RTL8762C Deep Low Power State

Version 1.1

2018/09/07

修订历史（Revision History）

日期	版本	修改	作者	Reviewer
2018/06/11	V1.0	第一版发布	Lory	Grace
2018/09/12	V1.1	更新 2.1 小节	Lory	Grace

目 录

图目录

1 DLPS 模式概述

RTL8762C 支持三种功耗模式： Power Down 模式， DLPS (Deep Lower Power State)模式和 Active 模式。此文档将详细介绍 DLPS 模式。

1.1 特性与限制

RTL8762C DLPS 模式有如下特点与限制：

1. 系统功耗低至 3uA@3V。
2. 系统从 DLPS 恢复需要 2ms 左右，进入 DLPS 需要 1.5ms 左右。
3. PAD 信号可以将系统从 DLPS 状态唤醒。
4. BLE 广播和连接事件可以周期性将系统从 DLPS 状态唤醒。
5. CPU 断电会导致 SWD 断开连接， 因此在使用 keil 进行在线 debug 时，需要禁止 DLPS 模式。

1.2 原理

系统在大部分时间处于空闲状态，此时可以让系统进入 DLPS 模式以降低功耗。在 DLPS 模式下，clock，CPU，Peripherals 等模块会掉电，掉电前需要保存必要的数据用以恢复系统。 当有事件需要处理时，系统会退出 DLPS 模式，并将 clock，CPU，Peripherals 等模块重新上电并恢复到进 DLPS 前的状态，然后响应唤醒事件。

2 DLPS 模式的进入与退出

图 2-1 描述了 DLPS 进出流程。

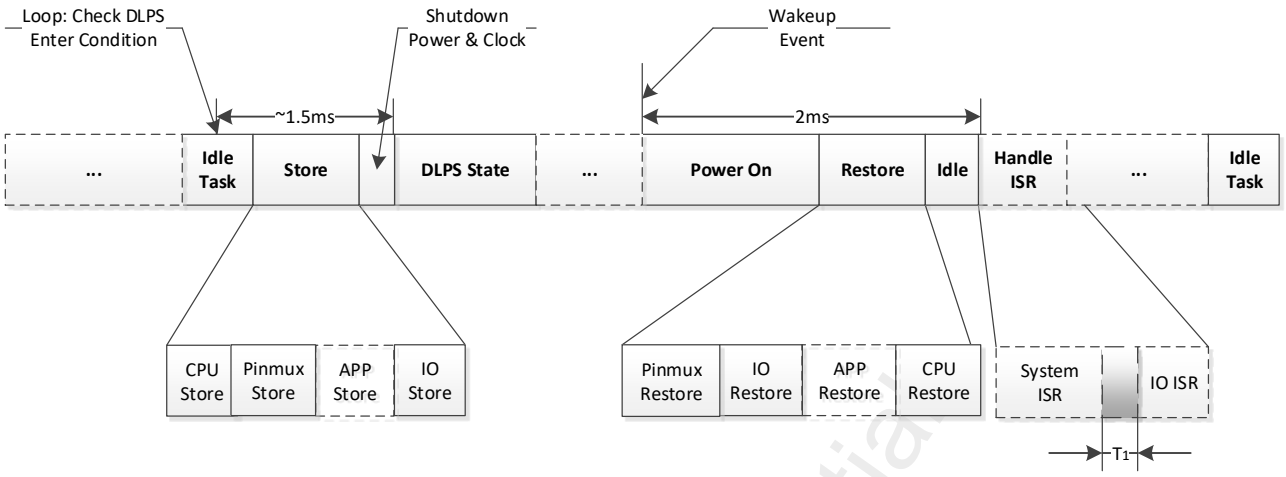

图 2-1 DLPS 进入和退出流程

2.1 DLPS 进入条件

当满足以下条件时，系统会进入 DLPS 模式：

1. 系统执行在 idle task， 其余 task 均处在 block 状态或 suspend 状态， 且没有中断发生。
2. OS，Stack，Peripheral，APP 注册的 Check callback 执行后均返回 true。
3. SW timer 周期大于 20ms。
4. BT 相关行为满足以下之一：
   1) Standby State
   2) BT Advertising State，Advertising Interval*0.625ms >=20ms
   3) BT Scan State，(Scan Interval – Scan Window)*0.625ms >= 15ms
   4) BT connection as Master role，Connection interval * 1.25ms > 12.5ms
   5) BT connection as Slave role，Connection interval* (1+ Slave latency)*1.25ms > 12.5ms

在 idle task 里会 check OS，Stack，Peripherals 及 APP，如果各个模块都允许进入 DLPS，则开始保存系统状态，然后进入 DLPS。

