CYW20819, CYW20820 low-power design guidelines

About this document

Scope and purpose

AN225270 describes how to use the power modes in AIROC™ CYW20819 and AIROC™ CYW20820 Bluetooth® & Bluetooth® LE system on chip devices to optimize power consumption. Major topics include the low-power modes in these devices, and power management techniques using those modes. It will provide tips and tricks to minimize the current consumption to increase battery life.

Intended audience

This application note is intended for users who want to optimize their applications for low power on CYW20819 or CYW20820 devices.

Table of contents

1 Introduction

Bluetooth® Low Energy devices such as heart-rate monitors are typically battery-operated. A long battery life is a key requirement for such devices. CYW20819 and CYW20820 give the flexibility of high performance and low power consumption using various low-power modes. This application note shows how to implement a low-power solution using the CYW20819 and CYW20820 devices. Later sections of the document will reference only the CYW20819 device but it is applicable for CYW20820 device as well.

CYW20819 is a Bluetooth® 5.2-compliant, stand-alone baseband processor with an integrated 2.4-GHz transceiver with support for both Bluetooth® LE and Classic Bluetooth®. The device is intended for use in audio, IoT, sensors (medical, home, security, and so on) and human interface device (HID) applications. Manufactured using an advanced 40-nm CMOS low-power fabrication process, CYW20819 employs a high level of integration to reduce external components, thereby minimizing system footprint and costs. Visit the product pages for datasheets, application notes, and development resources:

- AIROC™ CYW20819 Bluetooth® & Bluetooth® LE system on chip
- AIROC™ CYW20820 Bluetooth® & Bluetooth® LE system on chip

This application note requires a basic understanding of the CYW20819 device architecture and ability to develop an application using ModusToolbox™. If you are new to ModusToolbox™, see the ModusToolbox™ user guide.

2 Infineon resources

Infineon provides a wealth of data at the www.infineon.com page to help you to select the right IoT device for your design, and quickly and effectively integrate the device into your design. Infineon provides customer access to a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates. Customers can acquire technical documentation and software from the Infineon Support Community.

2.1 ModusToolbox™

Infineon provides ModusToolbox™ as the software development platform for CYW208xx applications. ModusToolbox™ software is a set of tools that enable you to integrate Infineon devices into your existing development methodology. One of the tools is a multi-platform, Eclipse IDE that supports application configuration and development.

2.2 Development kits

The CYW920819M2EVB-01 Development Kit enables you to evaluate single-chip Bluetooth® application using CYW20819. For more information on the kit including the user guide and schematics, see the CYW920819M2EVB-01 webpage.

