RTL8762E Deep Low Power State User Guide

V1.0

2022/05/09

Revision History

Date	Version	Comments	Author	Reviewer
2022/05/09	V1.0		Rui	Lory

Contents

Revision History 2
Contents 3
Figure List 6
1 DLPS Mode Overview 7
1.1 Features and Restrictions 7
1.2 Principle 7
2 Enter/exit from DLPS 8
2.1 Conditions for Entering DLPS Mode 8
2.2 Wakeup Events 8
2.3 DLPS Mode Entry Flow 9
2.4 DLPS Exit Flow 10
3 Hardware Status Storage and Recovery 11
3.1 CPU NVIC 11
3.2 PAD (AON) 11
3.3 Peripherals 11
3.4 External Sensor 11
3.5 Storage Flow 12
3.6 Recovery Flow 12
3.7 Usage of DLPS Settings about Peripherals 13
4 Pad Wakeup Features 16
4.1 Pad Wakeup 16
4.2 Keyscan 16
4.2.1 Key Trigger Detection 16
4.2.2 PAD Settings 18
4.3 GPIO 18
5 DLPS Scenarios about Bluetooth 20
5.1 Non-Link mode 20
5.2 Link mode 20

6 DLPS Mode API........ 21
6.1 lps_mode_set........ 21
6.2 dlps_check_cb_reg........ 21
6.3 DLPS_IORegUserDlpsEnterCb........ 21
6.4 DLPS_IORegUserDlpsExitCb........ 22
6.5 System_WakeUpPinEnable........ 22
6.6 System_WakeUpDebounceTime........ 22
6.7 System_WakeUpInterruptValue........ 23

Table List

Table 6-1 1.1 lps_mode_set 21
Table 6-2 dlps_check_cb_reg 21
Table 6-3 DLPS_IORegUserDlpsEnterC 21
Table 6-4 DLPS_IORegUserDlpsExitCb 22
Table 6-5 System_WakeUpPinEnable 22
Table 6-6 System_WakeUpDebounceTime 22
Table 6-7 System_WakeUpInterruptValue 23

Figure List

Figure 3-1 Hardware Store Flow 12
Figure 3-2 Hardware Restore Flow 13
Figure 4-1 keyscan interrupt missing because of debounce mechanism 17
Figure 4-2 keyscan interrupt detection with System IRQ........ 17
Figure 4-3 keyscan interrupt detection with wakeup debounce........ 18
Figure 4-4 GPIO interrupt detection with wakeup debounce 19

1 DLPS Mode Overview

RTL8762E supports three power modes: Power Down mode, DLPS (Deep Low Power State) mode and Active mode. This document describes DLPS mode in detail.

1.1 Features and Restrictions

RTL8762E DLPS mode has the following features and restrictions:

1. System power consumption is 3uA@3V.
2. System recovers from DLPS in about 2ms and enters DLPS within 1ms.
3. System can be woken up from DLPS by PAD and LPC signals.
4. System can be woken up from DLPS by BLE broadcast and connection event periodically.
5. As power-off of CPU will cause disconnection of SWD, DLPS mode must be disabled during online debugging with Keil.

1.2 Principle

Most of the time, the system is idle and components such as Clock, CPU and Peripherals can be powered off to reduce system power consumption. When any event is to be handled, the system will be woken up from DLPS mode, Clock, CPU, and Peripherals will be re-powered on and recovered to the previous status before entering DLPS, and then the wake-up events will be responded.

2 Enter/Exit from DLPS

2.1 Conditions for Entering DLPS Mode

The system can only enter DLPS mode when the following conditions are met the same time.

1. Idle task is running, while all the rest tasks are in blocked or suspended status, and no ISR occurs.
2. All the DLPS Check callback functions return true, which are registered by BT Stack, peripherals and application.
3. SW timer period or task delay period >= 20ms.
4. BT is in one of the states below and corresponding parameters meet the requirements.
   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
5. Stack, peripherals, and application shall be checked in idle task. If all entry requests are permitted, system will perform the store flow and enter DLPS mode.
6. If BT module is allowed to enter into DLPS, while platform not, BT module will enter into low power mode, while platform module keeps active. If BT module is not allowed, System will still keep active state.

