WICED Studio

WICED™ Development Power Save System

Doc. No.: 002-20894 Rev. **

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
www.cypress.com

Contents

About This Document

Purpose and Scope

This document describes the power saving implementation and features of the Cypress Wireless Internet Connectivity for Embedded Devices (WICED™; pronounced “wicked”) Development System for Wi-Fi stations.

Audience

This document is intended for software engineers using the WICED Development System to create low-power applications for embedded wireless devices.

Acronyms and Abbreviations

In most cases, acronyms and abbreviations are defined on first use.

For a comprehensive list of acronyms and other terms used in Cypress documents, go to www.cypress.com/glossary.

IoT Resources and Technical Support

Cypress provides a wealth of data at www.cypress.com/internet-things-iot to help you 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 (community.cypress.com/).

1 802.11 (Wi-Fi) Power Saving Overview

The subsections in this chapter are partly based on information contained in *802.11 Wireless Networks, The Definitive Guide [1]*. Developers planning to integrate products using 802.11 Wi-Fi are highly encouraged to purchase a copy of this book.

The 802.11 standard includes several parameters that allow stations to save power, although power saving is accomplished at the expense of throughput or latency to the station.

1.1 Listen Interval

To save power, Wi-Fi stations sleep by powering down most of the Wi-Fi subsystems. While the stations are asleep, access points (APs) buffer frames for them. The sleeping Wi-Fi stations periodically wake up to listen to traffic announcements and to determine whether the AP has any buffered frames. When stations associate with an AP, part of the data in the Association Request frame is the listen interval. The listen interval indicates to the AP the frequency at which a power save station will wake up to listen to beacon frames.

An AP may use the listen interval as a guide to determine the duration it should retain buffered frames for a power station. Larger listen intervals require more AP memory for frame buffering.

In reality, APs generally do not consider the listen interval requested by a station. If a station sets the listen interval to a value, there is no guarantee that the AP will buffer all the packets received for the station while the station is asleep. Wi-Fi Certification test plans do not currently test behavior related to the listen interval, and implementation of listen interval functionality is not enforced.

Most APs enforce an association timeout on stations, i.e., if the AP has not received a frame from a station within the association timeout (usually 60 seconds) or the station has not returned an ACK to a keep-alive frame, the station will be disassociated. The association timeout cannot be negotiated by the station.

1.2 DTIM Period

The delivery traffic indication map (DTIM) period is a parameter associated with an infrastructure network, and is advertised in an AP Beacon frame. All Beacon frames include a traffic indication map (TIM) which indicate the availability of buffered frames to stations. Unicast frames buffered for individual stations are delivered in response to a query from the station. This polled approach is not suitable for multicast and broadcast frames, because it takes too much capacity to transmit multicast and broadcast frames multiple times. Instead of the polled approach, broadcast and multicast frames are delivered after every DTIM.

Increasing the DTIM allows stations to conserve power more effectively at the cost of buffer space in the AP and delays in reception of multicast and broadcast frames by all stations, including stations in active mode.

The default DTIM beacon interval for most APs is either DTIM=1 or DTIM=3. In the case of DTIM=3, the station needs to only wake from low-power mode to receive every third beacon and any ensuing queued broadcast or multicast traffic.

1.3 Practical Power Saving

Wi-Fi stations use different mechanisms to save power.

Stations with low duty cycle and long battery life requirements, Wi-Fi sensors for example, may use the standard 802.11 Power Save Poll (PS-Poll) mechanism. Stations requiring higher throughput, wireless speakers for example, may consider using a non-standard 802.11 power save mechanism. These methods are described in this section.

Stations requiring a more fine-grained power save mechanism to differentiate power save for various traffic priorities may consider using WMM Powersave. WMM Powersave is not currently supported by the WICED Studio.

1.3.1 Power Save Poll

Power Save Poll is suited for stations that primarily transmit data to the Wi-Fi network at low duty cycle. The PS-Poll mechanism works as follows:

1. Stations (STA) sends a frame to the AP with the Power Management bit set and then goes to sleep.
2. AP notes the STA has gone to sleep and buffers frames for the STA.
3. STA wakes up periodically to check the AP beacon frame for an indication of buffered frame(s). The wake period depends on the configured listen interval and whether the STA is configured to receive group addressed frames from a beacon containing a DTIM.
4. If the AP indicates it has buffered frame(s) addressed to the STA, the STA sends a PS-Poll frame to the AP.
5. The AP sends a frame to the STA, and sets the More Data bit if additional frames are buffered.
6. The STA might send another PS-Poll frame to retrieve additional buffered frames (if the More Data bit was set) or return to sleep.

1.3.2 802.11 Power Save without Poll

A non-standard mechanism known broadly as 802.11 Power Save without Poll (PS-non-Poll) enables Wi-Fi stations to use 802.11 power saving based on a 'trigger' frame as follows:

1. STA sends a frame to the AP with the Power Management bit set and then goes to sleep.
2. AP notes the STA has gone to sleep and buffers frames for the STA.
3. STA wakes up periodically. The STA may (or may not) first check the AP beacon frame for an indication of buffered frame(s) before sending any frames to the AP.
4. The STA sends a Null Function data frame, or a data frame if available, to the AP with the Power Management bit cleared.
5. The AP notes the STA is awake and sends buffered frame(s).
6. STA receives the frame(s), waits for a timeout period to receive additional frame(s) (if available), and then returns to sleep as described in Step 1.

1.3.3 Association Timeout Limit

If a station does not expect to receive directed frames from the AP asynchronously, and it is not interested in receiving broadcast or multicast traffic, then further power savings may be achieved.

Since APs generally have an association timeout limit with a default value of 60 seconds, a power save station must wake up before association timeout expires, and send or receive a directly addressed frame to inform the AP that the station is still associated. Failure to comply results in the AP de-authenticating the station. Some APs allow the association timeout limit to be set to a value higher than 60 seconds, but the higher limit is not signaled to the station and must be manually configured.

The Wi-Fi Alliance is currently working to adopt various new power save features that are in the 802.11v standard.

1.4 Final Note

Even though the 802.11 transmitted power consumption is at least five times higher than the received power consumption, even for medium transmit duty cycle applications, much of the energy in a battery powered Wi-Fi station is consumed by the receiver. Unless power save techniques are used, the 802.11 receiver may be powered ON for significant periods of time while the station waits for network clients to respond to requests. It does not take very long for a 130-mW receiver power consumption to discharge a battery.

The key to minimizing power consumption is to minimize the ON duty cycle of the system. Minimizing the integrated area under the current consumption curve is critical to obtaining maximum life from a battery.

2 WICED Power Save Implementation

The power saving implementation provided in the WICED Development System attempts to minimize the combined ON time of the Wi-Fi chip and the MCU.