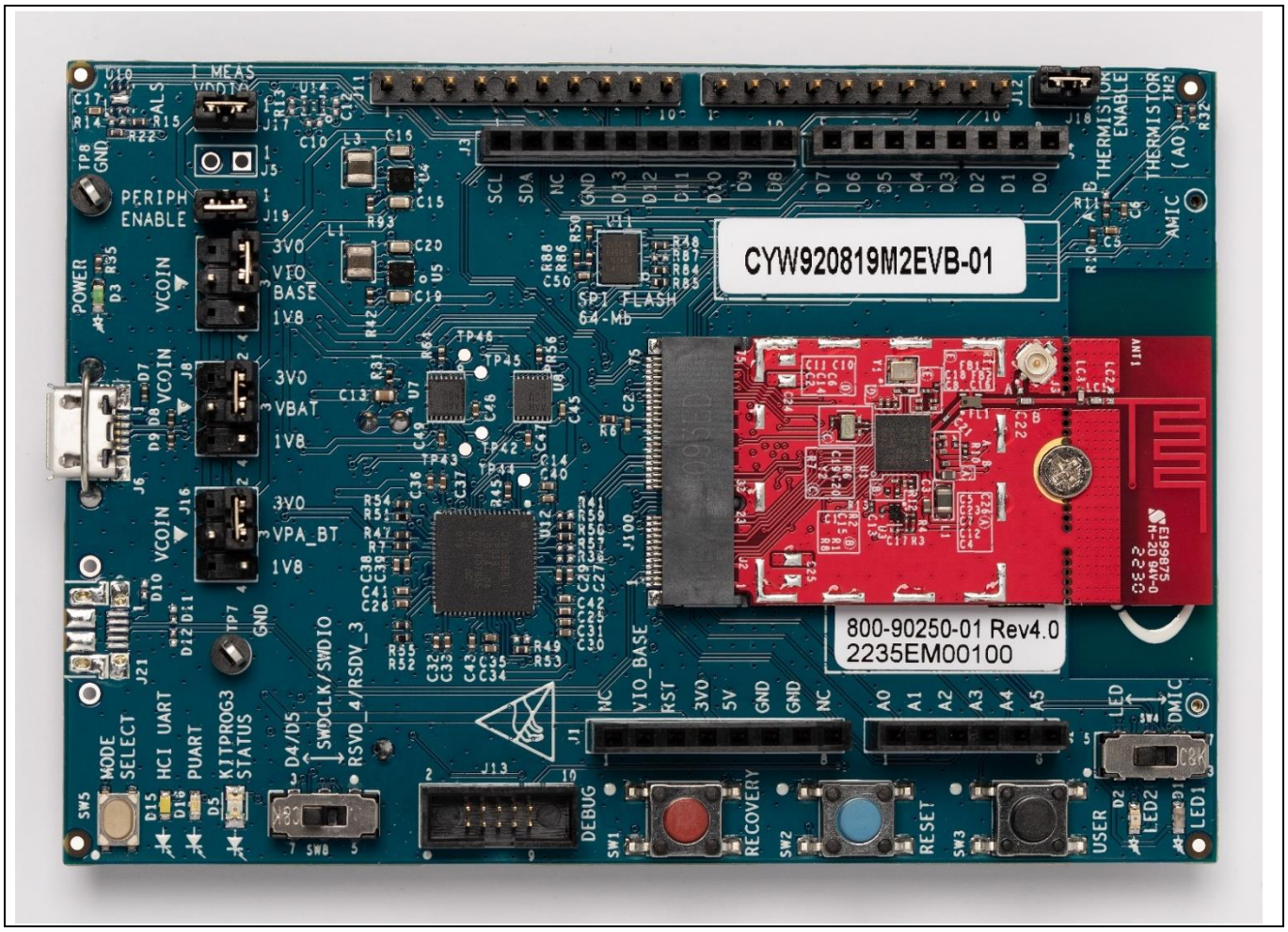

Figure 1 CYW920819M2EVB-01 Evaluation Kit

3 CYW20819 power management resources

3.1 System power modes

CYW20819 devices support various system power modes as shown in Figure 2—note that the power states are shown in the descending order of power consumption.

