Low-power system design with CYW43012 and PSoC 6 MCU

About this document

Scope and purpose

AN227910 describes how to use CYW43012 and PSoC 6 MCU to design a low-power connectivity solution for IoT applications.

The 28-nm radio combined with a 40-nm PSoC 6 MCU enables an ultra-low-power platform for IoT applications. This application note provides an overview of low-power modes and features in the CYW43012 device and describes various techniques such as host offload features to optimize power consumption in the system, assisted with Cypress' Low Power Assistant tool.

More code examples? We heard you.

To access an ever-growing list of hundreds of PSoC code examples, please visit our **code examples web page**. You can also explore the video training library **here**.

Table of contents

Table of contents

1 Introduction

Infineon **CYW43012** is a 28-nm, ultra-low-power device that supports single-stream, dual-band IEEE 802.11n-compliant and 802.11ac-friendly Wi-Fi MAC/baseband/radio and Bluetooth 5.0 BR/EDR/LE. The WLAN section supports an SDIO interface to the host MCU (PSoC 6 MCU), and the Bluetooth section supports a high-speed 4-wire UART interface to the host MCU.

PSoC 6 MCU bridges the gap between expensive, power-hungry application processors and low-performance microcontrollers (MCUs). The ultra-low-power PSoC 6 MCU architecture offers the processing performance needed by IoT devices, eliminating the tradeoffs between power and performance. The PSoC 62 series, built on an ultra-low-power 40-nm platform, complements the CYW43012 device, providing an ultra-low-power host interface to the Wi-Fi and Bluetooth radio available in CYW43012.

This application note discusses various low-power design techniques along with low-power features offered by the Infineon CYW43012 + PSoC 6 MCU platform and how to use them in your application. This application note requires a basic knowledge of the CYW43xx and PSoC 6 MCU architecture, and the ability to develop applications using Arm® Mbed™ OS and ModusToolbox™. If you are new to CYW43xx, PSoC 6 MCU, or Mbed OS, see the **Getting Started with PSoC 6 MCU and CYW43xxx in Mbed OS**.

2 Low-power overview

2.1 CYW43012 power modes

CYW43012 has been designed with the stringent power consumption requirements of battery-powered IoT devices in mind. **Table 1** lists the power modes supported in CYW43012.

Table 1 Power modes in CYW43012

Power mode	Description
Active	• All WLAN blocks in CYW43012 are powered up and fully functional with active carrier sensing and frame transmission and receiving. • All required regulators are enabled and put in the most efficient mode based on the load current. • Clock speeds are dynamically adjusted by the PMU sequencer.
Deep sleep (DS0)	• The radio, analog domains, and most of the linear regulators are powered down. The rest of CYW43012 remains powered up in an idle/retained state. • All main clocks (PLL, crystal oscillator, or TCXO) are shut down to reduce active power to the minimum level. The 32.768-kHz LPO clock is available only for the PMU sequencer. This is necessary to allow the PMU sequencer to wake up the chip and transition to active mode. • Complete RAM needed for WLAN firmware is retained.
Power down (Off)	• CYW43012 is effectively powered OFF by shutting down all internal regulators. The chip is brought out of this mode by external logic re-enabling the internal regulators.

CYW43012 includes an advanced WLAN power management unit (PMU) sequencer, which provides significant power savings by putting the CYW43012 device into various power management states appropriate to the current environment and activities that are being performed.

The PMU enables and disables internal regulators, switches, and other blocks based on the computation of the required resources and a table that describes the relationship between resources and the time needed to enable and disable them. Configurable, free-running counters (running at 32.768 kHz LPO clock) in the PMU sequencer are used to turn ON/turn OFF individual regulators and power switches. Clock speeds are dynamically changed (or gated altogether) for the current mode. Slower clock speeds are used wherever possible. See the **CYW43012 datasheet** for details on the PMU.