2.2 DLPS 唤醒条件

系统可以由以下事件唤醒退出 DLPS 模式：

1. PAD 唤醒信号

该功能需要调用以下 API 使能(参考 3.7 节)：

```
void System_WakeUpPinEnable(uint8_t Pin_Num， uint8_t Polarity， uint8_t DebounceEn)
```

2. BT 中断

1) BT 处于 advertising 状态，且 advertising anchor 到来。
2) BT 处于 Connection 状态，且 connection anchor 到来。
3) 有 BT 事件发生， 如对端连接请求等。

3. RTC 中断

该功能需要调用以下 API 使能.

```
void RTC_SystemWakeupConfig(FunctionalState NewState)
```

4. SW Timer timeout 事件

2.3 DLPS 模式进入流程

DLPS 模式进入流程如下：

1. 关中断

2. 询问各个模块是否能进入 DLPS 模式

1) 某个模块如果希望进入 DLPS 前能够询问自己，就需要预先向 DLPS Framework 注册 CB。
2) 当 DLPS Framework 执行 CB 的时候，各个模块再根据自己的情况告知 DLPS Framework 是否允许进入 DLPS。
3) 只要有任意一个模块不允许进入 DLPS，就不会进入 DLPS。

3. 系统状态保存

如果满足进入 DLPS 的条件，系统开始保存必要的状态，在这个过程中，会执行 APP 向系统注册的 enter callback。在目前的 SDK 中，CPU，peripherals 以及用户自定义状态的保存是在 APP 注册的 DLPS_ENTER CB 中实现的。

4. 下指令进入 DLPS 模式

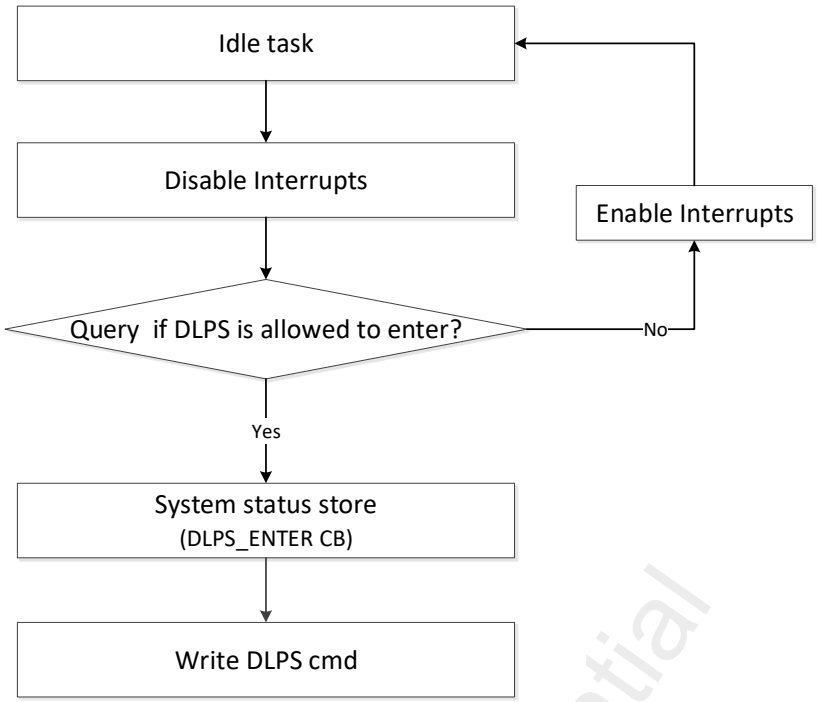

图 2-2 DLPS 进入流程

2.4 DLPS 模式退出流程

系统从 DLPS 模式醒来后，首先会恢复 power 和 clock， 随后 CPU 和 Peripherals 重新上电， 接下来 CPU 开始执行恢复流程，如图 2-3 所示：