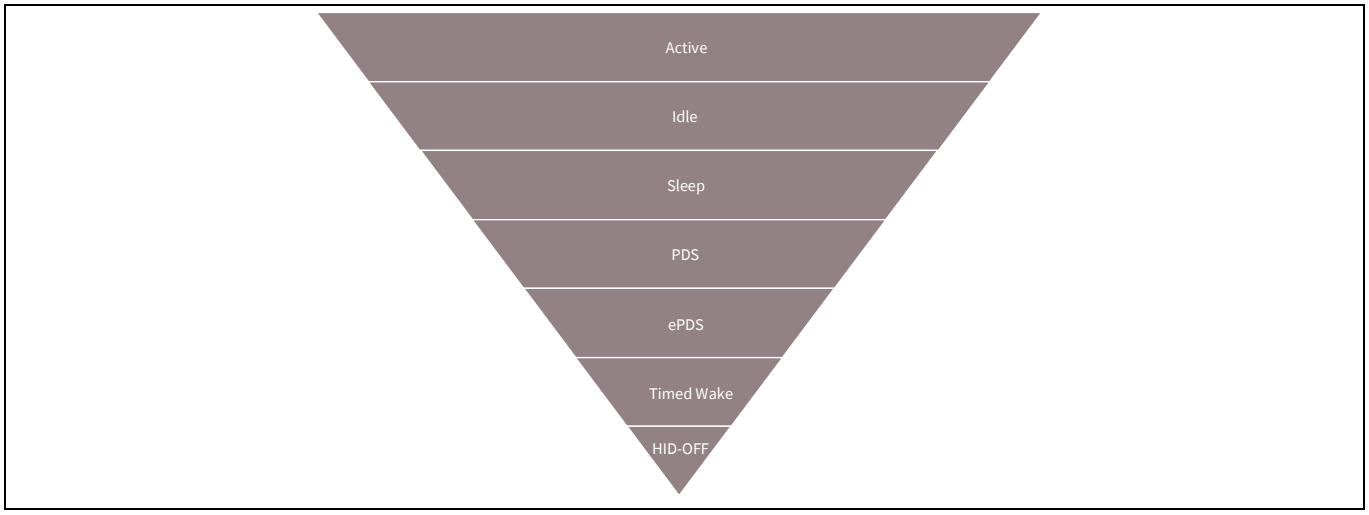

Figure 2 CYW920819 power modes

- **Active**: Active mode is the normal operating mode in which all peripherals are available and the CPU is active.
- **Idle**: In this mode, the CPU is in Wait for Interrupt (WFI) mode, and the HCLK, which is the high-frequency clock derived from the main crystal oscillator, is running at a lower clock speed. Other clocks are active and the state of the entire chip is retained. Idle mode is chosen when other lower-power modes are not possible.
- **PMU Sleep**: In this mode, the CPU is in WFI mode and the HCLK is not running. The PMU determines whether other clocks can be turned OFF, and does so accordingly. The state of the entire chip is retained including SRAM; internal LDO regulators run at a lower voltage (voltage is managed by the PMU), and SRAM is retained
- **Power Down Sleep (PDS):** Bluetooth® radio is powered down and digital core is mostly powered down except for RAM, registers, and some core logic. CYW20819 can wake up either after a programmed period of time has expired or if an external event such has switch interrupt is received via one of the GPIOs.
- **ePDS (extended PDS) mode**: This is an extension of the PDS Mode. In this mode, only the main RAM and ePDS control circuitry retain power. As in other modes, CYW20819 can wake up either after a programmed period or upon receiving an external event.
- **Timed wake:** Lean High Land IOs (LHL GPIOs), Real time clock (RTC), and Low power oscillator (LPO) are the only active blocks in this mode; the SRAM is not retained. The RTC that runs off the LPO is used to wake the device up after a predetermined time. The device starts executing from reset; therefore, the wakeup time is similar to that of POR.
- **HID-OFF (Deep Sleep) mode**: The core, radio, and regulators are powered down. Only the LHL I/O domain is powered. In this mode, CYW20819 can be woken up either by an event on one of the GPIOs or after a certain amount of time has expired. After wakeup, the device will go through full firmware initialization although it will retain enough information to determine that it came out of HID-OFF and the event that caused the wakeup. The LPO and RTC are turned off in this mode.

CYW20819 power management resources

Table 1 Low-power modes summary

Property / mode	Active	Idle	PDS	ePDS	Timed Wake/ HID-OFF
Clocks	All clocks active	HCLK at 48 MHz. Other clocks active	HCLK OFF, Other clocks depend on PMU	Only LPO active	Only LPO active
SRAM retained?	NA	Yes	Yes	Yes	No
Wake sources	NA	Any configured interrupt	GPIO, timer, Bluetooth® activity	LHL GPIO, timer, Bluetooth® activity	LHL GPIO, timer
Bluetooth® activities	All allowed	All allowed	Everything allowed	Everything allowed. Minimum sleep time should be ~10 ms	None

Table 2 shows the state of the peripherals in various power states. “WFI” means Wake From Interrupt and “Retained” means that the SRAM contents are retained.

Table 2 Peripherals in low-power modes

Peripheral / mode	Active	Idle	Sleep	PDS	ePDS	Timed Wake/ HID-OFF
CPU	On	WFI	WFI	WFI	WFI	WFI
SPI	On	On	On	Off	Off	Off
I2C	On	On	On	Off	Off	Off
HCI UART	On	On	On	Off	Off	Off
PUART	On	On	On	Off	Off	Off
ADC	On	On	On	Off	Off	Off
PWM	On	On	On	Off	Off	Off
TRNG	On	On	On	Off	Off	Off
Key scan	On	On	On	On	On	Off
Timer	On	On	On	On	On	Off
GPIO	On	On	On	On	On	On
SRAM	On	On	On	Retained	Retained	Off
WDT	On	On	On	On	Off	Off
RTC	On	On	On	On	On	On
LPO	On	On	On	On	On	On

3.2 Device power domains

The power domains on the chip (and the peripherals they supply) are managed by the PMU as listed in Table 3:

Table 3 Device power domains

Power domain	Peripherals	Operational power modes
VDDC	SRAM, Patch RAM	VDDC power domain is used to supply power to SRAMs in ePDS mode so that the contents are retained.
VDDCP	PWM	PWM is switched OFF in ePDS mode but can work in higher-power modes.
VDDCG	SWD, I2C, SPI, PUART, HCI UART, WDT, Dual Input 32-bit Timer, flash, ROM	These hardware blocks can operate until the PMU enters ePDS.
VBAT/LHL	GPIO, Analog PMU, RTC, LPO	These hardware blocks can operate until the PMU enters HID-Off.
VBAT/LHL	Aux ADC	Aux ADC can operate until the PMU enters ePDS.