2.1 Wi-Fi Power Saving

There are eight WICED API functions available to control power saving on the Wi-Fi chip. Table 2-1 describes these functions.

`wiced_init()`	Powers up the Wi-Fi chip and initializes the SPI/SDIO bus interface, RTOS, and networking interface.
`wiced_wifi_enable_powersave()`	Enables PS-Poll mode on the Wi-Fi chip (see Power Save Poll)
`wiced_wifi_enable_powersave_with_throughput()`	Enables 802.11 PS-non-Poll mode on the Wi-Fi chip (see 802.11 Power Save without Poll). Use this mode when it is important to maintain throughput.
`wiced_wifi_disable_powersave()`	Disables power save mode on the Wi-Fi chip
`wiced_wifi_set_listen_interval()`	Sets the number of beacons to be skipped while the Wi-Fi chip is sleeping (only works when Wi-Fi power save is enabled)
`wiced_wifi_set_listen_interval_assoc()`	Informs the AP how many beacons will be skipped while the Wi-Fi chip is sleeping. The number of beacons actually skipped is set by the `wiced_wifi_set_listen_interval()` function (only works when Wi-Fi power save is enabled)
`wiced_wifi_get_listen_interval()`	Returns information about the number of beacons to be skipped while the Wi-Fi chip is sleeping
`wiced_deinit()`	Cuts power to the Wi-Fi chip entirely. Among other tasks, this function de-initializes network interfaces and threads associated with the wireless network interface.

Table 2-1. WICED Wi-Fi API Functions to Control Power Saving

When the Wi-Fi chip is enabled for power save, it sleeps between DTIMs as described in Practical Power Saving. If the beacon listen interval is subsequently configured to a value other than zero, the Wi-Fi chip only wakes up according to the value configured. It no longer wakes at DTIM intervals.

2.2 MCU Power Saving: STM32 and AT91SAM4S

This section applies to both STMicroelectronics STM32 and Atmel (Now Microchip) AT91SAM4S MCUs. The power save mode used by WICED for STM32 is Stop mode, and for AT91SAM4S is Wait Mode. In this section, Stop mode and Wait mode are referred to as power save mode. In power save mode, the MCU retains the entire contents of RAM.

There should be no noticeable difference in the operation of an application when MCU power save is enabled, other than a significant reduction in current consumption when the processor idles, and a possible reduction in maximum network data throughput.

2.2.1 How to Control MCU Power Save

There are four WICED API functions available to control power saving on the MCU. Table 2-2 describes these functions.
.

`wiced_platform_mcu_enable_powersave()`	Enables power save mode on the MCU
`wiced_platform_mcu_disable_powersave()`	Disables power save mode on the MCU
`wiced_network_suspend()`	Suspends network services and disables all network related timers. This API call prevents the MCU from unnecessarily waking up to service high frequency network timers
`wiced_network_resume()`	Resumes network services

Table 2-2. WICED API Functions to Control MCU Power Saving

Note:

- If an application uses interrupts, other than external general purpose I/O (GPIO) interrupts or the RTC interrupt, these interrupts will not be detected or serviced while the MCU is in power save mode.
- If the global `#define WICED_DISABLE_MCU_POWERSAVE` is enabled in *<WICED-Studio>/include/wiced_defaults.h*, the API functions in Table 2-2 will have no effect.
- When `wiced_platform_mcu_enable_powersave()` is called, the MCU might not immediately enter power save mode. The behavior of power save mode for FreeRTOs and ThreadX is described in RTOS Operation with MCU Power Save Mode.

2.2.2 RTOS Operation with MCU Power Save Mode

In the MCU power save mode, all clocks except the 32-kHz real time clock (RTC) are powered down. All memory and registers retain state. There are only two ways to exit power save mode: an external interrupt (i.e., an edge on a GPIO pin) or an RTC interrupt.

With power save mode enabled, the WICED Wi-Fi driver enables an out-of-band (OOB) interrupt on an external GPIO from the Wi-Fi chip. The OOB interrupt is required because the SDIO bus interface clock and interrupt handling on the MCU are disabled when power save mode is active. There is no additional requirement if the Wi-Fi chip uses gSPI to communicate with the MCU, since a GPIO interrupt is required for normal communication.

FreeRTOS Implementation

A platform specific function `platform_power_down_hook()` is integrated with the FreeRTOS idle thread to make applications work almost independent of power save mode.

The FreeRTOS scheduler normally runs the idle thread at every system tick when no other thread is ready to run. Rather than running the idle thread when power save is enabled, the processor is placed in power save mode. The FreeRTOS scheduler works as follows:

1. The idle thread determines the next pending timeout for delayed threads and calls the platform specific idle handler `platform_power_down_hook()` function.
2. The idle handler checks whether any threads have disabled power save mode. If so, power down is disabled. Otherwise continue to step 3.
3. The idle handler programs the RTC with the next pending timeout value.
4. The idle handler directs the MCU to enter power save mode until the next interrupt.
5. Later, an interrupt occurs either from an external GPIO pin or from the RTC timeout.
6. The processor wakes and the idle handler disables the RTC interrupt.
7. The idle handler reads the time elapsed from the RTC and returns it to the idle thread.
8. The idle thread uses the elapsed time to instigate any processing that was missed (for example, update the tick count).
9. The scheduler runs to start the threads that are now ready to run.

ThreadX Implementation

Unlike FreeRTOS, ThreadX does not provide an idle thread mechanism. ThreadX waits in the SVC software interrupt handler if there is no other task ready to run. The power save mechanism for ThreadX works as follows:

1. If no task is ready to run, the wait routine inside the SVC handler calls the `tx_low_power_enter()`ThreadX function.
2. The `tx_low_power_enter()` function checks the time remaining until the next ThreadX timer is due to expire. This timer expiry value is passed to the `platform_power_down_hook()`platform-specific idle handler.
3. The idle handler checks whether all threads mutually agree to enter the power save mode. If there is no agreement, a wait for interrupt (WFI) instruction is called, which makes the processor enter idle mode, leaving MCU peripherals powered up. The processor returns from idle mode as soon as one of the available interrupts is triggered. If all threads agree to enter power save mode, continue to step 4.
4. The idle handler programs the RTC with the next pending timeout value.
5. The idle handler directs the MCU to enter power save mode until the next interrupt.
6. After an interval, an interrupt occurs either from an external pin or from the RTC timeout.
7. The processor wakes and the idle handler disables the RTC interrupt.
8. The idle handler reads the elapsed time from the RTC.
9. The `tx_low_power_enter()` function passes the elapsed time to another ThreadX function, `tx_time_increment()`. This function updates the timer list and determines whether any tasks are ready to run.
10. The SVC handler restores the context of the task (if any) and the processor starts execution.