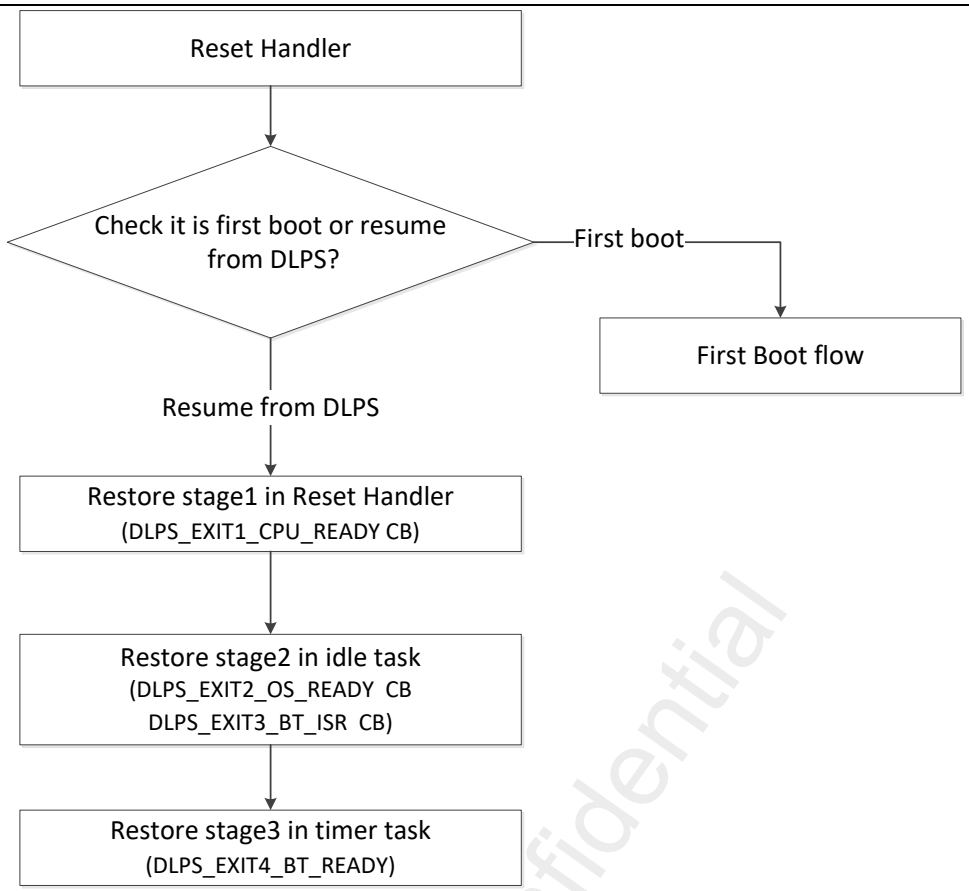

图 2-3 DLPS 恢复流程

1. Reset Handler

系统退出 DLPS 模式后，会触发 Reset Handler 异常。在 Reset Handler 中会检查重启的原因。如果是重新上电，系统会走重启流程。如果是从 DLPS 模式唤醒， 系统会走 DLPS 恢复流程。

2. Restore stage1 in Reset Handler

1) 从 DLPS 模式醒来后，系统会首先恢复 watch dog，clock 与 flash。
2) 开始执行 DLPS_EXIT1_CPU_READY CB（如果注册过的话）。注意，在这个 CB 中只能操作 watch dog 和 SWD 等特殊模块。
3) 开始恢复 Log，一些 peripheral 配置，FPU 和 MPU。
4) 跳转到 idle task。

3. Restore stage2 in Idle Task

1) 在 idle task 中会恢复 BLE 模块，随后执行 DLPS_EXIT2_OS_READY CB 和 DLPS_EXIT3_BT_ISR CB（如果注册过的话）。注意，只有 stack 可以注册这两个 CB。
2) BLE 模块恢复后，开始打开中断，OS 恢复调度。

4. Restore stage3 in Timer Task

DLPS 恢复的最后阶段是在 timer task 中完成的，在目前的 SDK 中，APP 会向系统注册

DLPS_EXIT4_BT_READY CB，这个 CB 会在 timer task 中执行。DLPS_EXIT4_BT_READY CB 中会完成 CPU，peripherals 及用户自定义状态的恢复。

注意： 如果系统是被 PAD 信号唤醒的，且系统使能了 System Interrupt， System ISR 会被触发。

3 硬件状态保存与恢复

3.1 CPU

系统进入 DLPS 后 CPU 会掉电，因此需要在进入 DLPS 前保存 NVIC 寄存器，并在退出 DLPS 后恢复 NVIC 寄存器。

3.2 PAD (AON)

PAD 在 DLPS 模式下不会掉电，因此不需要保存其状态。但是为了防止漏电，在进 DLPS 时需要对 PAD 做如下设定：

1. 系统没有使用到的 PAD，包括 package 没有出引脚的 PAD 必须设为 {SW mode，Input mode，Pull Down}。
2. 系统使用到的 PAD 必须设为{SW mode，Input mode，Pull Up/Pull Down}，Pull Up 还是 Pull Down 取决于外围电路。
3. 设置了唤醒功能的 PAD 需要设定为{SW mode，Input mode，Pull Up/Pull Down}，Pull Up/Pull Down 状态要与 wakeup 信号的极性相反。
4. 退出 DLPS 时要把 PAD 设置恢复成原来的状态。