3.3 Power Management Unit (PMU)

The PMU is a software implementation with the necessary hardware support to lower the average power consumption of the device. The PMU module is executed from the context of the lowest-priority thread, which is the Idle thread. Whenever the PMU gets CPU time, it tries to put the device in one of the low-power modes available in the device. The primary tasks of the PMU include the following:

- Implementing low-power modes, transitions, and timing
- Communicating with different blocks and modules to register callbacks for pre-sleep and post-sleep states
- Implementing the wake ISR, which wakes up the device at the required time
- Switching and calibrating of low-power oscillators (LPO)

3.3.1 Low-power oscillator (hardware)

The LPO is a clock that runs in low-power modes to source some blocks. This clock is used to time the wakeup at desired intervals. The Bluetooth® specification requires an accuracy of ±250 ppm for Basic Rate/Enhanced Data Rate (BR/EDR) and ±500 ppm for Bluetooth® Low Energy. Therefore, the LPO accuracy must lie within this range if there is an active Bluetooth® operation.

The LPO uses one of the following possible sources:

- **Low-power xtal (LPX)**: This clock is derived from the external main crystal oscillator but runs in low-power mode during sleep. LPX typically has a frequency of 1 MHz, which is derived from the 24-MHz crystal and provides a clock accuracy of ±20 ppm or better but consumes relatively more current compared to other LPO sources. LPX is useful when maintaining a BR/EDR connection in a master role where high accuracy is required. It can sometimes also serve as a substitute for the external 32-kHz crystal.
- **External LPO clock**: CYW20819 has pins to which an external digital clock or a 32.768-kHz crystal can be connected; the accuracy should be within ±250 ppm as required by the Bluetooth® specification. If an external LPO is used, the PMU will use it under all conditions when a 20-ppm accuracy clock is not required. If a 20-ppm clock is required at any time, the LPX will be used instead of the external LPO clock.

- **Internal LPOs**: The internal LPOs belong to the Lean High Land (LHL) domain, which is always powered ON. There are two internal LPOs: one which has lower accuracy and consumes less current, and another with higher accuracy but consumes more current. These LPOs are calibrated by the PMU and used based on the accuracy required by the Bluetooth® subsystem.

3.3.2 PMU tasks with respect to LPO (software)

- **Switching**: The PMU switches between different LPOs. Active peripherals can request a clock with a particular ppm value so that the PMU can select an appropriate LPO source based on the required ppm. The PMU might also turn OFF unused LPO sources to save power. The usual preference for the LPO source is in the following order: External LPO clock > Internal LPO > LPX. Although it is possible for AIROC™ Bluetooth® devices to enter low-power modes using internal LPOs, it is recommended to use an external LPO.
- **Calibration**: The PMU calibrates internal LPOs at regular intervals to improve accuracy. Calibration is performed against the 24-MHz external crystal. The PMU calibrates internal LPOs every 500 ms when active. If in low-power mode, the calibration is done when the chip wakes up. The calibration takes place only if the internal LPOs are selected as the source.

3.3.3 PMU operation

The PMU module executes from the context of the lowest-priority thread (Idle thread). It interacts with other modules to determine whether it can put the system to sleep. The PMU collects the necessary information to determine whether to go to sleep and the duration of sleep. The PMU manages the states of the clocks and the LDO voltages in low-power modes.

Power modes differ in the following aspects:

- Blocks that are turned OFF
- Duration for which the system goes to sleep
- Accuracy of the internal clock while sleeping
- Wakeup sequences

3.4 Implementing low power in ModusToolbox™

The PMU determines all power mode transitions. The firmware can control whether PDS or ePDS are allowed, but it cannot prevent Idle or Sleep. It is up to the PMU to determine which sleep mode to enter depending on scheduled events. For example, even if the user firmware allows ePDS, the PMU may decide not to go into ePDS because of an event scheduled for a short time in the future. In that case, going to ePDS would not be beneficial because there is time (and power) required to shut down and reinitialize the system. The PMU can transition to Idle or Sleep at any time. Figure 4 shows the flow for entering low-power modes.