2.3 MCU Power Saving: CYW94390x

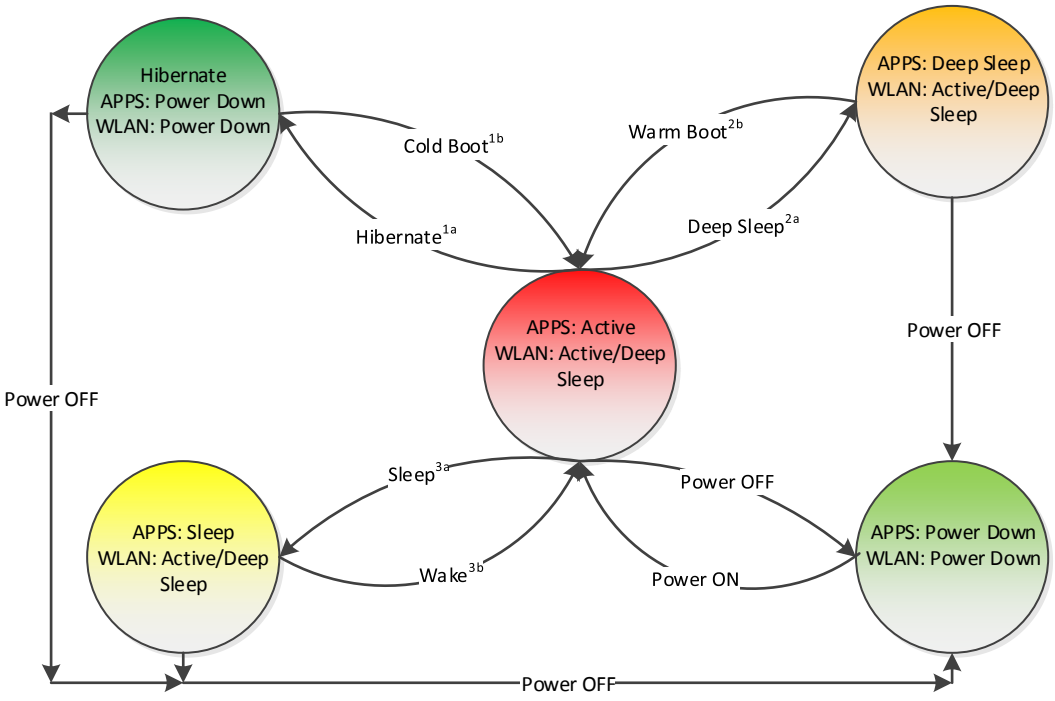

Figure 2-1. CYW94390x Power State Mode Transitions

The following are the power states in CYW4390x:

- Active
- Sleep (3a and 3b in Figure 2-1)
- Deep Sleep (2a and 2b in Figure 2-1)
- Hibernate (1a and 1b in Figure 2-1)

2.3.1 Active

The application CPU runs at 320 MHz when the system is idle, and the OS scheduler keeps looping to find the next thread to schedule.

By default, WLAN can go to low-power state. To change this behavior, issue:
`iovar: wwd_wifi_set_iovar_value( IOVAR_STR_MPC, 0, WWD_STA_INTERFACE ))`

2.3.2 Sleep

When the system is idle, the application CPU executes WFI instruction and switches to use ILP clock (32.7 kHz), and RTOS timer is set to tick-less mode. Tick-less mode helps in minimizing the current consumed as the application CPU does not have to wake up periodically (at each RTOS timer tick interrupt).

Wake up from sleep is through any timer or interrupt source.

Note: `PLATFORM_POWERSAVE_DEFAULT=1` and `PLATFORM_MCU_POWERSAVE_MODE_INIT=PLATFORM_MCU_POWERSAVE_MODE_SLEEP` needs to be defined in that application *makefile* for the system to enter sleep.

Note: Example application: *WICED_Studio/apps/powersave/powersave.c* `(POWERSAVE_STANDALONE_TEST_ACTIVE_WAIT_FOR_WLAN)`

Figure 2-2. Example of Sleep State Power Consumption

Figure 2-2 shows an example for Sleep state power consumption of ~6 mA Sleep current and ~8 mA average current consumed on BCM943907WCD2 board with *powersave.c*: `POWERSAVE_STANDALONE_TEST_ACTIVE_WAIT_FOR_WLAN`.

2.3.3 Deep Sleep

The application domain (CYM4390x MCU and peripherals) is powered down, 8 KB of always-on-memory (AONRAM) is preserved. AONRAM is used to save partial state of the WICED application and to accelerate booting. Returning from this mode is via warm boot (the difference from cold boot/power-on-reset is that 8 KB of AONRAM is preserved).

There are two ways the system can enter Deep Sleep:

1. **Co-operative:** OS scheduler detects whether the system is idle, it calculates the closest time when any thread is looking to wake up. This time is used to configure the timer to wake up.

   Note: Any busy-waiting kills these modes completely*.*

   Note: WICED system monitor is a RTOS thread that runs in the background. The monitor is scheduled periodically, and will result in the system waking up from Deep Sleep.

2. **Non-cooperative**: Application calls `platform_mcu_powersave_sleep()` which disables all interrupts except from timer, program timer, and go to sleep (WFI).

The application domain can be woken up from Deep Sleep by:

- **PMU timer:** Before the application goes to Deep Sleep, it configures the timer as explained earlier.
- **WLAN firmware:** WLAN firmware can wake up the application processor if it needs to communicate.
- **External event (GPIO):** GPIO_0 – GPIO_15 (except GPIO_13) can be programmed as GCI GPIOs to wake up the application CUP from Deep Sleep.

  Note: `PLATFORM_POWERSAVE_DEFAULT=1` and `PLATFORM_MCU_POWERSAVE_MODE_INIT=PLATFORM_MCU_POWERSAVE_MODE_DEEP_SLEEP` need to be defined in the application *makefile* for the system to enter Deep Sleep.

  Note: Example application: *WICED_Studio/apps/test/apps_sleep*: This application demonstrates how to save variables in AONRAM and co-operative sleep.

  Note: Example application: *WICED_Studio/apps/snip/ping_deepsleep/ping_deepsleep.c*.

  Note: Example application: *WICED_Studio/apps/powersave* (`POWERSAVE_STANDALONE_TEST_DEEPSLEEP_NO_ASSOC`) demonstrates non-cooperative (forced) sleep.

Figure 2-3. Example of Deep Sleep Average Power Consumption

Figure 2-3 shows the Deep Sleep average power consumption of ~402 uA (VBAT+VDDIO) on BCM943907WCD2 board with *powersave.c* `POWERSAVE_STANDALONE_TEST_DEEPSLEEP_ASSOC`.

2.3.4 Hibernate

In this mode, application CPU core, WLAN CPU core, peripherals, and PMU are switched OFF, except for silicon. System is woken up by timer or external pin.

