AN225270

CYW208xx BLE Low-Power Guidelines

Author: Abhishek Khinvesra

Associated Part Family: CYW20819, CYW20820, CYW89820

Associated Code Examples: CE225540

Related Application Notes: AN225684

AN225270 describes how to use CYW208xx power modes to optimize power consumption. The CYW208xx Bluetooth (BT) system-on-chip (SoC) is a BT 5.0-compliant, Arm® Cortex®-M4 CPU-based ultra-low-power SoC that supports both Classic BT and Bluetooth Low Energy (BLE). 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.

Contents

1 Introduction

Bluetooth Low Energy (BLE) devices such as heart-rate monitors are typically battery-operated. A long battery life is a key requirement for such devices. CYW208xx gives 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 CYW208xx device.

CYW208xx is a Bluetooth (BT) 5.0-compliant SoC with an integrated 2.4-GHz transceiver with support for both BLE and Classic Bluetooth. The device is intended for use in audio, IoT, sensors (medical, home, security), and HID markets. CYW208xx features high levels of integration to reduce external components, thereby minimizing system footprint and costs.

This application note requires a basic understanding of the CYW208xx device architecture and ability to develop an application using ModusToolbox™. If you are new to CYW208xx, see the AN225684 - Getting Started with CYW208xx application note. If you are new to ModusToolbox, see the ModusToolbox IDE User Guide.

2 Cypress Resources

Cypress provides a wealth of data at http://www.cypress.com/internet-things-iot to help you to select the right IoT device for your design, and quickly and effectively integrate the device into your design. Cypress 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 Cypress Support Community website (http://community.cypress.com/).

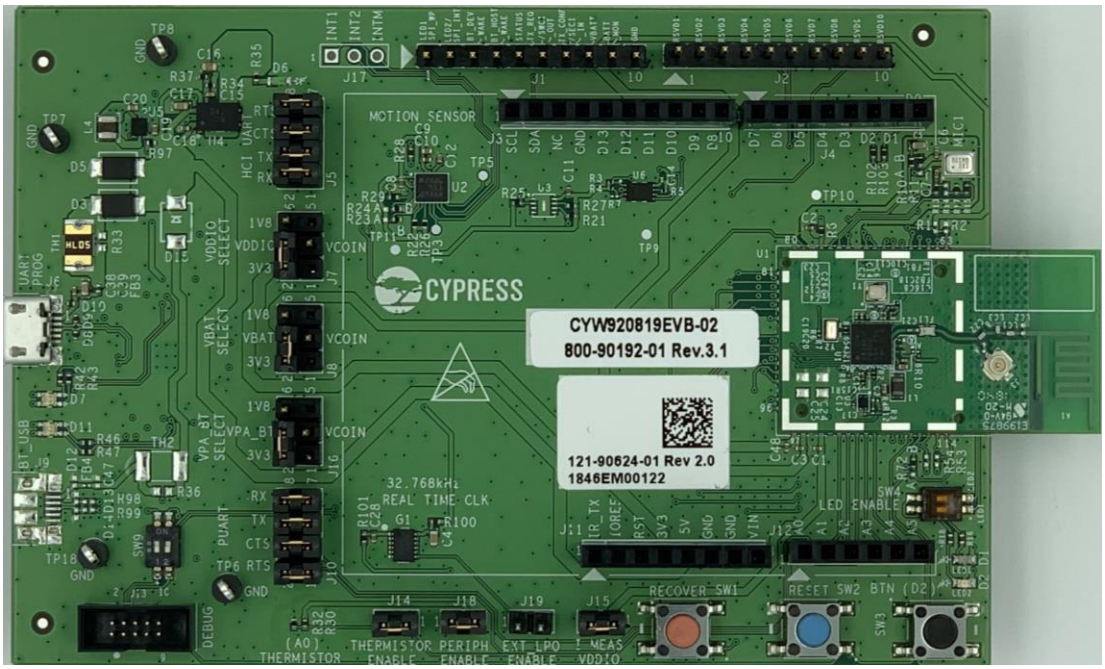
2.1 ModusToolbox

Cypress provides ModusToolbox as the software development platform for CYW208xx applications. ModusToolbox software is a set of tools that enable you to integrate Cypress devices into your existing development methodology. One of the tools is a multi-platform, Eclipse-based Integrated Development Environment (IDE) called the ModusToolbox IDE that supports application configuration and development.

2.2 Development Kits

CYW920819EVB-02 is the development kit that supports prototyping and application development using the CYW20819 device. More information on the kit including the User Guide and schematics can be found on the CYW920819EVB-02 web page. Similar kits can be found for the CYW20820 (CYW920820EVB-02) and CYW89820 (CYW989820EVB-01) devices.