2.2 PSoC 6 MCU power modes

PSoC 6 MCU features seven power modes that are split into system modes that affect the whole device, and standard Arm® CPU modes that affect only one CPU. The system power modes are Low-Power (LP), Ultra-Low-Power (ULP), deep sleep, and hibernate. The Arm CPU power modes are active, sleep, and deep-sleep, and are available in system LP and ULP power modes. See **AN219528** for details on PSoC 6 MCU low-power modes and power reduction techniques. **Table 2** summarizes PSoC 6 MCU power modes and provides a simplified version for representing power mode transitions in a PSoC 6 MCU and CYW43012 design (**Figure 1**).

Table 2 Power modes in PSoC 6 MCU

Power mode	Description	Simplified power mode
System LP	• All resources are available with maximum power and speed. • All CPU power modes supported.	Active mode
System ULP	• All blocks are available, but core voltage lowered resulting in reduced high-frequency clock frequencies. • All CPU power modes supported.	
CPU active	• Normal CPU code execution • Available in system LP and ULP power modes	
CPU sleep	• CPU halts code execution. • Available in system LP and ULP power modes	Sleep mode
CPU deep sleep	• CPU halts code execution. • Requests system deep sleep entry • Available in system LP and ULP power modes	Deep sleep
System deep sleep	• Occurs when all CPUs are in CPU deep sleep • CPUs, most peripherals, and high-frequency clocks are OFF. • Low frequency clock is ON. • Low-power analog and some digital peripherals are available for operation and as wakeup sources. • SRAM is retained.	
System hibernate	• CPUs and clocks are OFF. • GPIO output states are frozen. • Low-power comparator, RTC alarm, and dedicated WAKEUP pins are available to wake up the system. • Backup domain is available.	Hibernate

2.3 CYW43012 power-related hardware signals

CYW43012 has six hardware signals that provide the power control interface to the host (PSoC 6 MCU).

Table 3 Hardware signals in CYW43012

Signal	Description
WL_REG_ON	This signal is used by the PMU (with BT_REG_ON) to power up the WLAN section. It is ORed with the BT_REG_ON input to control internal CYW43012 regulators. When this pin is HIGH, the regulators are enabled, and the WLAN section is out of reset. When this pin is LOW, the WLAN section is in reset. If BT_REG_ON and WL_REG_ON are both LOW, the regulators are disabled.
BT_REG_ON	This signal is used by the PMU (with WL_REG_ON) to decide whether to power down internal CYW43012 regulators. When this pin is HIGH, the regulators are enabled, and the Bluetooth section is out of reset. When this pin is LOW, the Bluetooth section is in reset. If BT_REG_ON and WL_REG_ON are LOW, the regulators will be disabled.
BT_DEV_WAKE	Bluetooth device wakeup: Signal from the host (PSoC 6 MCU) to CYW43012 indicating that

Low-power overview

Signal	Description
	the host requires attention like sending data to the Bluetooth controller. • Asserted: The Bluetooth device must wake up or remain awake. • Deasserted: The Bluetooth device may sleep when all other sleep criteria are met. The polarity of this signal is software-configurable and can be asserted HIGH or LOW. Default is asserted HIGH.
BT_HOST_WAKE	Host wakeup signal from the CYW43012 Bluetooth controller to the host indicating that the CYW43012 Bluetooth controller has data for the host. • Asserted: host device must wake up or remain awake. • Deasserted: host device may sleep when all other sleep criteria are met. The polarity of this signal is software-configurable and can be asserted HIGH or LOW. Default is asserted HIGH.
WL_HOST_WAKE	Host wakeup signal from the CYW43012 WLAN device to the host indicating that the CYW43012 WLAN device has data for the host to process. This signal is an active HIGH signal. • HIGH: host device must wake up or remain awake. • LOW: host device may sleep when sleep criteria are met.
WL_DEV_WAKE	WLAN Device wakeup signal. This signal is not used in current designs and SDIO-based wakeup is used to the wakeup the WLAN device.