CYW20819 power management resources

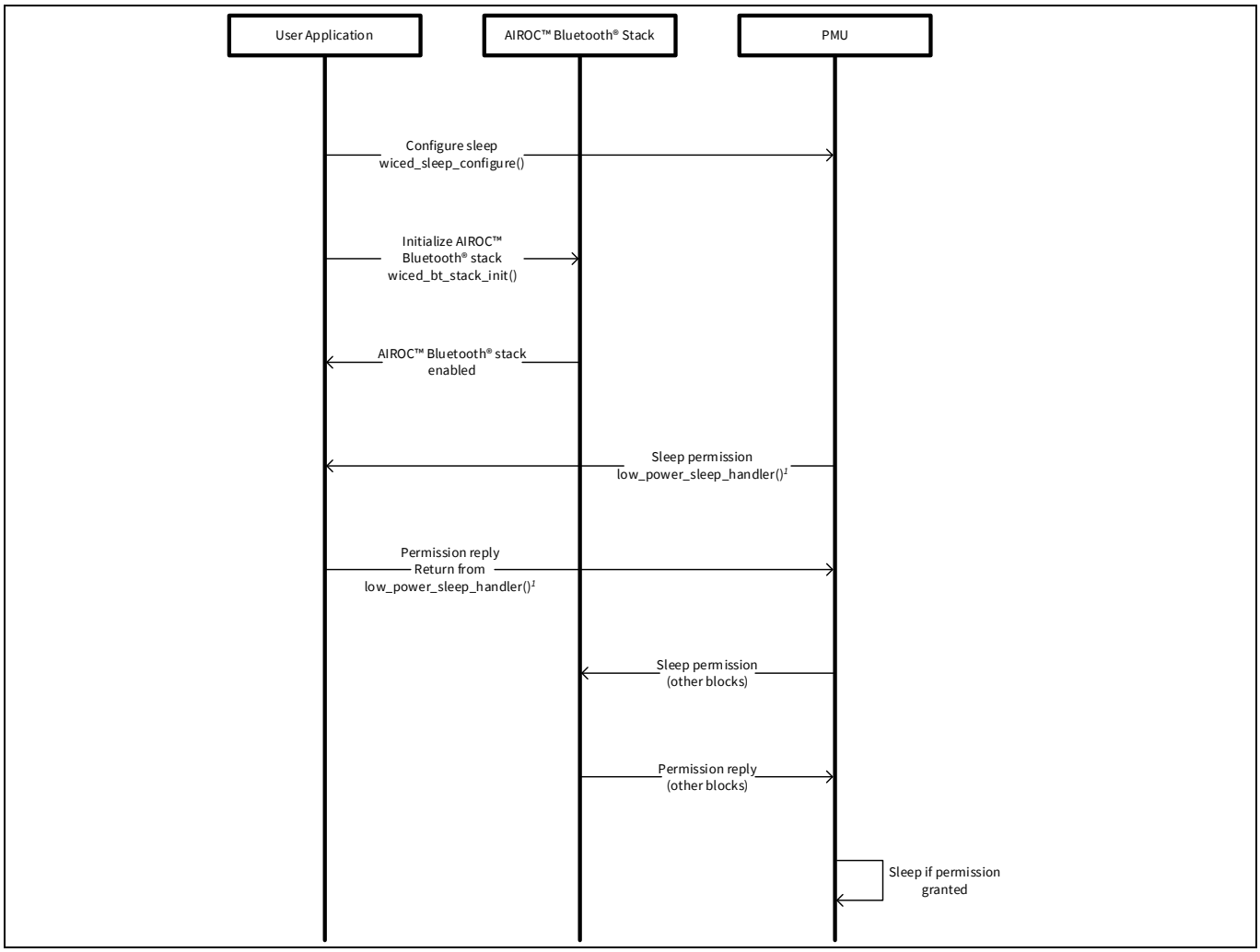

Figure 3 Low-power mode transitions

low_power_sleep_handler *is a custom function implemented in the example referenced in the application note. You should implement your own function per your requirements. See Section 3.4.3 for more details.*

In firmware, you configure sleep by providing wake sources and by providing a callback function that the PMU will call whenever it wants to go to low-power mode. In the callback function, you can disallow sleep, allow sleep without shutdown (ePDS), or allow sleep with shutdown (timed wake/HID-Off) depending on your firmware requirements. You can also provide a post-sleep callback that the PMU calls after it wakes up from ePDS or PDS. This call can be used to initialize peripherals that lose their context when the device enters ePDS.

Do the following to enable low power in your application:

1. Add sleep header files.
2. Set up the sleep configuration.
3. Create a sleep callback function.
4. Create a post-sleep callback function.
5. Create wakeup events (timers and threads).
6. Check for the reset reason (for ePDS wakeup).

3.4.1 Sleep header files

The header file *wiced_sleep.h* contains the API functions related to low-power operation of CYW20819. This header file must be included in the source code to call sleep API functions.