2.2 Wakeup Events

The system can be woken up by one of the following events to exit from DLPS mode.

1. PAD wakeup signal (This function can be enabled by calling the following API)

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

2. BT interrupt (which will be generated by the following scenarios)
   1) Advertising anchor arrives when BT is in advertising state.
   2) Connection event anchor arrives when BT connection is established.
   3) Any BT event occurs, such as Remote Connection Request, Receiving data.

3. RTC interrupt (This function can be enabled by calling the following API)

```
RTC_SystemWakeupConfig(ENABLE);
```

Note: As system handle RTC wake from DLPS, RTC IRQ may be delayed processing. So it is recommended to register DLPS check callback function, and calculate next wake up time in callback function. After platform wake up from DLPS, RTC IRQ is handled.

1) comment out RTC_WKConfig(RTC_COMP_WK_INDEX, ENABLE) to disable RTC wake up.
2) calculate the next wake up time for DLPS.

```
uint32_t RTC_tick; // unit: 32.15us

bool RTC_Check_GT(uint32_t *next_wakeup_time)   //unit 31.25us
{
    uint32_t wakeup_count = RTC_GetCompValue(RTC_COMP_INDEX)  - RTC_GetCounter() ;
    if (wakeup_count > 0)
    {
        *next_wakeup_time = wakeup_count * RTC_tick;
        return true;
    }
    else
    {
        return false;
    }
```

3) register DLPS check callback.

```
RTC_tick = (RTC_PRESCALER_VALUE + 1) ;
    if (false == dlps_check_cb_reg(RTC_Check_GT) )
    {
        DBG_DIRECT("Error: dlps_check_cb_reg(RTC_Check_GT)  failed!\n") ;
    }
```

4. **SW Timer timeout or task delay**
5. **LPC interrupt** (This function can be enabled by calling the following API)

```
LPC_WKCmd(ENABLE) ;
RTC_SystemWakeupConfig(ENABLE) ;
```

2.3 DLPS Mode Entry Flow

DLPS mode enter flow is shown below.

1. If a module needs to be inquired before entering DLPS mode, it should register callback function to DLPS Framework first. When DLPS Framework calls the callback functions, each module will inform DLPS

Framework to or not to enter DLPS based on the return value of callback function. BT module is based on platform module. Only when BT module enter DLPS, will system check if platform module needs to enter into DLPS.

2. If all conditions are satisfied and DLPS is allowed to be entered, the system first check BT module. If enabled, then store BT status and make BT module enter into sleep mode. Platform module is checked next, only when platform module is allowed to enter into sleep mode, platform status is stored and DLPS state is finally entered.
   1) Inquire BT module whether DLPS mode is allowed to enter.
   2) BT module store status and enter DLPS.
   3) Inquire platform module whether DLPS mode is allowed to enter.
   4) Platform module store status and enter DLPS.
   5) Send the command to enter DLPS mode.

2.4 DLPS Exit Flow

After system is woken up from DLPS mode, power and clock will be recovered firstly, CPU and Peripherals will be powered up secondly, and then CPU starts restoring flow. Only when platform exit from DLPS completely, will system check if BT module need to be waken up fromDLPS.

1. **Reset Handler**

   In reset handler, the reset reason will be detected. If system is powered on, it will perform First Boot flow. If system exits from DLPS mode, it will perform DLPS recovery flow.

2. **System module exit and restore from DLPS**
   1) Platform module exit from DLPS first, and restore status.
   2) Check if BT event wakes up DLPS. If so, BT module exit from DLPS and restore status. If not, BT module will keep in low power mode status.

3. **System exit from DLPS completely**

   In current SDK, the last stage of the exit process will be executed in timer task. The restore of CPU, peripherals and user defined status is implemented before system exit from DLPS completely.

Note: If system is woken up by PAD signal and System Interrupt is enabled, System ISR will be triggered.

3 Hardware Status Storage and Recovery

3.1 CPU NVIC

CPU will be powered off upon entry to DLPS mode, so NVIC registers must be saved before entering DLPS mode and then recovered after exiting from DLPS mode.