3.3 外设

外设进 DLPS 时会掉电，所以进出 DLPS 时需要保存和恢复相关设定。退出 DLPS 时需要先重新使能模块并打开相应时钟，再恢复相关设定。

外设列表如下，具体可参见 SDK 相关代码：

1. Pinmux
2. I2C
3. TIME
4. QuadDecoder
5. IR
6. RTC

7. UART
8. ADC
9. SPI0~2
10. Keyscan
11. DMIC
12. GPIO
13. PWM0~3
14. GDMA0~6

3.4 外挂 Sensor

外挂 Sensor 在进出 DLPS 时的处理分两种情况：

1. 如果 Sensor 不掉电， 则不用恢复。
2. 如果 Sensor 掉电， 则需要注册 application callback 函数，并在其中恢复 sensor 设定 (重新初始化)。

3.5 状态保存流程

硬件状态保存流程如图 3-1 所示：

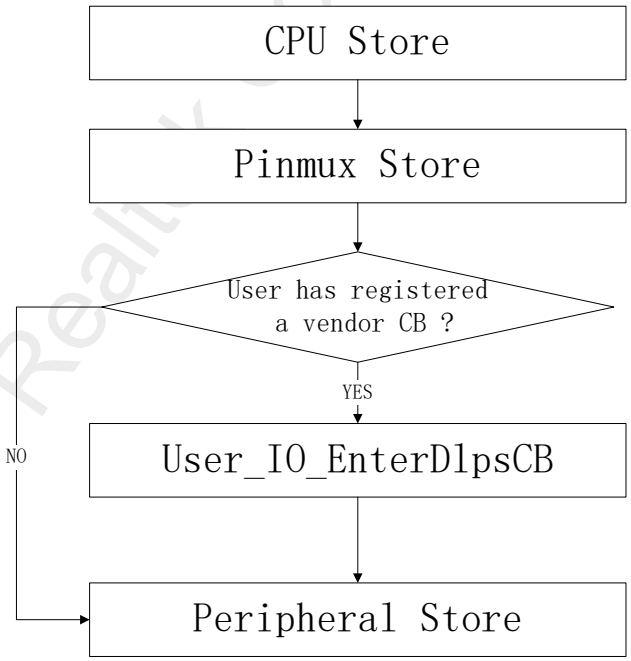

图 3-1 硬件状态保存流程

CPU， pinmux 和 peripherals 的状态会由系统自动保存。 PAD 状态如何设定取决于 APP， 因此需要在 APP 中调用 DLPS_IO_RegUserDlpsEnterCb 注册 vendor callback function，并在该 CB 中根据需要

设定 PAD 状态。 如果有外挂 sensors 需要在进 DLPS 前进行保存操作， 也应该放在 vendor callback 中实现。

3.6 恢复流程

硬件设定恢复流程如下：

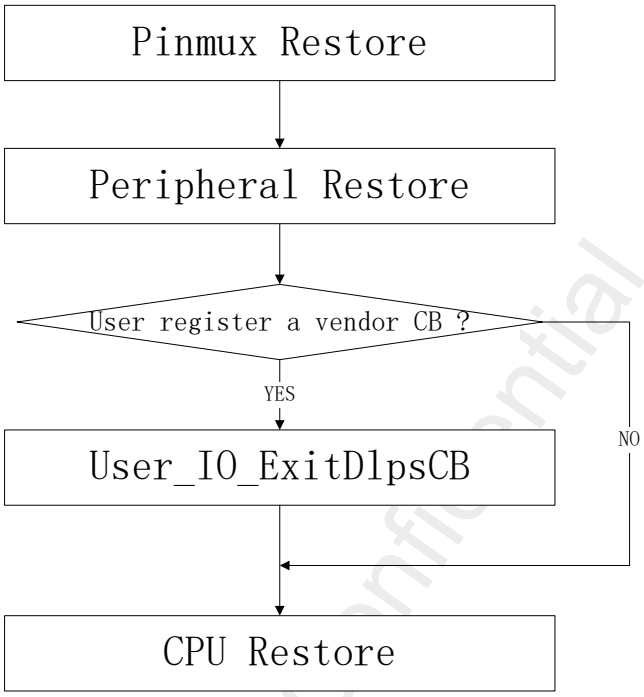

图 3-2 硬件设定恢复流程

退出 DLPS 时，系统会自动恢复 CPU， pinmux 和 peripherals。 出 DLPS 时，PAD 如何设定取决于 APP， 因此需要在 APP 中调用 DLPS_IO_RegUserDlpsExitCb 注册 vendor CB，在 vendor CB 里恢复 PAD 设定。如果有外部 sensor 在出 DLPS 后需要恢复的操作，也应该放在 vendor CB 里实现。