3.4.2 Sleep configuration

The `wiced_sleep_configure` function is used to enable low-power operation of the device. The parameter passed to this function is a pointer to a structure of type `wiced_sleep_config_t` that contains the sleep configuration information. The structure is defined as follows:

```
/** This structure defines the sleep configuration parameters to be passed to
 *  wiced_sleep_configure API
*/
typedef struct
{
    wiced_sleep_mode_type_t  sleep_mode;                 /**< Defines whether to
sleep with or
without transport */
    wiced_sleep_wake_type_t  host_wake_mode;             /**< Defines the active
level for host wake
signal */
    wiced_sleep_wake_type_t  device_wake_mode;           /**< Defines the active
level for device
wake signal. If device wake signal is not
present on the device then GPIO defined in
device_wake_gpio_num is used */
    uint8_t device_wake_source;                          /**< Device wake
source(s). See 'wake
sources' defines for more details. GPIO is
mandatory if WICED_SLEEP_MODE_TRANSPORT is
used as sleep_mode*/
    uint32_t  device_wake_gpio_num;                              /**< GPIO# for
device wake, mandatory for
WICED_SLEEP_MODE_TRANSPORT */
    wiced_sleep_allow_check_callback      sleep_permit_handler;        /**< Call
back to be called by sleep
framework to poll for sleep permission */
     wiced_sleep_post_sleep_callback  post_sleep_cback_handler;  /**< Callback to
application on wake
                                                                  from sleep */
}wiced_sleep_config_t;
```

CYW20819 power management resources

In the firmware, you need to do the following to configure sleep:

1. Declare a global variable of type `wiced_sleep_config_t`.
2. Initialize the elements of the structure just after stack initialization.
3. Call `wiced_sleep_configure()`.

The elements in the structure are:

- **sleep_mode:** `WICED_SLEEP_MODE_NO_TRANSPORT` or `WICED_SLEEP_MODE_TRANSPORT`.

  If you select `WICED_SLEEP_MODE_NO_TRANSPORT`, the device will enter sleep only if no host is connected (i.e., HCI UART CTS line not asserted). If you select `WICED_SLEEP_MODE_TRANSPORT`, the device will enter sleep only when an external HCI host is connected i.e., the HCI UART CTS line is asserted and deasserted at least once. If the device is being used stand-alone without an external HCI host, you should choose `WICED_SLEEP_NO_TRANSPORT`.
- **host_wake_mode:** `WICED_SLEEP_WAKE_ACTIVE_LOW` or `WICED_SLEEP_WAKE_ACTIVE_HIGH` depending on the polarity of the interrupt to wake the host (if a host is connected). This only applies if sleep_mode is `WICED_SLEEP_MODE_TRANSPORT`. The Host Wake function is on a dedicated device pin, but it can be multiplexed into other I/Os (this multiplexing feature is not currently supported in the API).
- **device_wake_mode:** `WICED_SLEEP_WAKE_ACTIVE_LOW` or `WICED_SLEEP_WAKE_ACTIVE_HIGH` depending on the polarity of the interrupt for the host to wake the device (if a host is connected). This only applies if sleep_mode is `WICED_SLEEP_MODE_TRANSPORT`. The Device Wake function is on a dedicated device pin, but it can be multiplexed into other I/Os (this multiplexing feature is not currently supported in the API).
- **device_wake_source:** The wake source can be key scan, quadrature sensor, GPIO, or a combination of those. For example, you may want to use an interrupt from a sensor as a GPIO wake source so that the device wakes whenever new sensor data is available.

```
/** Wake sources.*/
#define WICED_SLEEP_WAKE_SOURCE_KEYSCAN  (1<<0)  /**< Enable wake from keyscan */
#define WICED_SLEEP_WAKE_SOURCE_QUAD     (1<<1)  /**< Enable wake from quadrature sensor */
#define WICED_SLEEP_WAKE_SOURCE_GPIO     (1<<2)  /**< Enable wake from GPIO */
#define WICED_SLEEP_WAKE_SOURCE_MASK     (WICED_SLEEP_WAKE_SOURCE_GPIO | \
                                         WICED_SLEEP_WAKE_SOURCE_KEYSCAN | \
                                                WICED_SLEEP_WAKE_SOURCE_QUAD) /**< All
      wake sources */
```

- **device_wake_gpio_num:** This entry specifies the GPIO that is used to wake the device from sleep. This applies only if `device_wake_source` includes GPIO. Another way to configure the GPIO for wakeup is to register the GPIO for interrupt in the user application.
- **sleep_permit_handler:** This element requires you to provide a function pointer for the callback function that will be called by the PMU to request sleep permission and when sleep is entered. This function will be described next.
- **Post_sleep_cback_handler:** This element requires you to provide a function pointer for the callback function that will be called by the PMU after the device wakes up from ePDS and PDS as described below.