3.2 PAD (AON)

PAD will not be powered off upon entry to DLPS, so storage is not required. However to prevent electric leakage, PAD must be set as below when entering DLPS:

1. Pins which have not been used, including pins disabled in IC, must be set as (SW mode, Input mode, Pull Down).
2. Pins which have been used should be set as (SW mode, Input mode, Pull Up/Pull Down), whether a pin should be pulled up or down depends on the external circuit.
3. The pins which have wake-up ability should be set as (SW mode, Input mode, Pull Up/Pull Down), the pull mode of a wake-up pin should be opposite to its wake-up polarity.
4. The pins should be recovered to the original settings after exiting from DLPS.

3.3 Peripherals

Peripherals will be powered off upon entry to DLPS, so the related settings must be saved before entering DLPS and recovered after exiting from DLPS. Before recovering peripheral settings, the peripheral module should be enabled and the clock should be started firstly.

3.4 External Sensor

When an external sensor enters or exits from DLPS, processing is performed in two cases.

1. If Sensor is not power off, it does not need to be recovered.
2. If Sensor is powered off, application callback function must be registered and sensor setting will be recovered in the function (re-initialize).

3.5 Storage Flow

Hardware status storage flow is shown in Figure 3-1

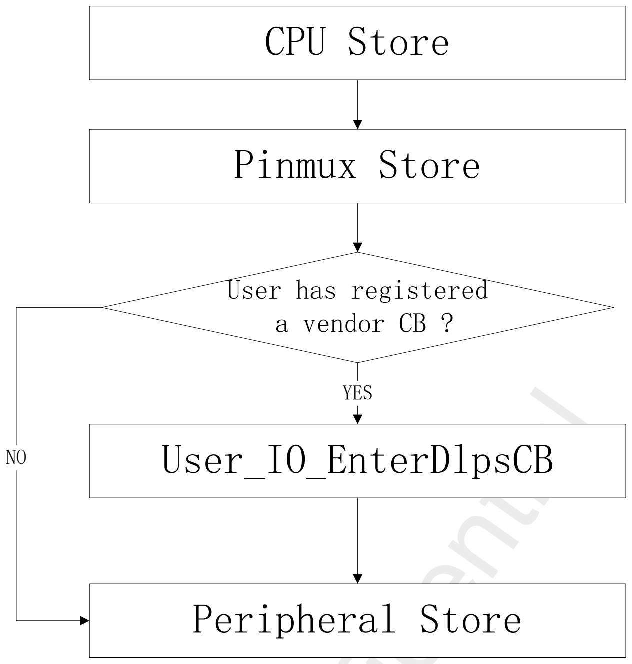

Figure 3-1 Hardware Store Flow

The store of CPU, pinmux and peripherals has already been implemented by system. Because pin settings during entering DLPS vary depending on the application, related pin settings are handled in the vendor callback function which is registered through DLPS_IORegUserDlpsEnterCb by the application. If external sensors are used and need to be handled, the implementation should also be placed in vendor callback.

3.6 Recovery Flow

A flow of recovering hardware settings is shown below in Figure 3-2.