Note: Example application: *WICED_Studio/apps/test/diagnostics/hibernation*

Figure 2-4. Example of Hibernate State Average Power Consumption

Figure 2-4 shows the Hibernate state average power consumption: ~4uA (VBAT+VDDIO) on BCM943907WCD2 board with *diagnostics/hibernation.c*.

2.3.5 Global Definitions for Power Save

Table 2-3 lists the RTOS timer options.

PLATFORM_TICK_CPU	CPU timer driver. It runs on CPU frequency (normally 320MHZ). This is enabled by default.
PLATFORM_TICK_PMU	PMU timer driver. It runs on 32.7 kHZ clock. This time resides in always-on domain and runs regardless of the domain state of the application. It is the only choice when the application is in Deep Sleep. Set to 1 if `PLATFORM_POWERSAVE_DEFAULT` is defined.
At least one timer driver must be compiled. For power-saving feature both are needed. When both timers are compiled, CPU timer is used at start up and PMU timer is used during low-power states.	

Table 2-3. RTOS Timer Options

Tick-less Mode	PMU Timer (enabled if `PLATFORM_APPS_POWERSAVE` is defined)	CPU Timer
`PLATFORM_TICK_POWERSAVE_MODE_TICKLESS_ALWAYS`	If requested sleep time is less than `PLATFORM_TICK_PMU_TICKLESS_THRESH` (default: 5) then normal mode, otherwise tick-less. In normal mode, timer is triggered for each RTOS tick (1 ms).	If requested sleep time is less than `PLATFORM_TICK_CPU_TICKLESS_THRESH` (default: 1) then normal mode, otherwise tick-less. In normal mode, timer is triggered for each RTOS tick (1 ms).
`PLATFORM_TICK_POWERSAVE_MODE_TICKLESS_NEVER`	Normal mode. Timer is triggered for each RTOS tick (1 ms).	Normal mode. Timer is triggered for each RTOS tick (1 ms).

PLATFORM_TICK_POWERSAVE_MODE_TICKLESS_IF_MCU_POWERSAVE_ENABLED This is default mode.	Tick-less mode if MCU power-saving is enabled (platform_mcu_powersave_enable()) and requested sleep time is more or equivalent to PLATFORM_TICK_PMU_TICKLESS_THRESH (default: 5), otherwise normal mode.	Tick-less mode if MCU power-saving is enabled (platform_mcu_powersave_enable()) and requested sleep time is more or equivalent to PLATFORM_TICK_CPU_TICKLESS_THRESH (default: 1), otherwise normal mode.

Table 2-4. Tick-less modes PLATFORM_TICK_POWERSAVE

Tick-less mode reduces power consumption, but the system may have more latency when it reacts to external events, and time flow is less accurate.

Power Save Mode	PMU Timer	CPU Timer
PLATFORM_MCU_POWERSAVE_MODE_SLEEP This is default mode. Main purpose is to have reduced power consumption without clocks running (when PMU timer used).	No clocks are requested when CPU idle. Application power switches are forced to be ON. When CPU is idle. Peripheral devices which require HT and ALP clocks may not work. It is the responsibility of the device driver to request the required clock and release as soon as it can. Alternatively, the driver or application can disable power saving temporarily, which switches to CPU timer. Applications do not go to Deep Sleep.	HT clock (and ALP due to dependency) is requested always (forced via ClockControlStatus register). Application power switches are forced to be ON.
PLATFORM_MCU_POWERSAVE_MODE_SLEEP_WITH_ALP Main purpose is to have reduced power consumption with ALP clock running (when PMU timer used).	ALP clock is requested when CPU idle (forced via ClockControlStatus register). Application power switches are forced to be ON. When CPU is idle. Peripheral devices which require HT clock may not work. It is the responsibility of the device driver to request the required clock and release as soon as it can. Alternatively, the driver or application can disable power-saving temporarily, which switches to CPU timer Applications do not go to Deep Sleep.	HT and ALP clocks are requested always (forced via ClockControlStatus register). Application power switches are forced to be ON.
PLATFORM_MCU_POWERSAVE_MODE_DEEP_SLEEP Main purpose is to have the application domain switched OFF completely (when PMU timer used).	No clocks are requested when CPU idle. Application power switches are NOT forced to be ON. As soon as CPU becomes idle (execute WFI instruction) because of the threads sleeping, applications may enter Deep Sleep mode. If power saving is not enabled (via platform_mcu_powersave_enable()) or requested sleep time is less than PLATFORM_TICK_PMU_DEEP_SLEEP_THRESH (default: 100), then CPU will not execute the WFI instruction and will not enter Deep Sleep.	HT clock is requested always (forced via ClockControlStatus register).

Table 2-5 Power Save Modes

All CPU timer modes are essentially the same, regardless of power save mode. When the CPU timer is involved, applications consume maximum power and do not go to Deep Sleep. Applications can quickly switch between timers within the same power save mode.

Definitions	What it does	When it is set
PLATFORM_POWERSAVE_DEFAULT	Enables or disables all power save features	Should be defined in the application *makefile*
PLATFORM_CORES_POWERSAVE	Enables or disables power save optimizations related to various cores. (GMAC, I2S, USB) enables memory clock gating. (defined in *platform_mcu_powersave.c*)	PLATFORM_POWERSAVE_DEFAULT
PLATFORM_APPS_POWERSAVE	Enables or disables application domain power saving	PLATFORM_POWERSAVE_DEFAULT
PLATFORM_WLAN_POWERSAVE	If Wi-Fi firmware is not downloaded to WLAN, applications will put WLAN to lowest power mode.	!(WICED_NO_WIFI) NO_WIFI_FIRMWARE: YES
PLATFORM_TICK_POWERSAVE	Ticks power save feature enable or disable	If PLATFORM_APPS_POWERSAVE is defined
PLATFORM_MCU_POWERSAVE_INIT_STATE	Defines MCU power save mode set during platform initialization	Defined as PLATFORM_MCU_POWERSAVE_MODE_SLEEP
PLATFORM_TICK_STATS	if PLATFORM_TICK_POWERSAVE && PLATFORM_TICK_STATS powersave_stat_avg_sleep_ticks, powersave_stat_avg_run_ticks, avg_load = run_ticks/total_ticks req, adj = How many sleep ticks are requested and how many are given call_no : How many times goes to suspend power_down_perm : How many times power down is permitted powersave_disable_counter: platform_mcu_powersave_enable()/disable() #if PLATFORM_WLAN_POWERSAVE_STATS powersave_wlan_res_call_counter platform_wlan_res_up_begin_stamp platform_wlan_res_up_time platform_wlan_res_wait_up_time	
PLATFORM_TICK_POWERSAVE_MODE_INIT	Any value from Table 2-4 (for defaults, see the next column)	PLATFORM_TICK_POWERSAVE_MODE_TICKLESS_IF_MCU_POWERSAVE_ENABLED
PLATFORM_CPU_CLOCK_FREQUENCY	PLATFORM_CPU_CLOCK_FREQUENCY_320_MHZ	

Table 2-6 CPU Timer Modes

2.3.6 CYM4390x: Example Applications for Power Save

2.3.6.1 Application Sleep (Deep Sleep)

<WICED_Studio>/apps/test/apps_sleep

In this example, the application:

- Periodically goes to Deep Sleep in a co-operative way
- Wakes up through timer
- Saves variables in AONRAM and verifies them after warm boot
- Registers event handlers for entering and leaving Deep Sleep
- Does not load firmware to WLAN side

Note: Application sleep is woken up by the system timer thread. To make the application sleep longer, either disable watchdog (define `WICED_DISABLE_WATCHDOG` in application *makefile*) or increase system timer timeout (modify `DEFAULT_SYSTEM_MONITOR_PERIOD` in *system_monitor.c*).