2.4 Power mode transition

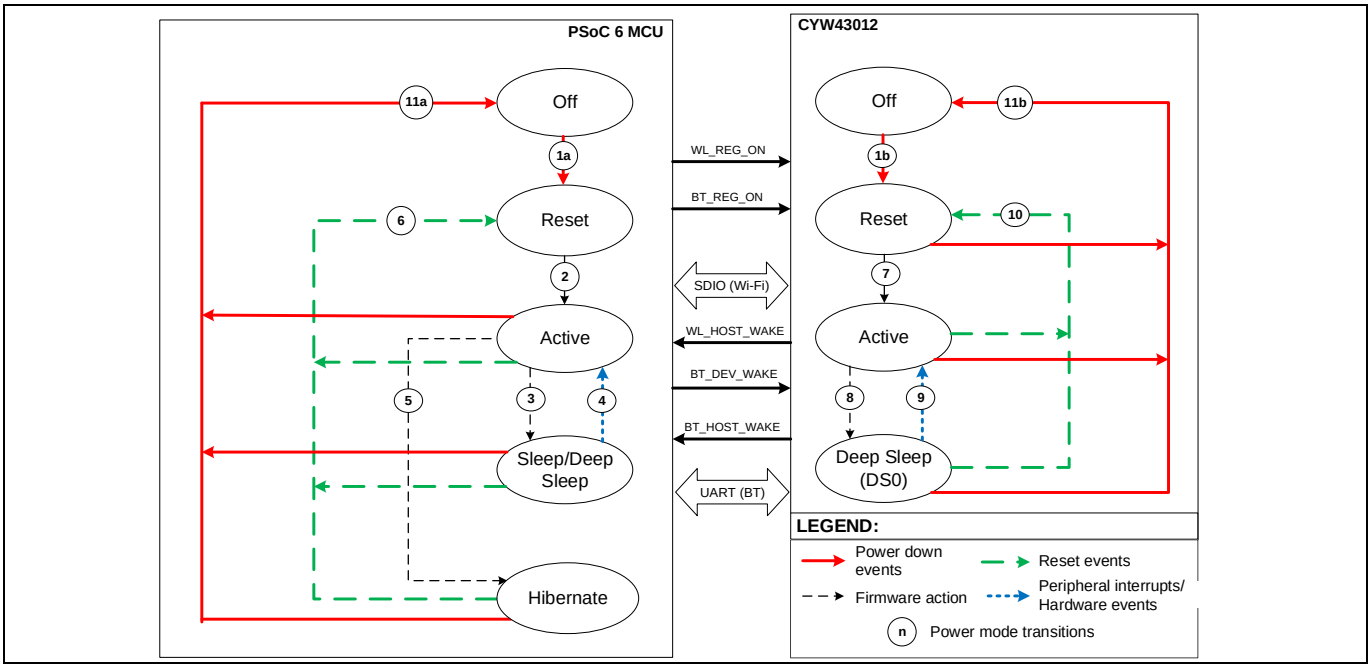

Figure 1 Simplified power mode transitions in "PSoC 6 MCU[a] + CYW43012" design

Transition	Description	Trigger
1a	PSoC OFF to reset transition	Power supply applied to PSoC 6 MCU.
1b	CYW43012 OFF to reset transition	Power supply applied to CYW43012.

Low-power overview

Transition	Description	Trigger
2	PSoC reset to active transition	Firmware (boot/startup) transitions from reset to active mode.
3	PSoC active to sleep or deep sleep transition	Firmware (OS idle task) transitions from active to sleep or deep sleep mode.
4	PSoC sleep or deep sleep to active transition	Wakeup (interrupt) from sleep/deep sleep; typical sources in a CYW43012 design: • WL_HOST_WAKE (deep sleep or sleep) • BT_HOST_WAKE (deep sleep or sleep) • SDIO and UART interrupt (sleep) • Low-power timer interrupts
5	PSoC active to hibernate transition	Firmware (enter hibernate) transitions from active to hibernate mode
6	PSoC transitions to reset state	Any reset event including hibernate wakeup, external reset etc.
7	CYW43012 reset to active transition	External events transition CYW43012 out of reset; either BT_REG_ON or WL_REG_ON pulled HIGH by PSoC 6 MCU.
8	CYW43012 active to deep sleep transition	CYW43012 enters DS0 mode; managed by on-chip PMU and requires enabling power save in the device
9	CYW43012 deep sleep to active transition	External events or PMU transitions CYW43012 to active mode; possible PSoC 6 MCU actions: • BT_DEV_WAKE • SDIO interrupt
10	CYW43012 transitions to reset state	External events transition CYW43012 to reset state; both WL_REG_ON and BT_REG_ON are pulled LOW by PSoC 6 MCU.
11a	PSoC transitions to OFF state	Power supply removed from PSoC 6 MCU (Power Down)
11b	CYW43012 transitions to OFF state	Power supply removed from CYW43012 (Power Down)

Note[a] PSoC 6 MCU power modes are represented in a simplified form for ease of depicting the transitions. See **Table 2** for details.