Figure 1. CYW920819EVB-02 Evaluation Kit



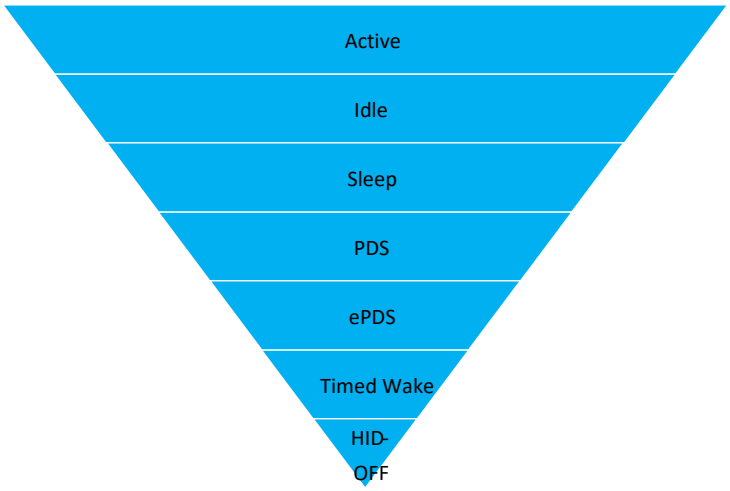
3 CYW208xx Power Management Device Resources

This section describes the supported power modes. It also describes the various hardware and software resources available in the CYW208xx device to implement low-power applications.

3.1 System Power Modes

The CYW208xx devices support various system power modes as shown in Figure 2:

Figure 2. CYW208xx 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 the Wait for Interrupt (WFI) state and the HCLK, which is the high-frequency clock derived from the main crystal oscillator, is running at a lower clock speed (4 MHz if DMA is not active; 8 MHz if DMA is active). 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. Wake from this mode is almost instantaneous as the device must service only the new interrupt. Any configured interrupt will wake the device.

Sleep: In this mode, the CPU is in the WFI state and the HCLK is not running. The Power Management Unit (PMU) determines whether other clocks can be turned OFF, and does so accordingly. If there is no activity or there is only BLE activity like a connection or advertisement, the 24-MHz external crystal is turned OFF; if there is a BR/EDR master activity, the 24-MHz crystal is kept ON. The state of the entire chip is retained including SRAM; internal LDO regulators run at a lower voltage (voltage is managed by the PMU). Peripherals like SPI, I2C, and UART are powered ON. Wake can be through various sources such as a GPIO interrupt, timer, or BT activity.

Before entering sleep, the PMU takes a snapshot of all the enabled interrupts and then disables them so that only the wake ISR is allowed. Upon wakeup, the wake ISR is executed first, all the system clocks are restored; the wake ISR is then cleared. After returning from the wake ISR, all other interrupts are enabled, and any pending interrupts are serviced. If the device goes to sleep with the 24-MHz crystal OFF, the wakeup time is ~2 ms; if the device goes to sleep with the crystal ON, the wakeup time is ~500 µs.

PDS: This mode is the same as the sleep mode with most of the peripherals such as UART and SPI turned OFF. The entire SRAM is retained.

ePDS: In this mode, the entire device is OFF except for the RTC, LPO, and Lean High Land (LHL) GPIO interrupt to wake the device. The entire SRAM memory is retained. The device enters this state through WFI but exits through a reset vector. It can wake up using an LHL GPIO interrupt, a timer (RTC), or BT activity.

Before entering this mode, the PMU stores the CPU state and some register values (including the program counter) in SRAM. Upon wakeup, the device will go to the reset vector and will determine if the wakeup was from ePDS or power on reset. If the wakeup is from ePDS, it will restore the register values and CPU. As a result, it will start executing from the instruction after the WFI instruction.

The PMU enters ePDS only if the minimum time to sleep is ~10 ms or larger. Therefore, if BT activity such as a connection event is scheduled within the next 10 ms, the device will not enter ePDS. Wakeup time is ~2 ms.

Timed-Wake: Lean High Land IOs (LHL, lowest power domain. See Table 3), RTC, and LPO are the only active blocks in Timed-Wake and SRAM is not retained. 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 the same as that of power on reset.

HID-OFF: This mode is the same as the Timed-Wake mode except that the RTC cannot wake up the device. Everything is OFF in this mode and the SRAM is not retained. The only wakeup source is an LHL GPIO interrupt. The device starts executing from reset after wakeup; therefore, the wakeup time is the same as that of power-on reset.