Figure 2-5. Apps_sleep.c: Power Consumption on BCM943907WCD2 board ~375 uA (in Deep Sleep)

2.3.6.2 Ping Deep Sleep (Deep Sleep + Networking)

<WICED_Studio>/apps/snip/ping_deepsleep

In this example, the application:

- Connects to AP and pings AP periodically
- Periodically goes into Deep Sleep in a Co-operative way
  - Checks for networking to be idle
  - Suspends networking stack
- Resumes networking after coming out of Warm boot

Figure 2-6. Ping Deep Sleep: Average power consumption ~1.5 mA

2.3.6.3 Ping Power Save

<WICED_Studio>/apps/snip/ping_powersave/ping_powersave.c

In the low-power networking demo, the system connects to AP and periodically pings the AP. In between pings, the system suspends network timers and enters low-power mode (Sleep) for `WIFI_SLEEP_TIME` seconds. The system resumes network timers when coming out of Sleep.

Figure 2-7. Ping_powersave: Average current consumption 52.6 mA

2.3.6.4 Powersave (Deep Sleep + Sleep Use Cases)

<WICED_Studio>/apps/test/powersave/powersave.c

- Console-based application
- Commands can be
  - Entered using UART console
  - Statically compiled
- Following features are supported

```
/*
* WLAN is not associated with AP.
* APPS go to deep-sleep for long period, wake-up for short period, then again go to deep-
sleep for long period.
 */
#define POWERSAVE_STANDALONE_TEST_DEEPSLEEP_NO_ASSOC              0
```

```
/*
 * WLAN is associated with AP.
 * APPS go to deep-sleep for long period, wake-up for short period, then again go to deep-
sleep for long period.
 */
#define POWERSAVE_STANDALONE_TEST_DEEPSLEEP_ASSOC
```

```
/*
* WLAN is associated with AP.
* APPS is waiting packets from WLAN. APPS is powered on but no clocks running.
       * When WLAN pass packet to APPS the APPS wake-up and HT clock requested, after
packet handled clocks dropped and wait started again.
*/
#define POWERSAVE_STANDALONE_TEST_WAIT_FOR_WLAN                    0
```

```
/*
* WLAN is associated with AP.
* APPS is waiting packets from WLAN.
 * APPS is powered on, ALP clock available, bus is on ILP clock (so UART and interrupts
are working).
* When WLAN pass packet to APPS the APPS wake-up and HT clock requested, after packet
handled HT clock dropped.
*/
#define POWERSAVE_STANDALONE_TEST_ACTIVE_WAIT_FOR_WLAN            0
```

```
/*
* WLAN is associated with AP.
* APPS is running iPerf UDP server.
* APPS is powered on, HT clock available, bus is running on ALP clock.
* APPS backplane and CPU frequency reduced to 80MHZ.
*/
#define POWERSAVE_STANDALONE_TEST_LOW_POWER_NETWORKING            0
```

```
/*
* WLAN is not associated with AP.
       * APPS go to deep-sleep for short period, wake up and immediately go to back to
deep-sleep, repeat this cycle continuously.
* For more precise profiling may worth to go to UART driver and make TX function (e.g.
uart_slow_transmit_bytes()) do nothing.
*/
#define POWERSAVE_STANDALONE_TEST_WAKEUP_FROM_DEEP_SLEEP_PROFILE  0
```

2.3.7 BT + 4390x Low Power

<WICED_Studio>/apps/snip/bluetooth/ble_hello_sensor/ on CY943907WAE3 board

The Bluetooth Low Energy (BLE) hello sensor application initializes the BT stack and advertises periodically. A BLE-aware application (for example, LightBlue application running on Ipad) can connect to the device.

By default, Deep Sleep mode is enabled in *ble_hello_sensor.c* for CY943907WAE3 platform.

Figure 2-8. Example of Going to Deep Sleep and Waking Up Periodically

Figure 2-8 shows *Ble_hello_sensor.c* going to Deep Sleep and waking up periodically with Deep Sleep current of ~691 uA and average current 14.2 mA on a CY943907WAE3 board.

For power consumption in Sleep mode, modify *ble_hello_sensor.mk* to add the following:

```
GLOBAL_DEFINES    += PLATFORM_MCU_POWERSAVE_MODE_INIT=PLATFORM_MCU_POWERSAVE_MODE_ _SLEEP
```

Figure 2-9. Example of Going to Sleep and Waking Up Periodically

Figure 2-9 shows *Ble_hello_sensor.c* going to sleep and waking up periodically with sleep current of ~9.4 mA and average current of 22.78 mA on a CY943907WAE3 board.

3 Example Power Measurements

The plots provided in this section show various aspects of the power consumption using the WICED Ping Powersave snippet application running on a BCM943362WCD4 WICED module acting as a Wi-Fi station. The application pings the gateway at one second intervals with Wi-Fi power save enabled. The RTOS/TCP stack used to take the measurements is FreeRTOS/LwIP, however ThreadX/NetX and ThreadX/NetXDuo yield virtually identical results. All plots show current consumption separated out for the BCM43362 Wi-Fi chip and STM32F205 MCU. The power supply to the module is 3.3 V.

3.1 The Big Picture

Figure 3-1 is a 5-second capture of current consumed when the application sends an ICMP ping to the AP at intervals of approximately 1 second. The large blue spikes are Wi-Fi packet transmissions related to the Ping transmission. The smaller blue spikes at intervals of approximately 300 ms occur when the Wi-Fi wakes to listen to the AP for a DTIM beacon.

Red spikes indicate when the MCU is awake. Nearly all MCU wakeups coincide with the transmission of a ping packet. The red spike coinciding with the small blue spike just after 4000 ms is a result of the Wi-Fi chip waking the MCU to pass a received packet to the network stack, in this case possibly an ARP packet.

Figure 3-2 shows the code used to take the following measurement.