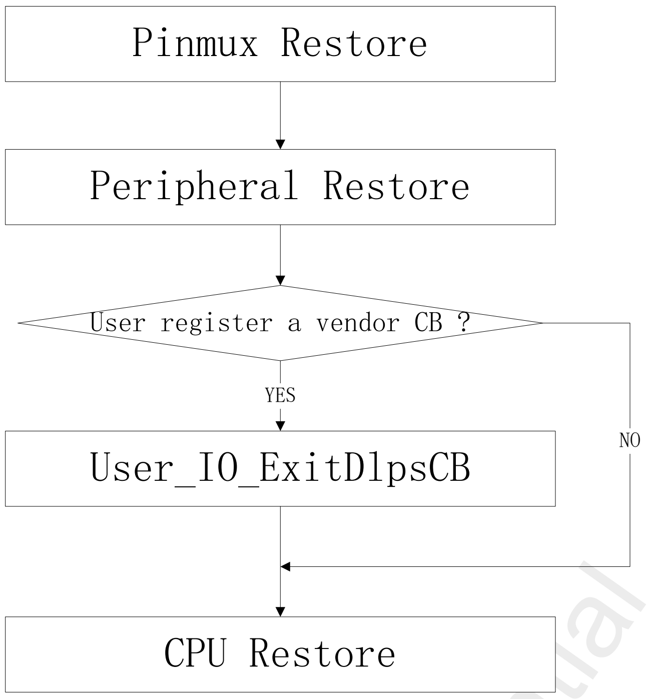

Figure 3-2 Hardware Restore Flow

The restore of CPU, pinmux and peripherals has already been implemented by system. Because pin settings during exiting from DLPS vary depending on the application, related pin setting is handled in the callback function which is registered through DLPS_IORegUserDlpsExitCb by the application. If external sensors are used and need to be handled, the implementation should also be placed in vendor callback.

3.7 Usage of DLPS Settings about Peripherals

Each application has a copy of board.h file, which contains the following DLPS settings for hardware.

```
/* 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 */
#define USE_ADC_DLPS              0
#define USE_CTC_DLPS              0
#define USE_GPIO_DLPS             1
#define USE_I2C0_DLPS             0
#define USE_I2C1_DLPS             0
#if  (ROM_WATCH_DOG_ENABLE == 1)
#define USE_TIM_DLPS              1 //must be 1 if enable watch dog
#else
#define USE_TIM_DLPS              0
```

```
#endif
#define USE_IR_DLPS             0
#define USE_KEYSCAN_DLPS        0
#define USE_QDECODER_DLPS       0
#define USE_SPI0_DLPS           0
#define USE_SPI1_DLPS           0
#define USE_SPI2W_DLPS          0
#define USE_UART0_DLPS          0
#define USE_UART1_DLPS          0
#define USE_I2S0_DLPS           0
#define USE_ENHTIM_DLPS         0
#define USE_CODEC_DLPS          0
```

If DLPS function of a peripheral needs to be enabled, the corresponding USE_XXX_ DLPS macro should be defined as "1". If any peripheral is used, the following API should be called in PwrMgr_Init() of the application to register peripheral DLPS function:

```
DLPS_IORegister() ;
```

Through the two steps above, storage and recovery functions of DLPS for the peripheral will be activated. In this case, DLPS function is enabled for the two peripherals: ADC and UART.

If some operations need to be performed during entering or exiting from DLPS, the two steps below should be followed.

1. Define macro USE_USER_DEFINE_DLPS_EXIT_CB or USE_USER_DEFINE_DLPS_ENTER_CB as ‘1’ in board.h.
2. Call the following API in application to register and implement callback function.

