFF底电流:



- 64M主频DCDC_OFF底电流:



LP3模式

LP3睡眠模式可以用外部中断唤醒，在app_config.h配置两个宏：

```
#define SDK_DEEP_SLEEP_ENABLE 1
#define DEBUG_MODE 0
```

配置参数：外部中断上升沿唤醒

```
static void ls_sleep_enter_LP3(void)
{
  struct deep_sleep_wakeup  wakeup;
    memset (&wakeup,0,sizeof(wakeup));
    wakeup.pa07 = 1;
    wakeup.pa07_rising_edge = 1;
    enter_deep_sleep_mode_lvl2_lvl3(&wakeup);//调用睡眠函数
}
```

唤醒函数配置如下：

```c
void exitpa07_iowkup_init(void)
{
    io_cfg_input(PA07);
    io_pull_write(PA07, IO_PULL_DOWN);
    io_exti_config(PA07,INT_EDGE_RISING);
    io_exti_enable(PA07,true);
}
```

配置参数：外部中断下降沿唤醒

```c
static void ls_sleep_enter_LP3(void)
{
  struct deep_sleep_wakeup  wakeup;
    memset (&wakeup,0,sizeof(wakeup));
    wakeup.pa07 = 1;
    wakeup.pa07_rising_edge = 0;
    enter_deep_sleep_mode_lvl2_lvl3(&wakeup);//调用睡眠函数
}
```

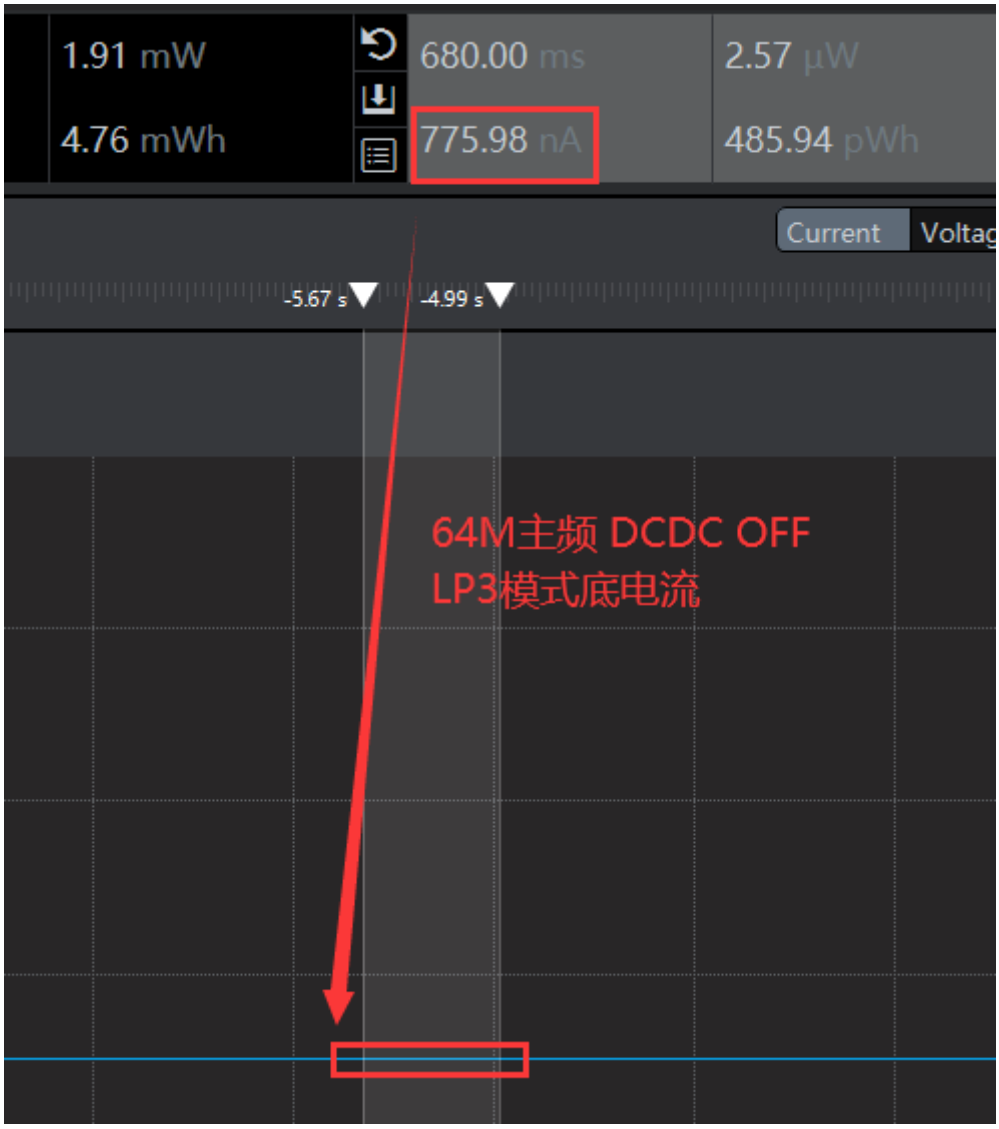
唤醒函数配置如下：

```c
void exitpa07_iowkup_init(void)
{
    io_cfg_input(PA07);
    io_pull_write(PA07, IO_PULL_UP);
    io_exti_config(PA07,INT_EDGE_FALLING);
    io_exti_enable(PA07,true);
}
```