Table 1 summarizes low-power modes in CYW208xx and their properties:

Table 1. Low-Power Modes Summary

Property / Mode	Active	Idle	Sleep/PDS	ePDS	HID-OFF/Timed Wake
Clocks	All clocks active	HCLK at 4/8 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, BT activity	LHL GPIO, timer, BT activity	LHL GPIO, timer
BT activities	Everything allowed	Everything allowed	Everything allowed	Everything allowed. Minimum sleep time should be ~10 ms	Nothing allowed

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

Table 2. Peripherals in Low-Power Modes

Peripheral / Mode	Active	Idle	Sleep	PDS	ePDS	HID-OFF/Timed Wake
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
Keyscan	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	The 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	The Aux ADC can operate until the PMU enters ePDS.

3.3 Power Management Unit (PMU)

This section details the hardware and software resources used to implement low-power modes. 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
- Detecting, 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. This clock has low power consumption, which makes it ideal for operation in low-power modes. The Bluetooth specification requires an accuracy of ±250 ppm for Basic Rate/Enhanced Data Rate (BR/EDR) and ±500 ppm for Bluetooth Low Energy (BLE). 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:** CYW208xx 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, then the PMU will use this under all conditions when a 20-ppm clock is not required. If a 20-ppm clock is required at any time, then 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 BT 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 WICED 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. The 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.

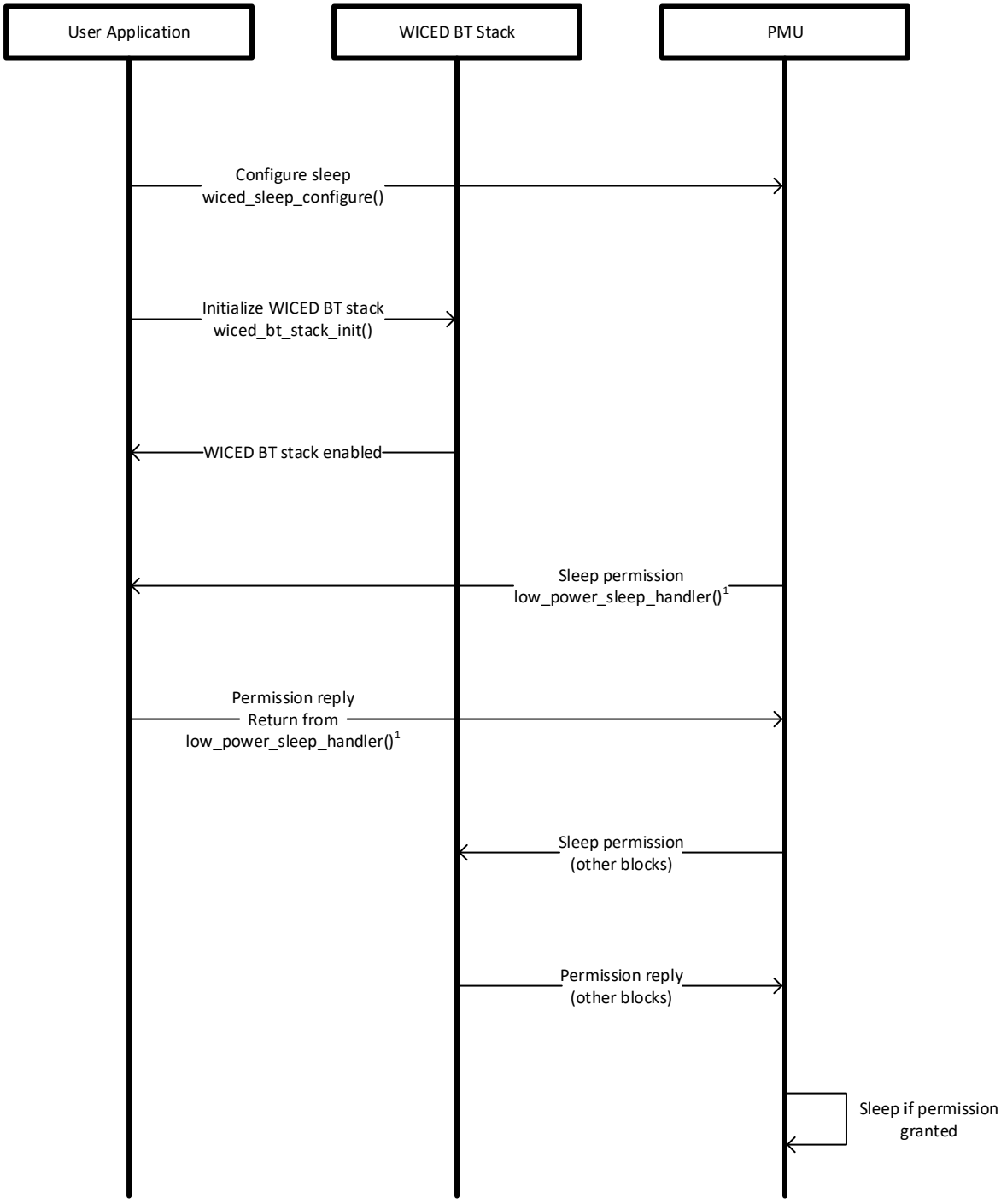
Power modes differ in the following aspects:

- 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 ePDS, Timed-Wake, or HID-Off are allowed, but it cannot not 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 re-initialize the system. The PMU can transition to Idle or Sleep at any time. Figure 3 shows the flow for entering low-power modes.

Figure 3. Low-Power Mode Transitions



1 – `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 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 sleep configuration.
3. Create a sleep callback function.
4. Create a post-sleep callback function.
5. Create wakeup events (timers, threads, etc.)
6. Check for reset reason (For HID-Off and Timed-Wake).

3.4.1 Sleep Header Files

The header file *wiced_sleep.h* contains the API functions related to low-power operation of CYW208xx. This header file must be included in the source code to call sleep API functions.

3.4.2 Sleep Configuration

The function `wiced_sleep_configure` 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 like this:

```
/** 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;
```

In the firmware, you need to do the following for sleep configuration:

1. Declare a global variable of type `wiced_sleep_config_t`.
2. Initialize all 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 the former, the device will enter sleep only if no host is connected (i.e., HCI UART CTS line not asserted). If you select the latter, the device will enter sleep only when an external HCI host is connected (i.e., HCI UART CTS line is asserted). 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). This pin is not available on the 208xx 40-pin package, but the `device_wake_source` pin can be used for this purpose.

device_wake_source: The wake source can be keyscan, 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 */
```

device_wake_gpio_num: This entry specifies the GPIO that is used to wake the device from sleep. This only applies if `device_wake_source` includes GPIO. Another way to configure the GPIO for wake up 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 or PDS. This function will be 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 ePDS or PDS 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. You should not depend on this parameter as a wake source for ePDS mode.

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 will 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 re-initialize any peripherals. If the parameter value is TRUE, the peripherals need to be re-initialized and configured; otherwise not. Only the PUART is initialized by the stack by default, so you don't need to re-initialize 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`.

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 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 Power-On Reset (POR), wakeup from timed-wake mode, or GPIO wakeup.

3.5 Recommendations for Low Power

3.5.1 Use CYW208xx 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 unnecessary 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 CPU active period power consumption and CPU sleep period power consumption. 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 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 the 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 CYW208xx device supports ThreadX by Express Logic which is built into the device ROM and a license is included for anyone using WICED chips. WICED provides 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 of some sort 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 CE225540 – CYW208xx Switching Between Power Modes

This code example shows how to enter and exit ePDS and HID-Off/Timed-Wake. 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 document CE – CYW208xx Low Power available along with the code example in the Cypress GitHub repo.

5 Summary

Many power management options can be used in CYW208xx devices. By following proper methods, you can optimize your design and ensure that the power modes and features of the CYW208xx give the best option for the lowest power consumption without degrading the performance of battery powered devices.

6 Related Documents

Application Notes	
AN225684 - Getting started with CYW208xx	Describes CYW208xx Bluetooth SoC and how to build your first BLE application using the device in ModusToolbox IDE
Code Examples	
Visit the Cypress GitHub site for a comprehensive collection of code examples using ModusToolbox IDE	
Device Documentation	
CYW20819 Device Datasheet	
Development Kit	
CYW920819EVB-02 Evaluation Kit	
Tool Documentation	
ModusToolbox IDE	The Cypress IDE for IoT designers

About the Author and Contributors

Name: Abhishek Khinvesra

Title: Staff Applications Engineer

Background: Abhishek Khinvesra has a B.Tech degree in Electronics and Instrumentation Engineering from Vellore Institute of Technology, India. He has many years of experience in developing BLE-based applications.

Document History

Document Title: AN225270 - CYW208xx BLE Low-Power Guidelines

Document Number: 002-25270

Revision	ECN	Submission Date	Description of Change
**	6498021	03/01/2019	New Application Note
*A	6619159	07/10/2019	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	6757678	12/20/2019	Changes related to Timed-Wake/HID-Off mode

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709