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

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

DLPS_IORegUserDlpsExitCb(DlpsExitCallback) ;
DLPS_IORegUserDlpsEnterCb(DlpsEnterCallback) ;
```

In the case above, DlpsEnterCallback() and DlpsExitCallback() will be executed respectively during entering and exiting from DLPS, and the application can implement some operations in the two function, such as PAD setting, or operation on peripherals.

4 Pad Wakeup Features

4.1 Pad Wakeup

PAD have the function to wake up the system from DLPS mode. Developers can refer to related hardware manual to get more details. Developers can call the following API to enable wakeup function of a pin. When polarity of a signal on the pin is the same as the wakeup level, the system will be woken up, and then recovered from DLPS mode to Active mode.

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

For example, if P3_2 is expected to wake up system from DLPS mode when signal on the pin becomes high level, the following configuration should be set.

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

Wakeup debounce could be enabled by calling System_WakeUpPinEnable and set DebounceEn to 1:
The debounce time could be configured by calling System_WakeUpDebounceTime:

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

After system is woken up, System_WakeUpInterruptValue could be called in system handler to query which pin has woken up the system:

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

4.2 Keyscan

4.2.1 Key Trigger Detection

The debounce mechanism will cause keyscan interrupt missing as is shown in Figure 4-1.

1. A key is pressed, the system is woken up from DLPS, and begins the recovery process.
2. Keyscan recovery is completed, keyscan begins debounce, and debounce is expected to be completed at 5, keyscan interrupt will not be triggered until debouncing duration expires.
3. Keyscan debouncing duration.

4. There is nothing to do, so idle task is entered, and DLPS mode is entered before keycan debounce is completed.
5. Keyscan debounce is expected to be completed at this point.
6. If key is still pressed, system will be woken up from DLPS mode again, 2~5 will be repeated until the key is released, but keycan interrupt can't be detected by the system in the whole process.

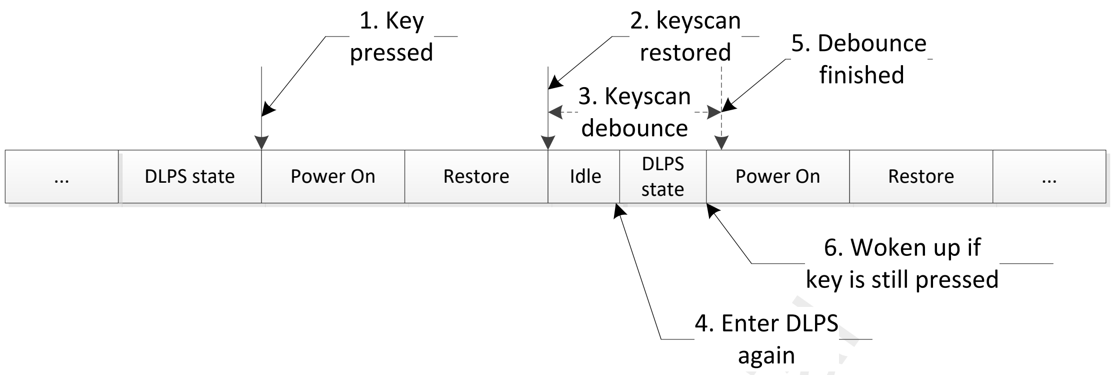

Figure 4-1 keyscan interrupt missing because of debounce mechanism

System ISR can be used to fix this issue, as is depicted in Figure 4-2:

1. A key is pressed, the system is woken up from DLPS, and begins the recovery process.
2. Keyscan recovery is completed, keyscan begins debounce, and debounce is expected to be completed at 5, keyscan interrupt will not be triggered until debouncing duration expires.
3. System interrupt is triggered, and DLPS will be disabled in the ISR.
4. Keyscan debouncing duration.