3 WLAN power optimization techniques

3.1 IEEE 802.11 (Wi-Fi) power saving

Topics in this section are partly based on information contained in IEEE **802.11 specification document**. 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.

This section only provides an overview of power saving methods and associated terms from the IEEE specification. If you are already familiar with the topic and interested in the implementation, refer to **Wi-Fi power save implementation**.

3.1.1 Beacon Interval

Beacon frames are periodic frames broadcast by the Wi-Fi access points (APs) to communicate throughout the serviced area the characteristics of the connection offered to the Wi-Fi stations (STAs). This information is used by STAs trying to connect to the network as well as STAs already associated to the Basic Service Set (BSS). The period at which beacon frames are transmitted is called “Target Beacon Transmission Time” (TBTT) or simply “Beacon Interval”. even though the beacon interval is configurable for a given AP, it is typically set to 102.4 ms (100 time units, where 1 time unit is 1024 µs per 802.11 specification). The beacon interval information is available as part of the beacon frame itself. See the **802.11 specification** for details. In addition, each beacon includes a Traffic Indication Map (TIM) that contains information regarding packets buffered for STAs. It is the responsibility of the STAs to read the TIM and retrieve the packets destined for them.

3.1.2 Listen Interval

To save power, STAs sleep by powering down most of the Wi-Fi subsystem. While the STAs are asleep, APs buffer frames for them and indicate the availability through beacon TIMs. The sleeping STAs periodically wake up to listen to traffic announcements (TIMs) and determine whether the AP has any buffered frames. When STAs 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 save station. Larger listen intervals require more AP memory for frame buffering.

In reality, APs generally do not consider the listen interval requested by a STA. If a STA 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. In addition, if the AP supports Delivery Traffic Indication Map (DTIM), that setting may override the listen interval requested by STAs.

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.

3.1.3 DTIM Period

The DTIM period is a parameter associated with an infrastructure network and is advertised in the AP Beacon frame. The polled approach using standard TIMs 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 interval.

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

It should be noted that the listen interval is recommended to be equal to (or less than) the DTIM period for the STA to receive all unicast, multicast, and broadcast packets. This ensures an optimal power and latency tradeoff.

Usually, the DTIM period is set as the number of beacon intervals in the AP. The default DTIM period for most APs is either DTIM=1 (DTIM every beacon), DTIM=3 (DTIM every third beacon), or DTIM=10. 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.

Figure 2 explains the beacon interval, listen interval, and DTIM period in an infrastructure network.