3.7 外设 DLPS 相关设定

APP project 会有独立的 board.h 文件，其中包含如下硬件相关的 DLPS 设定：

```
/* if use user define dlps enter/dlps exit callback function */
#define USE_USER_DEFINE_DLPS_EXIT_CB        1
#define USE_USER_DEFINE_DLPS_ENTER_CB       1



/* if use any peripherals below ,  #define it 1 else 0 */
#define USE_I2C0_DLPS            0
```

```
#define USE_I2C1_DLPS           0
#define USE_TIM_DLPS            0
#define USE_QDECODER_DLPS       0
#define USE_IR_DLPS             0
#define USE_RTC_DLPS            0
#define USE_UART_DLPS           1
#define USE_ADC_DLPS            1
#define USE_SPI0_DLPS           0
#define USE_SPI1_DLPS           0
#define USE_SPI2W_DLPS          0
#define USE_KEYSCAN_DLPS        0
#define USE_DMIC_DLPS           0
#define USE_GPIO_DLPS           0
#define USE_PWM0_DLPS           0
#define USE_PWM1_DLPS           0
#define USE_PWM2_DLPS           0
#define USE_PWM3_DLPS           0


#define USE_GDMACHANNEL0_DLPS       0
#define USE_GDMACHANNEL1_DLPS       0
#define USE_GDMACHANNEL2_DLPS       0
#define USE_GDMACHANNEL3_DLPS       0
#define USE_GDMACHANNEL4_DLPS       0
#define USE_GDMACHANNEL5_DLPS       0
#define USE_GDMACHANNEL6_DLPS       0
```

如果 APP 中使用了某个外设，且需要在进出 DLPS 时，系统能自动保存、恢复其状态，则需要将该外设对应的 USE_XXX_ DLPS 宏设置为 "1"。 APP 还需要在 PwrMgr_Init()中调用如下 API 来注册 IO DLPS CB，系统会在该 CB 里自动完成相关外设的保存和恢复：

```
DLPS_IO_Register();
```

上面的例子中，使能了 ADC 与 UART 的自动保存、恢复功能。

如果需要在进出 DLPS 时做一些 APP 自定义的操作，则需要设置以下两处：

1. 在 board.h 中将 USE_USER_DEFINE_DLPS_EXIT_CB 与 USE_USER_DEFINE_DLPS_ENTER_CB 配置为 1。

2. 在 APP 中调用如下 API 来注册并实现 vender callback 函数。

```
void DlpsExitCallback(void)
{
    //do something here
}


void DlpsEnterCallback(void)
{
    //do something here
}


DLPS_IO_RegUserDlpsExitCb(DlpsExitCallback);
DLPS_IO_RegUserDlpsEnterCb(DlpsEnterCallback);
```

在上面的例子中，在进入和退出 DLPS 的过程中会分别执行 DlpsEnterCallback()和 DlpsExitCallback()，APP 可以在这两个函数中完成自定义操作，如 PAD 设置、外围设备的操作等。

4 PAD 唤醒功能

4.1 Pad 唤醒

RTL8762C 的一些 PAD 具有 DLPS 唤醒功能，详情可参见相关硬件手册。可以调用以下 API 使能某个 PAD 的唤醒功能。当此 PAD 的电平与唤醒电平相同时，会将系统从 DLPS 状态唤醒。

```
void System_WakeUpPinEnable(uint8_t Pin_Num， uint8_t Polarity， uint8_t DebounceEn)
```

例如希望 P3_2 为高电平时唤醒系统，可以做以下配置：

```
System_WakeUp_Pin_Enable(P3_2， PAD_WAKEUP_POL_HIGH， 0);
```

调用 System_WakeUpPinEnable 时，将 DebounceEn 设置为 1 可以使能 wakeup debounce 功能。

调用 System_WakeUpDebounceTime 可以设置 debounce time：

```
void System_WakeUpDebounceTime(uint8_t time)
```

系统从 DLPS 醒来后， 可以调用 System_WakeUpInterruptValue 来查询是哪个 PAD 唤醒了系统：

```
uint8_t System_WakeUpInterruptValue(uint8_t Pin_Num)
```