5. There is nothing to do, so idle task is entered, but DLPS mode can't be entered because it is disabled in System ISR.
6. Keyscan debounce is completed and keyscan IRQ is triggered and detected by the system, which will be handled in APP task.

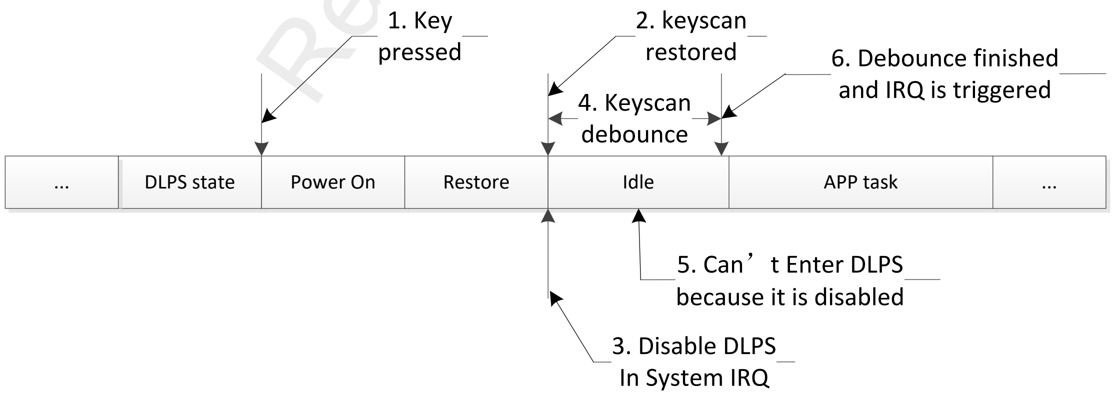
1. Key pressed
2. keyscan restored
4. Keyscan debounce
6. Debounce finished and IRQ is triggered
...
DLPS state
Power On
Restore
Idle
APP task
...
5. Can't Enter DLPS because it is disabled
3. Disable DLPS In System IRQ

Figure 4-2 keyscan interrupt detection with System IRQ

A better strategy is utilizing debounce wakeup feature, as is depicted in Figure 4-3:

1. A key is pressed, and wakeup debounce begins.
2. The system is woken up from DLPS after debounce ends, and begins the recovery process.
3. Keyscan recovery is completed before System IRQ occurs.
4. System IRQ occurs, keyscan debounce is disabled and keyscan IRQ will be triggered very soon.

Note: This strategy could decrease the system power consumption, since the active period is shortened.

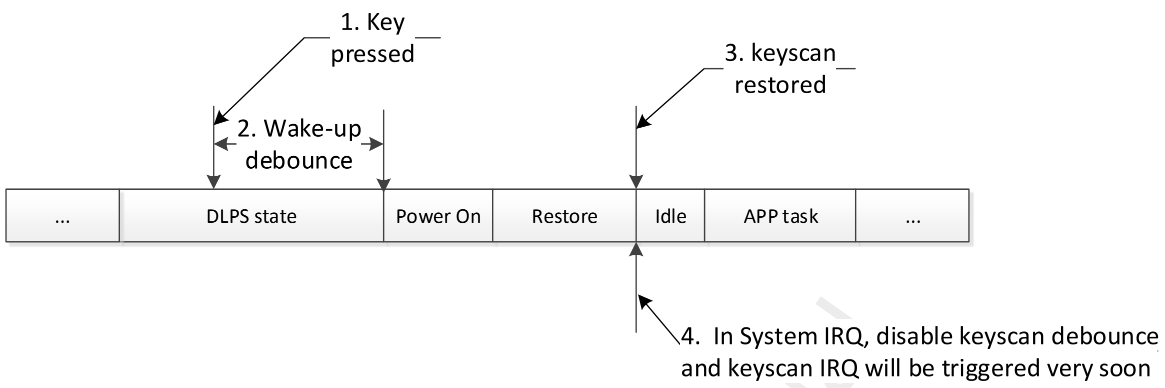

Figure 4-3 keyscan interrupt detection with wakeup debounce

4.2.2 PAD Settings

Before entering DLPS mode, Keyscan column must be set as:

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

After exiting from DLPS, Keyscan column must be set as:

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

DLPS_IORegUserDlpsEnterCb() and DLPS_IORegUserDlpsExitCb() can be used in PwrMgr_Init() to register the corresponding callback function.

4.3 GPIO

When GPIO is configured as edge trigger with debounce, there exists the same issue with keyscan, and the same mechanism with System IRQ should be applied.

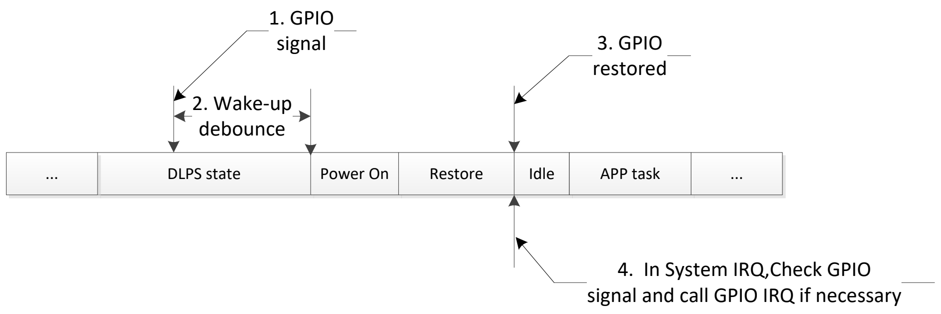

Figure 4-4 GPIO interrupt detection with wakeup debounce

5 DLPS Scenarios about Bluetooth

5.1 Non-Link mode

When no connection exists, three states are available: Standby State, Advertising State, and Scanning State.

1. In Standby State, no data is sent or received and no Bluetooth-related event wakes the system up from DLPS mode.
2. In Advertising State, if Adv_Interval * 0.625ms >= 20ms, then entering DLPS is permitted, otherwise, it is not permitted. If Advertising Type is Direct Advertising (High duty cycle), then entering DLPS is not permitted.
3. In Scanning State, if (Scan Interval – Scan Window) * 0.625ms >= 15ms, then entering DLPS mode is permitted, otherwise it is not permitted.

5.2 Link mode

In Link mode two roles are available: Slave Role and Master Role.

1. In Master Role, when Connection Interval * 1.25ms > 12.5ms, entering DLPS mode is permitted.
2. In Slave Role, when Connection Interval* (1+ Slave Latency) *1.25ms > 12.5ms, entering DLPS mode is permitted.

6 DLPS Mode API

6.1 lps_mode_set

Table 6-1 1.1 lps_mode_set

Prototype	**void lps_mode_set(PlatformPowerMode mode);**
Description	Enable/Disable LPS Mode
