# Chapter 8: Low Power

AIROC™ Bluetooth® SDK (BTSDK) MCUs provide many different low power modes. Most power mode transitions are handled automatically by the device's Power Measurement Unit (PMU). However, there are a few things to be aware of to make sure your application achieves the lowest power possible.

## Table of contents



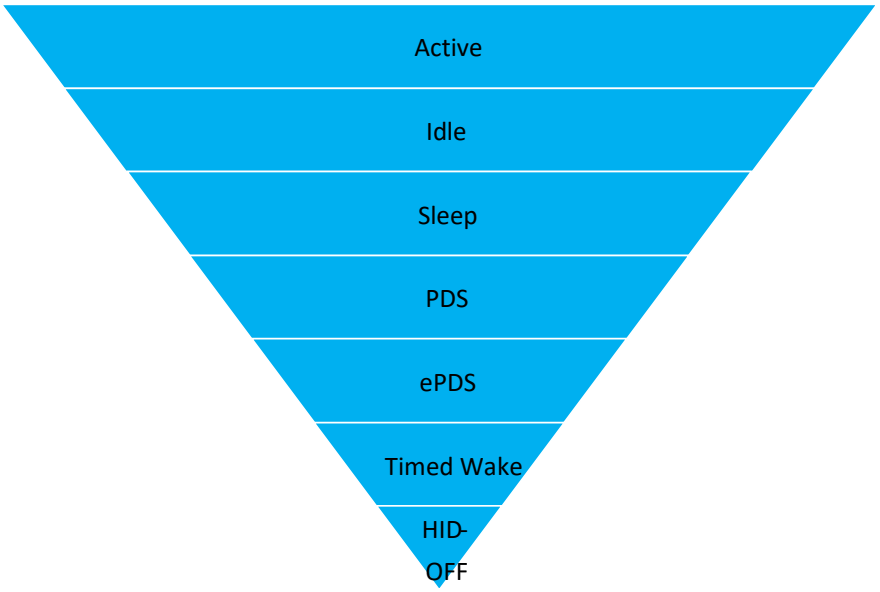
## Document conventions

| Convention | Usage | Example |
|---|---|---|
| `Courier New` | Displays code and text commands | `CY_ISR_PROTO(MyISR);`<br>`make build` |
| *Italics* | Displays file names and paths | *sourcefile.hex* |
| [**bracketed, bold**] | Displays keyboard commands in procedures | [**Enter**] or [**Ctrl**] [**C**] |
| **Menu > Selection** | Represents menu paths | **File > New Project > Clone** |
| **Bold** | Displays GUI commands, menu paths and selections, and icon names in procedures | Click the **Debugger** icon, and then click **Next**. |

## 8.1 Power mode overview

This chapter will cover the CYW20835 power modes. Other AIROC™ Bluetooth® SDK (BTSDK) MCUs may support different power modes. See the documentation for your specific device to see if there are differences. Much of the material in this chapter is taken from application note CYW20819 Low-Power Guidelines (AN225270). See that document for additional details.

The CYW20835 has 8 power modes which are illustrated here from highest to lowest power consumption.




A brief description of each mode is provided here:

| Mode | Description |
|---|---|
| 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) 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/Pause mode is chosen when the other lower power modes are not possible. Any configured interrupt can wake the device. |
| Sleep | In this mode, the CPU is in WFI and the HCLK is not running. The PMU determines if other clocks can be turned off and does so accordingly. The state of the entire chip is retained, the internal LDOs run at a lower voltage (voltage is managed by the PMU), and SRAM is retained. Wake can be from various sources such as GPIO, timer, or Bluetooth activity. |
| Power Down Sleep (PDS) | This mode is an extension of the PMU Sleep wherein most of the peripherals such as UART and SPI are turned OFF. The entire memory is retained, and on wakeup the execution resumes from where it was paused. |
| Enhanced Power Down Sleep (ePDS) | The entire device is OFF except for RTC, LPO, and a comparator to wake the device. The entire SRAM memory is retained. The device enters this state through WFI but exits through a reset vector. Before entering this mode, the PMU stores the CPU state and some register values (including the program counter) in SRAM, which are restored after wakeup so that the CPU starts executing from where it stopped. It can wake up using a GPIO, a timer, or Bluetooth activity. |

| Mode | Description |
|---|---|
| Timed-Wake | LHL GPIOs, RTC, and LPO are the only active blocks. The only wakeup sources are a timer or a GPIO interrupt. SRAM is not retained. 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 Timed-Wake, but in HID-OFF mode even the LPO and RTC are turned OFF. The only wakeup source is a GPIO interrupt. SRAM is not retained. The device starts executing from reset; therefore, the wakeup time is the same as that of power on reset. |