3.4.3 Sleep callback function

The sleep permission callback function takes one argument of type `wiced_sleep_poll_type_t`, which specifies the reason for the callback (`WICED_SLEEP_POLL_SLEEP_PERMISSION` or `WICED_SLEEP_POLL_TIME_TO_SLEEP`); it returns a `uint32_t`.

For a `WICED_SLEEP_POLL_SLEEP_PERMISSION` callback, the return value must be one of the following based on firmware requirements:

- `WICED_SLEEP_NOT_ALLOWED` – The application can return this value if it does not want the device to enter PDS or ePDS mode.
- `WICED_SLEEP_ALLOWED_WITHOUT_SHUTDOWN` – When this value is returned, the device can enter any low-power mode, including ePDS.
- `WICED_SLEEP_ALLOWED_WITH_SHUTDOWN` – When this value is returned, the device will enter timed wake or HID-Off mode.

For a `WICED_SLEEP_POLL_TIME_TO_SLEEP` callback, you must return the maximum time that the system should be allowed to sleep for ePDS mode. This is typically set to `WICED_SLEEP_MAX_TIME_TO_SLEEP`, but may also be returned as '0' if you don't want the system to go to sleep at that time. If you want to wake at a specific time, it is better to use a timer.

If you want to enter timed wake mode (i.e., HID-Off with wakeup at a predetermined time), this parameter can be used to specify the time the device should remain HID-Off mode. For example, if you pass 10000000, the device will enter HID-Off mode for 10 seconds. If you specify `WICED_SLEEP_MAX_TIME_TO_SLEEP`, the device to enter HID-OFF mode and will not wake until a GPIO interrupt occurs.

Remember that the PMU makes the final decision – it polls the firmware and each peripheral to see which type of sleep is allowed and how long sleep will be possible, and then decides which mode is possible.

3.4.4 Post-sleep callback function

The post–sleep callback function takes one argument of type `wiced_bool_t`, which specifies whether the application needs to reinitialize any peripherals. If the parameter value is `TRUE`, the peripherals need to be reinitialized and configured; otherwise not. Only the PUART is initialized by the stack by default, so you don't need to reinitialize it. The return type is `void`.

3.4.5 Wakeup events

The firmware may need events that cause it to wake periodically or on specific events. For example, you may need to read a sensor value every few seconds or respond to a user input such as a button press.

For periodic wakeup, you can either use a timer or threads with delays that allow the thread to sleep (i.e., `ALLOW_THREAD_TO_SLEEP`). The device will not enter ePDS/PDS unless all threads and timers are in a sleep state.

As previously discussed, during sleep configuration, the device wake source may be configured. If that source is set to GPIO, the specified GPIO will wake the system. However, you will not get a GPIO interrupt handler callback unless you register the callback function using `wiced_hal_gpio_register_pin_for_interrupt`. You can also use multiple GPIOs to wake up the device by registering multiple pins for interrupt.

You can also use timers to wake up from PDS and ePDS. It is sufficient to use functions present in the wiced_timer.h to set up timers for waking up from PDS. However, if you want to use timers in ePDS mode, you need to use the functions from clock_timer.h. This is because after waking up from ePDS the device loses the timer context. Therefore, even though the timer will wake up the timer from ePDS, it will not be able to call the callback. Also, there is no function to determine whether the wakeup from ePDS was due to a timer or GPIO; therefore, you need to compare the system time to determine whether the timeout has happened after wakeup. This can be done by noting the system time when going to sleep into a variable stored in Always-On (AON) RAM and after wakeup comparing the current time with the value in this variable and see whether the timeout has passed.

3.4.6 Entering HID-Off/Timed-Wake

HID-Off and timed wake modes can be entered by returning `WICED_SLEEP_ALLOWED_WITH_SHUTDOWN` from the sleep permission callback function for the `WICED_SLEEP_POLL_SLEEP_PERMISSION` case. You need to specify the time to sleep (for timed wake) by returning the time to sleep value from the sleep permission callback function for the `WICED_SLEEP_POLL_TIME_TO_SLEEP` case. See Sleep Callback Function for more details.

Note the following:

- If you are using internal LPOs, the PMU will switch to a higher-accuracy LPO when in a connection, which will consume a higher current.
- If the device gets disconnected, the PMU needs some time to switch to a low-accuracy LPO.

3.4.7 Wakeup reason

After waking up from HID-Off or timed wake mode, the device starts from the reset vector. The wakeup reason can be determined by using the following API function call:

```
wiced_sleep_wake_reason_t wiced_sleep_hid_off_wake_reason(void);
```

This function will return the wakeup reason to the application as POR, wakeup from Timed-Wake mode, or GPIO wakeup.