4.2 Keyscan 应用

4.2.1 按键检测

如图 4-1 所示，Keyscan 的 debounce 机制会导致 keyscan 中断丢失。

1. 按键按下，将系统从 DLPS 状态唤醒，开始恢复流程；
2. keyscan模块恢复完成，开始debounce，会在时刻5结束debounce， keyscan中断在debounce结束后才会产生；
3. keyscan debounce过程；
4. 在keyscan debounce期间，系统没有需要处理的事件，会进入idle task，在keyscan debounce结束前，再次进入DLPS模式；
5. Keyscan debounce在此时结束；
6. 系统再次进入DLPS时，如果按键仍然被按下，系统会再次被唤醒，重复2~5的过程，直到按键松开，但始终无法进入keyscan中断。

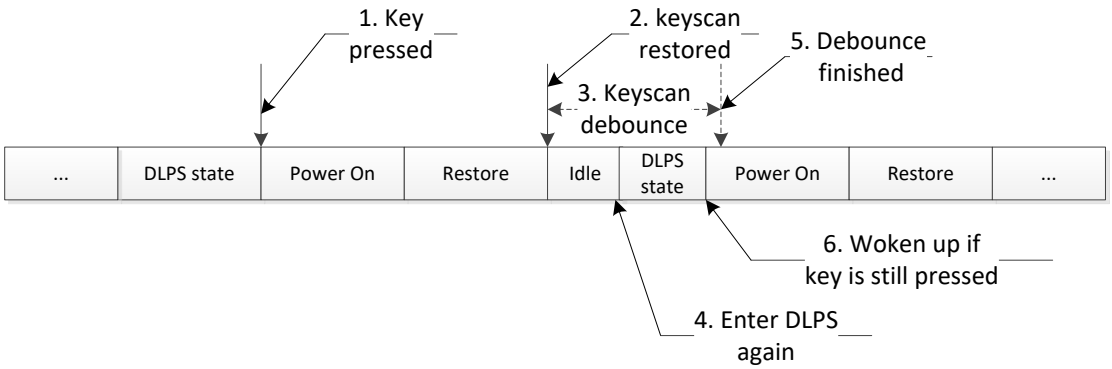

图 4-1 Keyscan 的 debounce 机制导致中断丢失

如图 4-2，可以利用 System ISR 来解决该问题：

1. 按键按下，将系统从 DLPS 状态唤醒，系统开始恢复流程；
2. keyscan模块恢复完成，开始debounce，会在时刻5结束debounce，keyscan中断在debounce结束后才会产生；
3. System中断产生，在System ISR中禁止系统进DLPS；
4. Keyscan debounce过程；
5. 系统没有事件要处理，进入idle task，但无法进入DLPS模式，因为DLPS在System ISR中已被禁止；
6. Keyscan debounce结束，keyscan中断产生，并被系统处理。

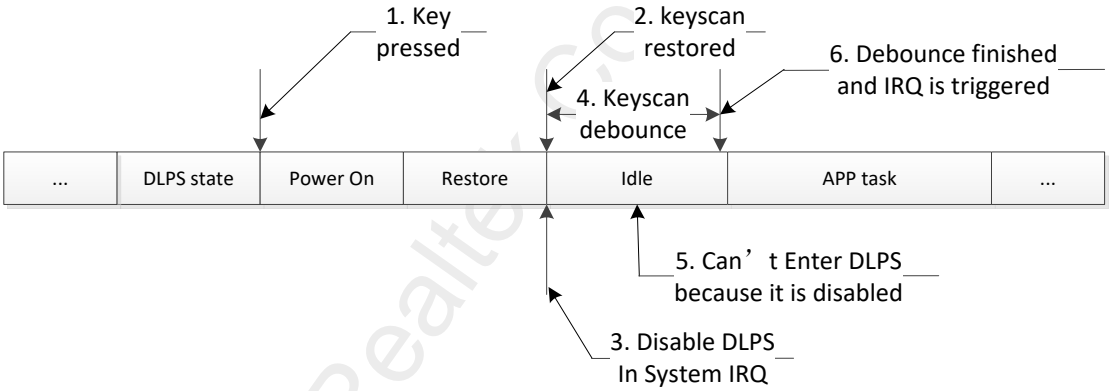

图 4-2 利用 System ISR 检测 keyscan 中断

另一种更好的策略是利用 PAD wakeup debounce 功能，如图图 4-3：

1. 按键按下，开始 wakeup debounce；
2. debounce 结束后，系统从 DLPS 状态被唤醒，开始恢复流程；
3. Keyscan模块恢复完成；
4. System中断产生，在该中断中禁止keyscan debounce，随后即会产生keyscan中断；