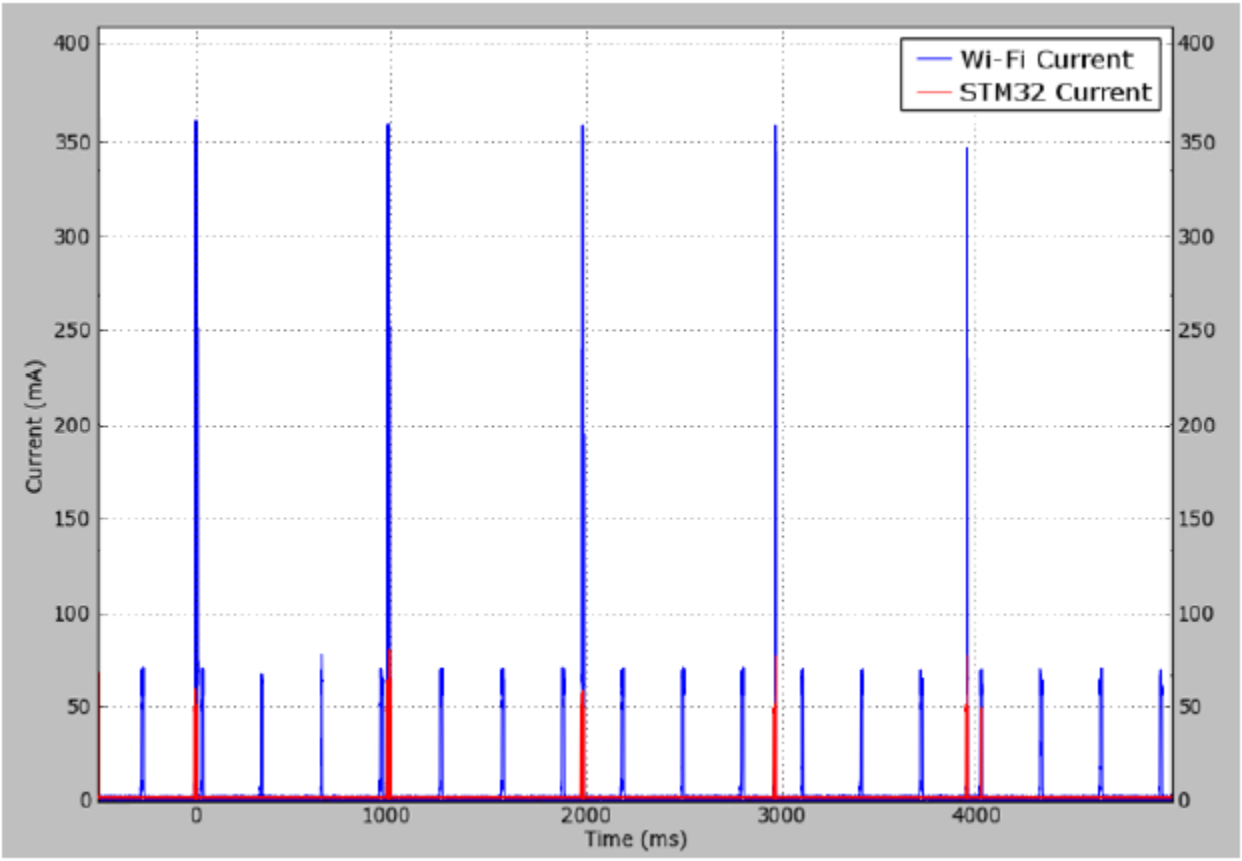

Figure 3-1. Current Consumption with the Ping Power Save application, AP DTIM=3

```
void application_start(void)
{
    wiced_init();  /* Initialise the WICED device */
    wiced_network_up( WICED_STA_INTERFACE, WICED_USE_EXTERNAL_DHCP_SERVER, NULL ); /*Connect*/
    wiced_platform_mcu_enable_powersave();            /* Enable MCU powersave */
    wiced_wifi_enable_powersave_with_throughput(10);  /* Enable Wi-Fi powersave */

    while (1)
    {
        send_ping();                    /* Send an ICMP ping to the gateway */
        wiced_network_suspend();        /* Suspend all network activity, including timers */
        wiced_rtos_delay_milliseconds( WIFI_SLEEP_TIME );  /* Sleep for a while */
        wiced_network_resume();         /* Resume network activity and restart timers */
    }
}
```

Figure 3-2. Representative Ping Power Save Application Source

3.2 DTIM Wake-up

Figure 3-3 shows the closer view of the Wi-Fi chip (blue trace) waking at an interval of 300 ms to receive the AP DTIM beacon. The calls to network suspend and resume have been removed for this plot, and a 100 ms IGMP timer that causes the MCU to wake (red trace) is now present.

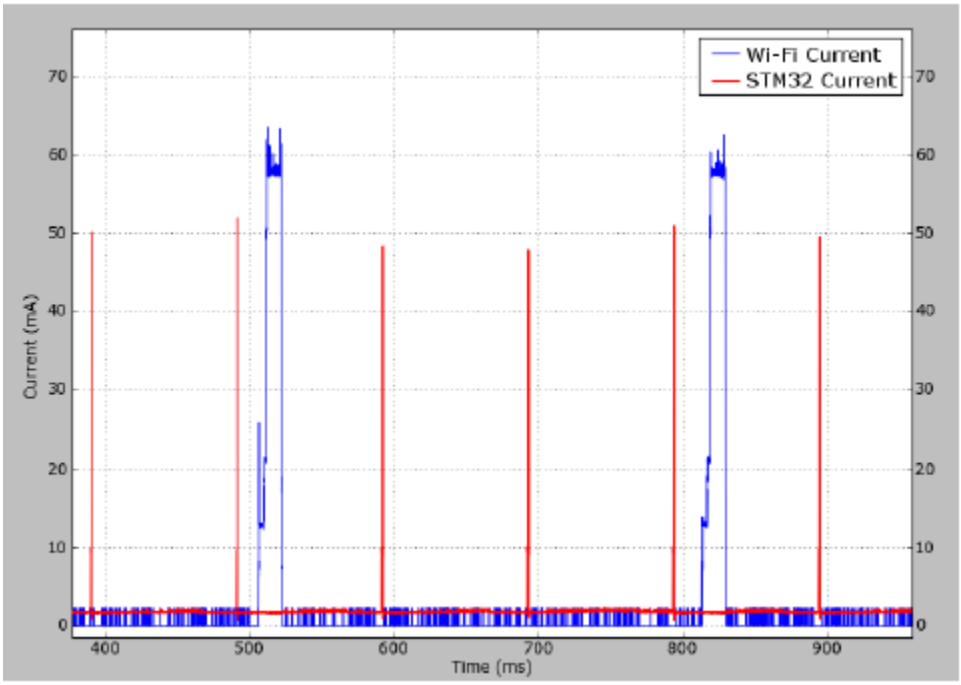

Figure 3-3. Current Consumption for DTIM Beacon Reception

3.3 Ping Transaction

Figure 3-4 shows the detailed current consumption of a ping transaction. The MCU wakes to send a ping packet. Once awake, it attempts to wake the Wi-Fi chip by polling the SDIO bus. With both the MCU and Wi-Fi chip awake, the transaction begins.