Parameters	Mode, PlatformPowerMode enumeration value 1. PLATFORM_POWERDOWN: can be woken up only by PAD and LPC, pad wake up debounce must be disabled in power down mode 2. PLATFORM_DLPS_PFM: DLPS mode 3. PLATFORM_ACTIVE: system will never enter DLPS mode or any low power mode

6.2 dlps_check_cb_reg

Table 6-2 dlps_check_cb_reg

Prototype	**BOOL dlps_check_cb_reg (DLPSEnterCheckFunc func) ;**
Description	Register inquiry callback function to DLPS Framework, and call this function each time before entering DLPS to decide whether DLPS is allowed to enter. DLPS will be disallowed if any inquiry callback function returns FALSE
Parameters	DLPSEnterCheckFunc func: Inquiry callback function

6.3 DLPS_IORegUserDlpsEnterCb

Table 6-3 DLPS_IORegUserDlpsEnterC

Prototype	**__STATIC_INLINE void DLPS_IORegUserDlpsEnterCb(DLPS_IO_EnterDlpsCB func);**
Description	DLPS_IO_EnterDlpsCb will be always registered in application sample project, and the callback function is used to save generic peripheral registers when entering DLPS mode and to store special peripheral settings related to application. App-specific peripherals storage actions need to be encapsulated into function and registered through this API
Parameters	DLPS_IO_EnterDlpsCB func: App-specific peripheral storage function

Note: Don’t do time-costing operation in DLPS enter callback, or DLPS wake up may be disturbed. As OS schedule and interrupt are all turned off, it’s not recommended to use OS API in DLPS enter callback function.

6.4 DLPS_IORegUserDlpsExitCb

Table 6-4 DLPS_IORegUserDlpsExitCb

Prototype	**__STATIC_INLINE void DLPS_IORegUserDlpsExitCb(DLPS_IO_ExitDlpsCB func);**
Description	DLPS_IO_ExitDlpsCb will always be registered in application sample project, and the callback function is used to restore generic peripheral registers when exiting from DLPS and to process App-specific peripheral recovery. App-specific peripheral recovery actions need to be encapsulated into function and registered through this API
Parameters	DLPS_IO_ExitDlpsCB func: App-specific peripheral recovery function

6.5 System_WakeUpPinEnable

Table 6-5 System_WakeUpPinEnable

Prototype	**void System_WakeUpPinEnable(uint8_t Pin_Num, uint8_t Polarity, uint8_t DebounceEn);**
Description	This API is used to configure wakeup pin
Parameters	1. Pin_Num: wakeup pin number, from ADC_0 to P4_1, please refer to rtl876x.h "Pin_Number" part 2. Polarity: wakeup polarity 1) PAD_WAKEUP_POL_HIGH: use high level wakeup 2) PAD_WAKEUP_POL_LOW: use low level wakeup 3. DebounceEn: enable/disable wakeup debounce

6.6 System_WakeUpDebounceTime

Table 6-6 System_WakeUpDebounceTime

Prototype	**void System_WakeUpDebounceTime(uint8_t time);**
Description	This API is used to set debounce time
Parameters	time: debounce time in ms

6.7 System_WakeUpInterruptValue

Table 6-7 System_WakeUpInterruptValue

Prototype	**uint8_t System_WakeUpInterruptValue(uint8_t Pin_Num);**
Description	This API is used to query which pin has woken up the system
Parameters	Pin_Num: wakeup pin number