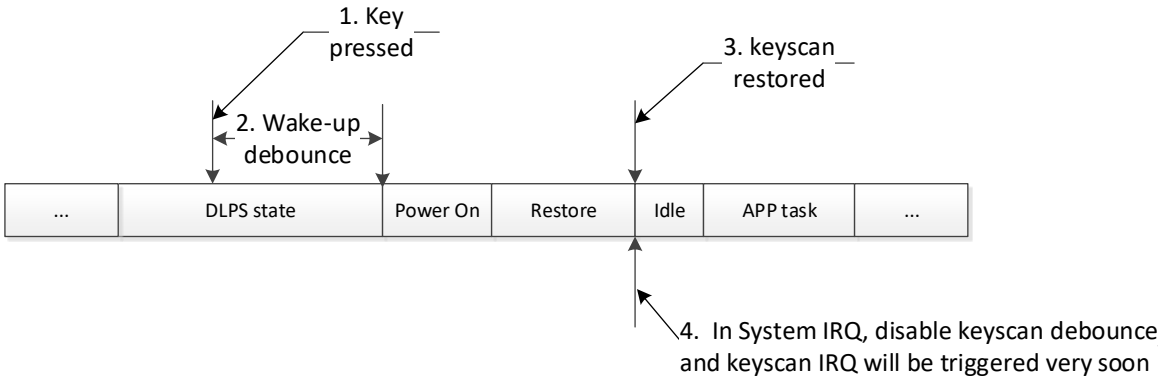

图 4-3 利用 PAD wakeup debounce 功能检测 keyscan 中断

利用 wakeup debounce 可以缩短 active 的时间，从而达到降低系统功耗的效果。

4.2.2 PAD 设置

在进 DLPS 前，Keyscan 的列必须设置为：

PAD_Config (Px_x， PAD_SW_MODE， PAD_IS_PWRON， PAD_PULL_NONE， PAD_OUT_ENABLE， PAD_OUT_LOW);

从 DLPS 醒来后， Keyscan 的列必须设置为：

PAD_Config (Px_ x， PAD_ PINMUX _MODE， PAD_IS_PWRON， PAD_PULL_NONE， PAD_OUT_ENABLE， PAD_OUT_LOW);

可以在 vendor callback 中完成上述设置。Vendor callback 可以在 PwrMgr_Init() 中调用 DLPS_IO_RegUserDlpsEnterCb()和 DLPS_IO_RegUserDlpsExitCb ()来注册。

4.3 GPIO 应用

当 GPIO 被配置为 edge trigger 且使能了 debounce 功能时，会存在与 keyscan 类似的问题，此时可以使用类似的机制来解决问题。

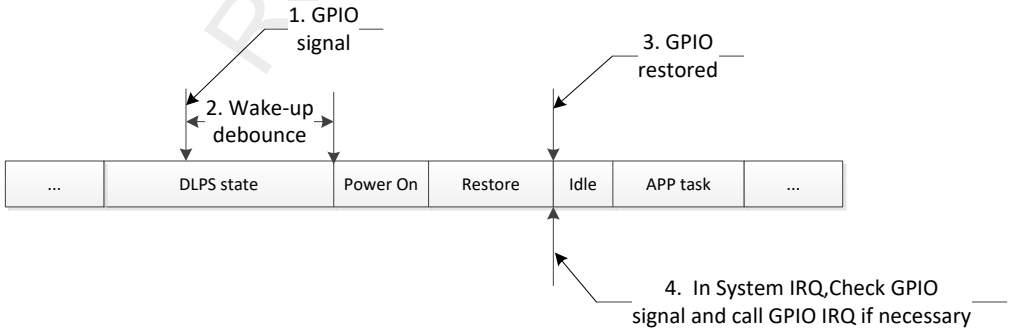

图 4-4 利用 PAD wakeup debounce 功能检测 GPIO 中断

5 蓝牙相关的 DLPS 应用场景

5.1 Non-Link mode

当没有连接时，BT 存在三种状态：Standby State，Advertising State 和 Scanning State。

1. 在 Standby State，不会收发数据，蓝牙不会将系统从 DLPS 状态唤醒。
2. 在 Advertising State，若 Adv_Interval * 0.625ms >= 20ms，则允许进入 DLPS，否则不允许。若 Advertising Type 为 Direct Advertising (High duty cycle)，则不允许进入 DLPS。
3. 在 Scanning State，若 (Scan Interval – Scan Window) * 0.625ms >= 15ms，则允许进入 DLPS，否则不允许。