There are four packets transmitted by the Wi-Fi chip, the first is a Null-function Data frame, which indicates to the AP that the Wi-Fi station is awake. The second is the ICMP Ping packet. The third is an 802.11 acknowledgement packet for the ping reply, and the final packet is another Null-Function Data frame to indicate to the AP that the station is about to return to sleep.

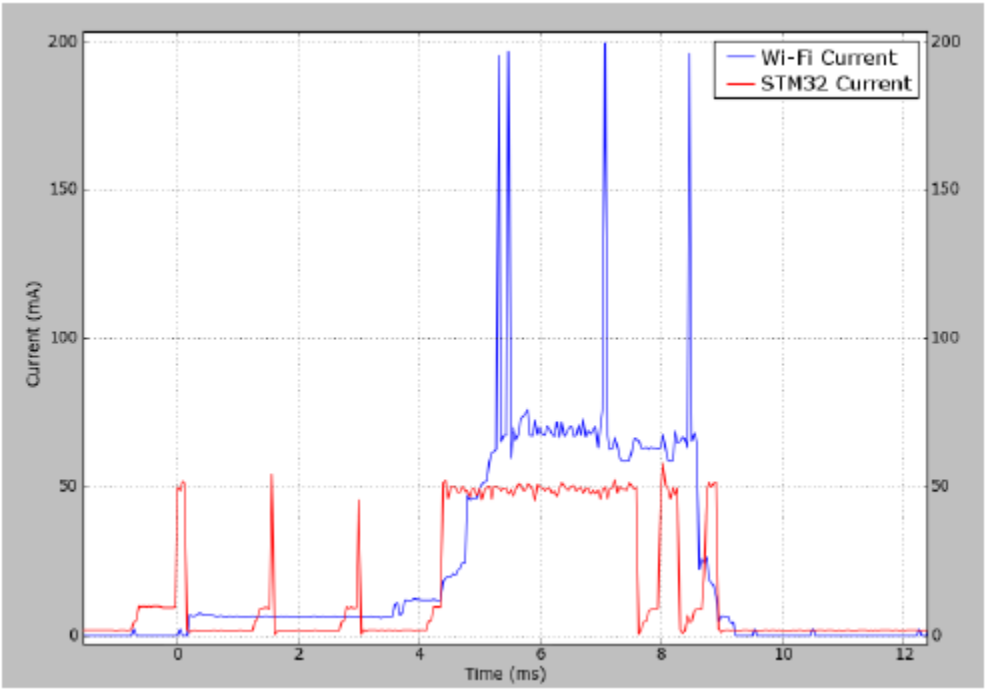

Figure 3-4. Current Consumption for a Ping Transaction

4 Low-Power Measurement Techniques

Measuring current consumption over a large dynamic range is difficult without the use of expensive instrumentation. Current consumed during the transmission of a Wi-Fi packet may be 300 mA or more, and current consumed during Sleep mode may be 10's of microamperes or less. The dynamic range of the measurement covers nearly six orders of magnitude with microsecond switching times.

Most WICED evaluation boards include a current measurement circuit that is accurate for high range (>5 mA) measurements. Much can be learned about the operation of an application by observing the current profile, and the circuit on the evaluation board provides an effective means to do this.

Measuring low-range sleep current is more challenging since the current measurement circuit requires a larger resistor in series with the power supply to the device under test. While the current draw is low, there is no issue, but when the board draws a large current (to transmit a WLAN packet, for instance), a large voltage drop occurs across the series resistor and the power supply browns out or fails entirely.

To complicate the measurement further, bulk capacitance provides short-term power to the board during high current loads. In very low-power Sleep modes, it may take seconds or even minutes for the energy in the bulk capacitance to dissipate. Energy supplied from the bulk capacitance is typically not measured by the low range current measurement circuit, since bulk capacitance is necessarily placed very close to the Wi-Fi subsystem. The energy provided by the bulk capacitance does not pass through the current measurement circuit.

4.1 Measuring Low-Range Sleep Current

To measure low-range sleep current using a WICED BCM943362WCD4_EVB evaluation board, the following items are needed:

- A custom low-power application
- Minor hardware modifications to the WICED evaluation board and WICED module
- Configuration of the BCM943362WCD4 platform
- A multimeter capable of measuring current in the milliamp and microamp range
- The correct measurement technique

4.1.1 Custom Low-Power Application 1: ping_powersave

The goal of the custom application is to make the period between large current spikes as long as possible. This provides time for the energy stored in any bulk capacitance to discharge so that the current draw from the DUT can reach a low-power steady state.

The WICED Studio ping power save snippet application is a good example to start with. The application should be modified as follows to minimize the MCU and Wi-Fi power consumption:

- Ensure the MCU and Wi-Fi power save API calls have not been commented out or removed, and check that MCU power save has not been disabled in the *wiced_defaults.h* header file.
- Before the main `while(1)` loop, add a call to the listen interval API function to force the Wi-Fi chip to stay in Sleep mode for periods of at least 25 seconds (assumes AP beacons are a typical 100ms apart) :

  ```
  wiced_wifi_set_listen_interval(250, WICED_LISTEN_INTERVAL_TIME_UNIT_BEACON);
  ```

- Near the top of *ping_powersave.c*, change the following directive to set the ping interval to 25 seconds: `#define WIFI_SLEEP_TIME` to `(25000 * MILLISECONDS)`

The typical client association timeout limit for most APs is 60 seconds, so it is important to check-in with the AP well within this timeframe to avoid being disassociated.

- Open the file *<WICED-Studio>/include/platforms/BCM943362WCD4/platform.h* and scroll to the bottom. Ensure the following directive is correctly set to use the STM32 MCO oscillator:

  `#define-WICED_WLAN_POWERSAVE_CLOCK_SOURCE--WICED_WLAN_POWERSAVE_CLOCK_IS_MCO`.

  In power save mode, the Wi-Fi chip requires a 32-kHz clock input to stay in sync with beacons from the AP. The MCO oscillator is used to provide the necessary 32-kHz clock when the STM32 is in low-power Stop mode.

  An STM32 timer may alternately be used to provide a 32-kHz signal using a pulse-width modulated (PWM) output. However, the STM32 cannot use MCU power save mode if a timer is used, since the timer powers down and the 32-kHz clock stops.