实测功耗：

- 16M主频DCDC_OFF底电流:

- 64M主频DCDC_OFF底电流:



3)MCU应用

外设在工作的时候是不能睡下去的，用完要 反初始化掉，注意外设的引脚也要反初始化处理，可通过RTC,外部IO唤醒；添加头文件#include "sleep.h"，在app_config.h配置如下两个宏：

```
#define SDK_DEEP_SLEEP_ENABLE 1
#define DEBUG_MODE 0
```

MCU应用LP0模式需调用睡眠函数：deep_sleep_no_ble()，LP2模式和LP3模式参照BLE应用下的配置，还要进行如下图设置：将sleep.c下的：

```
XIP_BANNED void before_wfi()
{
    sleep_mode_set();
//    while(REG_FIELD_RD(SYSCFG->PMU_PWR, SYSCFG_BLE_PWR3_ST)); //MCU应用注释这行代码
    ble_hclk_clr();
    switch_to_xo16m();
    SYSCFG->ANACFG0 &= ~(SYSCFG_EN_DPLL_MASK | SYSCFG_EN_DPLL_16M_RF_MASK |
        SYSCFG_EN_DPLL_128M_RF_MASK | SYSCFG_EN_DPLL_128M_EXT_MASK |
   SYSCFG_EN_QCLK_MASK);
    MODIFY_REG(SYSCFG->ANACFG1,SYSCFG_XO16M_ADJ_MASK | SYSCFG_XO16M_LP_MASK,
        (uint32_t)3<<SYSCFG_XO16M_ADJ_POS |
(uint32_t)0<<SYSCFG_XO16M_LP_POS);
}
```

4) 获取唤醒源

接口:

```
uint8_t get_wakeup_source(void),
```

不同唤醒源对应的返回值代码路径:　　ls_ble_sdk\dev\soc\arm_cm\le501x\sleep.h

获取RTC唤醒源测试代码：上电后进入休眠，RTC一秒后唤醒进行服务的添加和外设的初始化。

```
    case STACK_READY:
    {
        uint8_t addr[6];
        bool type;
        dev_manager_get_identity_bdaddr(addr,&type);
        LOG_I("type:%d,addr:",type);
        LOG_HEX(addr,sizeof(addr));
        HAL_RTC_Init(RTC_CKSEL_LSI);
        RTC_wkuptime_set(1);
        uint8_t wkup_source = get_wakeup_source();
        if(wkup_source == RTC_WKUP)
        {
            LOG_I("WKUP_SOURCE = %d",wkup_source);
            dev_manager_prf_dis_server_add(NO_SEC,0xffff);
            ls_uart_init();
        }
        else
        {
            ls_sleep_enter_lp2();
        }
    }break;
```

LOG打印出来的信息:

```
00> I/PLF:sys init
00> I/TINYFS:directory tree print:(-dir(idx) +record(idx))
00> 0(0)
00> |-65534(1)
00>   |+4(3)
00>   |+1(2)
00>
00> I/MAIN:type:1,addr:
00> 36447F7C51FE
00> I/MAIN:WKUP_SOURCE = 2
00> I/MAIN:profile:0, start handle:0xe
00>
```

获取PB15下降沿唤醒源测试代码：上电后进入休眠，PB15下降沿唤醒进行服务的添加和外设的初始化。

```
    case STACK_READY:
    {
        uint8_t addr[6];
        bool type;
        dev_manager_get_identity_bdaddr(addr,&type);
        LOG_I("type:%d,addr:",type);
        LOG_HEX(addr,sizeof(addr));
        exitpb15_iowkup_init();
        uint8_t wkup_source = get_wakeup_source();
        if(wkup_source == PB15_IO_WKUP_EDGE_FALLING)
          {
            LOG_I("WKUP_SOURCE = %d",wkup_source);
            dev_manager_prf_dis_server_add(NO_SEC,0xffff);
            ls_uart_init();
          }
         else
          {
            ls_sleep_enter_lp2();
          }
    }break;
```

LOG打印出来的信息:

```
00> I/PLF:sys init
00> I/TINYFS:directory tree print:(-dir(idx) +record(idx))
00> 0(0)
00>
00> I/MAIN:type:1,addr:
00> 630F6E0D43CF
00> I/MAIN:WKUP_SOURCE = 0
00> I/MAIN:profile:0, start handle:0xe
00>
```

注意事项：LP2 和 LP3 的睡眠模式，IO 口唤醒的时候，上升沿和下降沿都能唤醒；但是，LP2 如果配置了两种唤醒边沿的模式， 上升沿唤醒 之后，获取唤醒源时，下降沿唤醒的引脚对应的唤醒源都会被置位；