A summary of the modes and properties in each mode is shown here:

| Property / Mode | Active | Idle | Sleep / 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 retention | N/A | Yes | Yes | Yes | No |
| Wake Sources | N/A | Any configured interrupt | GPIO, timer, Bluetooth® activity | LHL GPIO, timer, Bluetooth® activity | LHL GPIO, timer |
| Bluetooth® activities | All allowed | All allowed | All allowed | All allowed, minimum time in ePDS ~10 ms | None |

The power domains on the chip (and the peripherals they supply) are managed automatically by the Power Management Unit (PMU) based on the power mode as described in the following table:

| S. No | Power Domain | Peripherals | Operational Power Modes |
|---|---|---|---|
| 1 | VDDC | SRAM, Patch RAM | The SRAM and Patch RAM are retained for ePDS mode and higher. |
| 2 | VDDCP | PWM | The PWM is available in PDS mode and higher. |
| 3 | VDDCG | SWD, I2C, SPI, PUART, WDT, ARM GPIO, Dual Input 32-bit Timer, flash, ROM | These hardware blocks can operate in PDS mode and higher. |
| 4 | VBAT/LHL | LHL GPIO, Analog PMU, RTC | These hardware blocks can operate in Timed Wake mode and higher. |
| 5 | VBAT/LHL | Aux ADC | The Aux ADC is available in PDS mode and higher. |

See the application note referenced above for additional details about which peripherals are available in each power mode.

### 8.1.1 PMU mode transitions

The PMU is responsible for all power mode transitions on the device. The application firmware can control whether ePDS, Timed Wake and HID-Off modes are allowed, but it cannot not prevent PDS, Idle or Sleep. It is up to the PMU to determine which sleep mode to enter depending on scheduled events. For example, if the firmware allows ePDS, the PMU may decide to go into PDS instead of 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, Sleep or PDS any time.

In the firmware, you configure sleep by providing wake sources and by providing a sleep permit handler callback function that the PMU will call whenever it wants to go into ePDS, Timed Wake, or HID-Off modes. In the callback function, you can allow or disallow the requested low power mode transition.

In ePDS and PDS modes, the device will wake every time a Bluetooth® LE event must be serviced. For example, during advertising, the device will wake every time a new advertising packet needs to be sent. Once a connection has been established, the device will wake for each connection interval.

### 8.1.2 Wake times

Approximate wake times for various modes are as follows:

| Mode | Typical Wake Time (ms) |
|---|---|
| Active | N/A |
| Idle | ISR service time – almost instantaneous |
| Sleep / PDS | 0.5 – 2 |
| ePDS | 2 |
| Timed Wake | Same as POR – a few hundred ms |

### 8.1.3 Current consumption

The exact current consumption depends on the firmware, but the approximate values to be expected are as follows:

| Mode | Typical Current (µA) |
|---|---|
| Active | >50 (very FW dependent - often much higher) |
| Idle | |
| Sleep / PDS | <20 |
| ePDS | ~10 with no Bluetooth LE |
| Timed Wake | ~3 |
| HID Off | ~3 |

## 8.2 Low-Power firmware

To use low power modes in the firmware, the following steps are required:

1. Add sleep header files
2. Set up sleep configuration
3. Create a sleep permit handler callback function
4. Create a post-sleep callback function
5. Update connection parameters on a GATT connection
6. Create wakeup events (GPIO interrupts, timers, threads, etc.)
7. Update Advertising Settings
8. Update Bluetooth Address Setting

### 8.2.1 Sleep header files

The header file *wiced_sleep.h* contains the API related to low power operation. That header file must be included in the source code to call the sleep API functions. You should also include *wiced_bt_l2c.h* (that's a lower-case L before the number 2) so that we will be able to update the connection parameters in the firmware via L2CAP.

### 8.2.2 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: (1) declare a global variable of type `wiced_sleep_config_t`; (2) initialize all the elements of the structure just after stack initialization; and (3) call `wiced_sleep_configure`.

The elements in the structure are:

- **sleep_mode**: This can be either `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_MODE_NO_TRANSPORT`.

- **host_wake_mode**: This can be either `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 onto other IOs.
- **device_wake_mode**: This can be either `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 IOs.
- **device_wake_source**: The wake source can be keyscan, quadrature sensor, GPIO, or a combination of those. For example, you may want to use a sensor interrupt 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 which GPIO is used to wake the device from sleep. This only applies if `device_wake_source` includes GPIO. An alternate 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 callback function that is be called by the PMU to request sleep permission and again just before sleep is entered. This function will be described next.
- **post_sleep_cback_handler**: This element requires you to provide a callback function that is called when the device wakes from sleep in case the application needs to take any action at that time.