4.1.2 Custom Low-Power Application 2: ble_hello_sensor

This application runs on BCM94343WWCD1 board, use the following WICED build string to build and download the application on the board

snip.bluetooth.ble_hello_sensor-BCM94343WWCD1 download download_apps run

After the application starts, device enters low power mode, but BT keeps advertising, once BCM94343WWCD1 is connected to a device using BT (for example: lightblue app in ios), it exits low power mode, and stays in active mode till the connection is disconnected, when the app again enters low power mode.

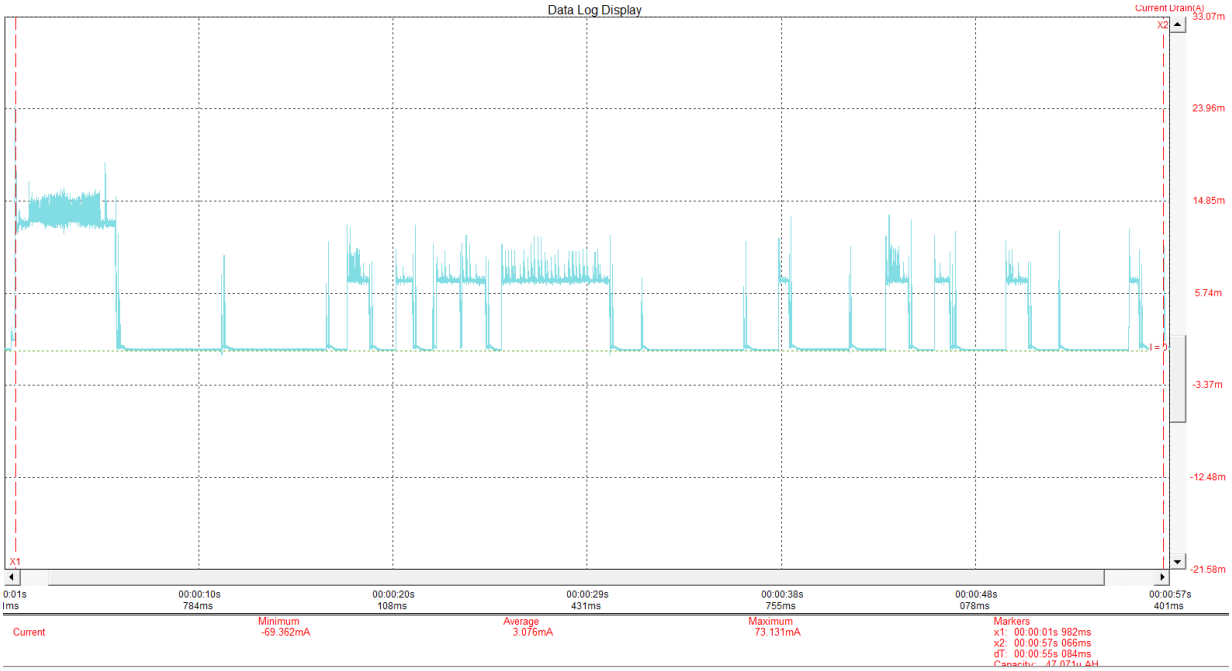

Figure 4.1 Graph of power consumption with ble_hello_sensor application on BCM94343WWCD1 board, sleep current ~1mA, average current ~3mA

4.1.3 WICED Module Modifications

The following modification is ONLY required for P100 and P200 revisions of the BCM943362WCD4 module.

Locate the STM32F205 MCU and short Pin 41 (PA8) to Pin 44 (PA11). A very limited number of STM32 pins are connected to an internal 32-kHz low-power MCO oscillator that remains operational in Sleep mode. Pin 41 may be configured to connect to the internal oscillator, Pin 44 may not. Pin 44 is however connected to the BCM43362 external sleep clock input pin. Bridging Pin 41 and Pin 44 connects the BCM43362 sleep clock input to the STM32 32-kHz low-power clock output.

Before proceeding, download and test the application works correctly by looking at the application current profile using an oscilloscope connected to the current measurement circuit on the evaluation board. After the WICED module connects to the AP, the current should be almost always near zero, with a spike once every 25 seconds when the ICMP ping is sent.

If the application fails to stay connected and continually transmits 802.11 probes, the WLAN may not be able to correctly keep in sync with beacons due to a poor timing reference. In this situation, it is possible the 32-kHz sleep clock is not connected to the BCM43362 or is not working properly.

Debug this issue before proceeding if necessary.

4.1.4 WICED Evaluation Board Modifications

Locate and desolder the 0805-sized SMT resistor R14. R14 is a 0.2-ohm resistor in series with the WICED module power supply.

Connect one lead of the multimeter to the V_{DD}_3V3 pad of R14, and connect the other lead of the multimeter to the V_{DD}_3V3_WIFI pad of R14. The multimeter is now connected in series with the WICED module power supply.

4.1.5 Measurement Technique

Follow these steps to measure low-power current:

1. On the multimeter, select the low-range current mode (microamps to milliamps).
2. Connect the multimeter leads across the location of R14, one lead on each pad of R14.
3. Apply a short across the location of R14. One technique to achieve this is to solder a 2-pin header across the location of R14, and place a jumper on the header.
4. While watching the UART output on the evaluation board, press **Reset** and wait for a print indicating a successful ping reply. An ICMP ping transaction occurs approximately at every 25 seconds.
5. Wait for 1 to 2 seconds after a print indicating a successful ping reply, then remove the short (jumper) across the location of R14.
6. The multimeter current settles to the minimum sleep current of approximately 500 microamps. Note that an intermittent current blip may also be observed at every 4 or 5 seconds. This current blip corresponds to an MCU timer firing which may briefly wake the MCU.
7. After making the measurement (within ~20 seconds), replace the short across the location of R14. If the WICED module attempts to transmit or transition into a high-power mode while the multimeter is in the low-range current mode, the module will almost certainly brown out. Additionally, there is some chance of blowing a fuse in the multimeter.

References

The references in this section may be used with this document.

Note: Cypress provides customers access to technical documentation and software through the WICED website (community.cypress.com/). Additional restricted material may be provided through the Customer Support Portal (CSP) and Downloads.

For Cypress documents, replace the 'xx' in the document number with the largest number available to ensure you have the most current version of this document.

Document (or Item) Name	Number	Source
[1] 802.11 Wireless Networks "The Definitive Guide"	2nd Edition, April 2005 Matthew S. Gast	O'REILLY
[2] WICED Quick Start Guide	WICED-QSG2xx-R	WICED Studio
[3] STM32F20x Reference Manual	CD00225773	ST website
[4] AT91SAM4S Datasheet		Atmel website

3.

Document Revision History

Document Title: WICED™ Development Power Save System

Document Number: 002-19004

Revision	ECN	Issue Date	Description of Change
**	5861835	08/23/2017	Created spec.

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

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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