5.2 Link mode

Link mode 下有两种角色：Slave Role 和 Master Role。

1. 在 Master Role 时，若 Connection Interval * 1.25ms > 12.5ms，允许进入 DLPS。
2. 在 Slave Role，若 Connection Interval* (1+ Slave Latency) *1.25ms > 12.5ms，允许进入 DLPS。

6 DLPS Mode API

6.1 lps_mode_set()

原型：

void lps_mode_set(LPSMode mode)。

作用：

使能/禁止 DLPS 模式。

参数：

功耗模式，LPS_MODE 枚举值。

1) LPM_POWERDOWN_MODE：只允许 PAD 唤醒。
2) LPM_HIBERNATE_MODE：只允许 PAD 和 RTC 唤醒
3) LPM_DLPS_MODE：DLPS 模式。
4) LPM_ACTIVE_MODE：系统不会进入任何低功耗模式。

6.2 dlps_hw_control_cb_reg()

原型：

BOOL dlps_hw_control_cb_reg (DLPSHWControlFunc func, DLPS_STATE DLPSState)

作用：

向系统注册 HW Control callback，并根据 dlps_state 在进出 DLPS 的适当阶段回调。

参数：

1） DLPSHWControlFunc func：回调函数。

2） DLPSState：DLPS_STATE 枚举值，指定 callback 回调时机。

（1） DLPS_ENTER：进 DLPS 时回调

（2） DLPS_EXIT1_CPU_READY：出 DLPS 的早期回调

（3） DLPS_EXIT4_BT_READY：出 DLPS 的最后阶段回调

示例：

```
//当系统进入 DLPS 时回调 DLPS_IO_EnterDlpsCb;
dlps_hw_control_cb_reg (DLPS_IO_EnterDlpsCb, DLPS_ENTER);
//当系统退出 DLPS 时回调 Call DLPS_IO_ExitDlpsCb;
dlps_hw_control_cb_reg(DLPS_IO_ExitDlpsCb, DLPS_EXIT4_BT_READY);
```

6.3 dlps_hw_control_cb_unreg()

原型：

VOID dlps_hw_control_cb_unreg (DLPSInterruptControlFunc func);

作用：

请求系统取消对 callback 的回调。调用该 API 后，系统在进出 DLPS 时将不再回调相应的 callback。

6.4 dlps_check_cb_reg()

原型：

BOOL dlps_check_cb_reg (DLPSEnterCheckFunc func);

作用：

向系统注册查询 callback，在系统试图进入 DLPS 前会回调该 callback，根据 callback 的返回值决定是否允许进入 DLPS 状态。有任何一个查询 callback 返回 FALSE，系统都不会进入 DLPS 状态。

参数：

func：进入 DLPS 前的查询 callback。

6.5 DLPS_IO_RegUserDlpsEnterCb()

原型：

__STATIC_INLINE void DLPS_IO_RegUserDlpsEnterCb(DLPS_IO_EnterDlpsCB func);

作用：

用于注册进 DLPS 时的 vendor callback; APP 自定义的 IO 保存动作需要在 vendor callback 中实现。

参数：

func：进入 DLPS 的 vendor callback。

6.6 DLPS_IO_RegUserDlpsExitCb

原型：

__STATIC_INLINE void DLPS_IO_RegUserDlpsExitCb(DLPS_IO_ExitDlpsCB func);

作用：

用于注册出 DLPS 时的 vendor callback，APP 自定义的 IO 恢复动作需要在 vendor callback 中实现。

参数：

DLPS_IO_ExitDlpsCB func：退出 DLPS 的 vendor callback。

6.7 System_WakeUpPinEnable

原型：

void System_WakeUpPinEnable(uint8_t Pin_Num, uint8_t Polarity, uint8_t DebounceEn);

作用：

用于配置 PAD 的 wakeup 功能

参数：

1） Pin_Num：可配置为 ADC_0 到 P4_1，详情参见 rtl876x.h 中"Pin_Number"部分

2） Polarity：唤醒极性：

（1） PAD_WAKEUP_POL_HIGH：高电平唤醒

（2） PAD_WAKEUP_POL_LOW：低电平唤醒

3） DebounceEn：使能/禁用 wakeup debounce

6.8 System_WakeUpDebounceTime

原型：

void System_WakeUpDebounceTime(uint8_t time);

作用：

用于设置 debounce 时间

参数：

time：debounce 时间，单位：ms

6.9 System_WakeUpInterruptValue

原型：

uint8_t System_WakeUpInterruptValue(uint8_t Pin_Num);

作用：

用于查询某个 PAD 是否是唤醒系统的 PAD，若返回 true，则该 PAD 即为唤醒系统的 PAD

参数：

Pin_Num：待查询的 PAD