### 8.2.3 Sleep permission callback function

The sleep permit handler 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`) and it returns a `uint32_t`.

For a `WICED_SLEEP_POLL_SLEEP_PERMISSION` callback the PMU is asking which modes are OK to use at the current time. The return value must be one of the following based on the requirements of the firmware:

- `WICED_SLEEP_NOT_ALLOWED` – The application can return this value if it does not want the device to enter PDS or lower sleep modes.
- `WICED_SLEEP_ALLOWED_WITHOUT_SHUTDOWN` – The application can return this value if low power is allowed. This value should be returned if the firmware wishes to allow PDS/ePDS but not Timed-Wake or HID-Off.
- `WICED_SLEEP_ALLOWED_WITH_SHUTDOWN` –When this value is returned, the device can enter any of the low power modes including Timed-Wake and HID-Off.

For a `WICED_SLEEP_POLL_TIME_TO_SLEEP` callback the PMU is about to enter a low power mode. You must return the maximum time that the system should be allowed to sleep. 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 typically should not depend on this parameter as a wake source.

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 makes sense.

### 8.2.4 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.

### 8.2.5 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 user input such as a button press.

For periodic wakeup, you can either use a timer or you can use threads with delays that allow the thread to sleep (i.e. `ALLOW_THREAD_TO_SLEEP`). The device will not enter sleep 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 then the specified GPIO interrupt will wake up 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`.

### 8.2.6 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 (i.e. HID-Off).

### 8.2.7 Connection parameter settings

The connection interval, latency and timeout are typically chosen by the Central. However, upon a connection, the Peripheral may request to update those values. The Central may or may not agree to the request. To reduce power consumption, it is recommended that the connection interval be set to 100ms or more once a GATT connection is established. In addition, the timeout should be set appropriately based on the connection interval, latency, and other application requirements.

An L2CAP function is used to request new values for the connection interval and timeout. This is done in the GATT connection callback when the connection state is "connected" (i.e. when the connection comes up). The

function takes the `BD_ADDR`, the min and max intervals in units of 1.25ms, latency in number of connection intervals that can be missed, and the timeout in units of 10ms.

To request new connection parameters, use the following function:

```
wiced_bt_l2cap_update_ble_conn_params (
    wiced_bt_device_address_t rem_bdRa,  // BD_ADDR of remote device
    uint16_t min_int,                    // Min connection interval in units of 1.25ms
    uint16_t max_int,                    // Max connection interval in units of 1.25ms
    uint16_t latency,                    // Latency - number of connection intervals
    uint16_t timeout);                   // Timeout in units of 10ms
```

### 8.2.8 Advertising settings

As you will see in the exercises, advertising can take a significant amount of power. If your device is expected to advertise frequently, reducing the advertising duty cycle as much as possible will reduce power consumption.

### 8.2.9 Bluetooth® address settings

Using the resolvable private address privacy mode (i.e. the RPA refresh timeout is set to something other than `WICED_BT_CFG_DEFAULT_RANDOM_ADDRESS_NEVER_CHANGE`) causes the firmware to wake frequently (about twice per second) due to a timer that is used to track the resolvable private addresses. This will cause increased power consumption. Therefore, RPA should be avoided in power critical applications.

## 8.3 Programming while in low power mode

When the device is asleep it is not listening for HCI commands which are needed to get the device in the correct mode for programming. To circumvent this issue, the kit has a "recovery" mode. To enter recovery mode:

1. Press and hold the recovery button on the base board (red button on the left)
2. Press and release the reset button (blue button in the center)
3. Release the recovery button
4. Program as normal

*Note:* *The recovery button just holds the SPI MISO pin of the external serial flash high so that when the device resets, it does not receive any instructions from the serial flash. Therefore, the device defaults to bootloader mode.*

## 8.4 Kit power measurement

The kit has three power supplies that can be measured by removing the appropriate shunt and instead connecting an ammeter across the jumper pins. The connection points are:

| Name | Use | Jumper |
|---|---|---|
| VDDIO | I/Os | J17 |
| VBAT | Core | J8 |
| VPA_BT | Bluetooth® power amplifier | J16 |

*Note:* *The VBAT and VPA_BT jumpers are also used to select the supply voltage value (1.8 V, 3.3 V, or from a coin cell battery). The ammeter must be connected across the same pins as the shunt to provide the same supply voltage.*

*Note:* *The CYW20835 is a 1.8 V device so even if 3.3 V is selected on the baseboard, the module has a regulator that steps the voltage down to 1.8 V to supply the device.*

*Note:* *The VDDIO supply voltage value is set by a different jumper named VIO_BASE (J7). However, that jumper also supplies several other components on the base board such as voltage level translators and digital microphones. Therefore, it is not recommended to use that jumper for measuring VDDIO.*

## 8.5 Exercises

### Exercise 1: Sleep Mode Impact on Power

In this exercise, you will create an application based on a low power example and program it to your kit to observe low power in action.