3.5 Recommendations for low power

3.5.1 Use CYW20819 to gate current paths

GPIOs P26, P27, P28, and P29 can source and sink higher currents (16 mA at 3.3 V and 8 mA at 1.8 V) than the rest of the GPIOs. These GPIOs can be used to directly power some external peripherals (like a temperature sensor that falls within the current limit). The power to the external peripherals can then directly be controlled through the CYW208xx device; when not required, the peripherals can be turned OFF, thus saving power at the system level.

3.5.2 Disable unused blocks

You can save current by disabling unused blocks like PWM and UARTs using the driver APIs.

3.5.3 Periodic wakeup timers

The average power consumption is determined by the power consumption in the CPU active and CPU sleep periods. To achieve the lowest power consumption, the CPU sleep period should be maximized, and the CPU active period should be minimized. All the low-power modes except HID-Off can maintain advertising (with the same data) or a connection (by sending empty packets), so the application does not have to wake the device just to advertise or maintain a connection. If you want to periodically read and send data, you can use application timers to wake the device.

3.5.4 Clocks

In some designs which require extensive data processing by the CPU, running the CPU at faster clock speeds can reduce average power consumption. The aim is to reduce the time spent by the CPU in active mode and race to lower power modes.

3.5.5 GPIOs

GPIOs can hold their logic states in low-power modes (ePDS and timed wake/HID-Off). This is helpful when you need to hold external logic at a fixed level, but it can lead to wasted power if the pins needlessly source or sink current. Also, upon wakeup from ePDS and timed wake/HID-Off mode, GPIOs will go to High-Z. You can use the following function if you want a specific GPIO to maintain its state upon wakeup from low-power modes:

```
wiced_hal_gpio_slimboot_reenforce_cfg ((uint8_t pin, uint16_t config)
```

You should analyze your system design and determine the best state for your GPIOs during low-power operation. If holding a digital output pin at logic 1 or 0 is best, match the same digital level using the following function:

```
void wiced_hal_gpio_set_pin_output (UINT32 pin, UINT32 val);
```

Configure all unused GPIOs to High-Z unless there is a specific reason to use a different drive mode. This can be either done in the platform config file or using the following API function:

```
wiced_hal_gpio_configure_pin (WICED_P00, (GPIO_OUTPUT_DISABLE |
GPIO_INPUT_DISABLE), GPIO_PIN_OUTPUT_LOW);
```

3.5.6 RTOS

The CYW20819 device supports ThreadX by Express Logic which is built into the device ROM; a license is included for anyone using AIROC™ chips. AIROC™ devices provide wrapper APIs for using RTOS features. The RTOS can be used to reduce the CPU active time by using threads for running various tasks and using mechanisms such as mutexes and semaphores to communicate between the threads. If properly done, this can eliminate the use of timers and polling in your application.

For example, consider an application running two tasks: Task1 and Task2. Task2 is dependent on completion of Task1. Instead of polling a flag to allow Task2 to run, you can set a semaphore in Task1 which will immediately allow Task2 to run once Task1 completes. This will avoid polling for a flag and thus will eliminate wasted CPU power.

4 Low-power code example

4.1 CE236664 – CYW20819/CYW20820 - BTSDK low power

This code example shows how to enter and exit ePDS and HID-Off/timed wake modes. This example will be used to show how to calculate average current consumption. The project uses a switch to transition between various power modes. For more details, see the README file in the CYW20819/CYW20820 – BTSDK low power code example.

References

[1] Application notes

- ModusToolbox™ 2.4 user guide

[2] Code examples

- Code examples for ModusToolbox™ software – Visit this code example for a comprehensive collection of code examples using ModusToolbox™ IDE.

[3] Device documentation

- CYW20819 device datasheet
- CYW20820 device datasheet

[4] Development kits

- CYW920820M2EVB-01 Evaluation Kit
- CYW920819M2EVB-01 Evaluation Kit

[5] Tool documentation

- Eclipse IDE for ModusToolbox™ - The Infineon IDE for IoT designers

Revision history

Document revision	Date	Description of changes
**	2019-03-01	Initial document
*A	2019-07-10	Updated document title Updated document to include all CYW208xx devices Added details about PDS mode Added details about post-sleep callback Added details about delay requirement before HID-Off Added details about GPIO reinforce API Added details about wakeup reason API
*B	2019-12-20	Changes related to timed wake/HID-Off mode
*C	2022-11-02	Updated to the Infineon template Updated low power recommendations
*D	2023-08-03	Updated CE Number and CE name

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2023-08-03

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-25270 Rev. *D

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.