**Application creation**

1. Create a new ModusToolbox™ application for the CYW920835M2EVB-01 BSP.

   Use the HAL Low Power code example. Name the new application *ch08_ex01_low_power*.

2. Open the Bluetooth® Configurator and change the device name to **<inits>_low_power**.
3. Save changes and close the configurator.
4. Open the m*akefile* and set `BT_DEVICE_ADDRESS?=random`
5. Look at the *README.md* file and review the files *low_power_208xx.c* and *208xx_ble.c* to familiarize yourself with how the application works.

**Testing**

1. Open a terminal window to the PUART.

   This will allow you to see sleep events and to determine how often the device wakes up and goes back to sleep during different operations.

2. Program the application onto the kit.

*Note:* *If your device is in a low power mode you may have to put it into recovery mode first to program it. To enter Recovery mode:*

- *Press and hold the recovery button*
- *Press and release the reset button*
- *Release the recovery button*

3. The device beings in HID-off mode. Press the user button on the kit to wake it up and to start advertisements.
4. Open AIROC™ Connect and connect to the device.
5. Open the Battery service widget.
6. Click **Start Notify**.
7. The application will send a battery level notification every 5 seconds.
8. Press the user button once on the kit to disconnect

   This puts the device into HID-off mode with a timed wake at 10 seconds.

*Note:* *If you press the user button before the 10 second timer expires, the device will wake right away. You can tell from the message printed whether the wake occurred due to a GPIO interrupt or timed wake.*

9. If you have an ammeter available, you can connect it across the appropriate jumper pins:

   - VDDIO: J17
   - VBAT: J8
   - VPA: J16

*Note:* *Disconnect power from the kit before removing shunts and connecting the ammeter.*

*Note:* *Turn on the ammeter before powering the kit since some ammeters are high impedance when turned off.*

*Note:* *Do not disconnect or turn off the ammeter while the kit is powered.*

*Note:* *The current will vary while the device switches power modes so you will have to observe min/max values or use an ammeter with an averaging function.*

**Questions**

1. Which lines in the code are used to configure and initialize sleep?

2. What is used as a wakeup source?

3. What is the name of the sleep permit handler function?

4. When are the connection interval, latency, and timeout values updated in the code?

# 8.6 Appendix

## 8.6.1 Exercise 1 answers

1. Which lines in the code are used to configure and initialize sleep?

   In *low_power_208xx.c*:

```
/* configure to sleep if sensor is idle */
low_power_sleep_config.sleep_mode               = WICED_SLEEP_MODE_NO_TRANSPORT;
low_power_sleep_config.device_wake_mode         = WICED_SLEEP_WAKE_ACTIVE_LOW;
low_power_sleep_config.device_wake_source       = WICED_SLEEP_WAKE_SOURCE_GPIO;
low_power_sleep_config.device_wake_gpio_num     =
      WICED_GET_PIN_FOR_BUTTON(WICED_PLATFORM_BUTTON_1);
low_power_sleep_config.host_wake_mode           = WICED_SLEEP_WAKE_ACTIVE_HIGH;
low_power_sleep_config.sleep_permit_handler     = low_power_sleep_handler;
low_power_sleep_config.post_sleep_cback_handler = low_power_post_sleep_cb;

if(WICED_BT_SUCCESS != wiced_sleep_configure(&low_power_sleep_config))
{
    WICED_BT_TRACE("Sleep Configure failed\r\n");
}
```

2. What is used as a wakeup source?

   `WICED_PLATFORM_BUTTON_1` is used as a wakeup source.

3. What is the name of the sleep permit handler function?

```
low_power_sleep_handler
```

4. When are the connection interval, latency, and timeout values updated in the code?

   The values are updated in the GATT connect callback function when a connection is established. The code is in *low_power_208xx_ble.c*:

```
/* Update connection parameters to 100 ms */
if(TRUE != wiced_bt_l2cap_update_ble_conn_params(p_conn_status->bd_addr,
    CONNECTION_INTERVAL, CONNECTION_INTERVAL,
    SLAVE_LATENCY, SUPERVISION_TIMEOUT))
    {
        WICED_BT_TRACE("Connection parameters update failed\r\n");
    }
```

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

**Published by**

**Infineon Technologies AG**

**81726 Munich, Germany**



**IMPORTANT NOTICE**

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”) .

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, 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.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

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.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

**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.