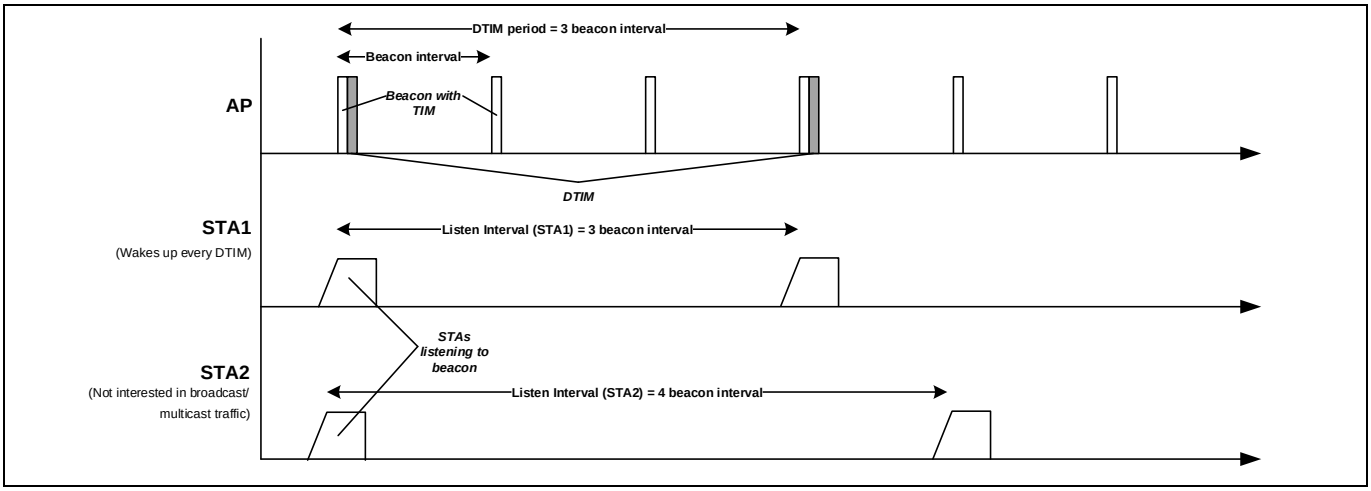

Figure 2 802.11 infrastructure network operation

3.1.4 Power saving methods

Although 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.

Wi-Fi STAs use different mechanisms to save power. STAs 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. STAs 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.

Note: STAs requiring a more fine-grained power save mechanism to differentiate power save for various traffic priorities may consider using Unscheduled Automatic Power Save Delivery (U-APSD) or Wi-Fi Multimedia (WMM) power save. CYW43012 currently does not support WMM power save.

3.1.4.1 Power Save Poll

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

1. STA sends a frame to the AP with the Power Management bit set and then goes to sleep.
2. AP notes that 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 frames. 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 that it has buffered frames 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.
7. When the STA needs to transmit data, it can wake up and transmit data anytime without waiting for the beacon interval as AP is active all the time.

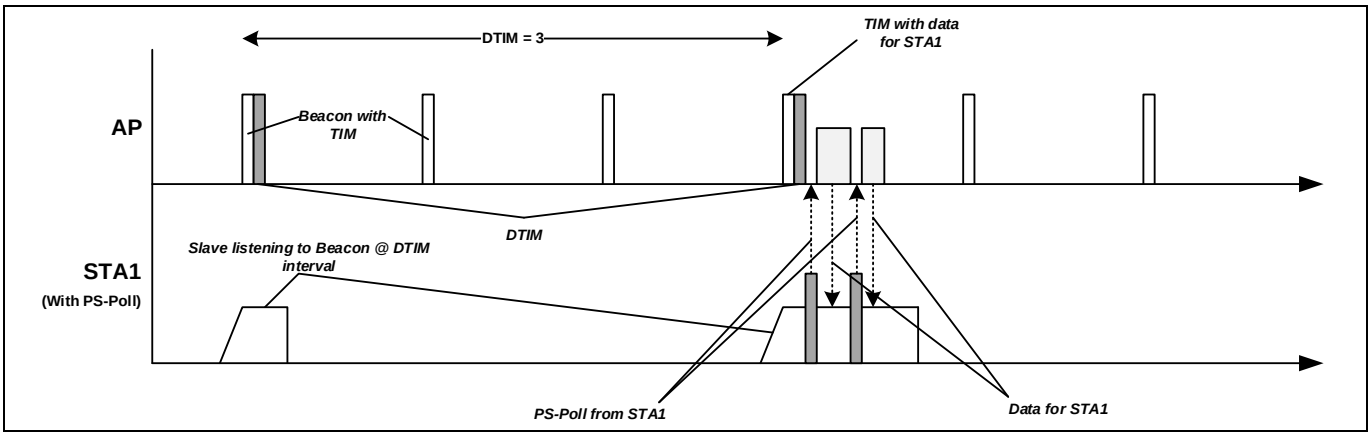

Figure 3 Power Save Poll

3.1.4.2 802.11 Power Save Without Poll

A non-standard mechanism known broadly as 802.11 Power Save without Poll (PS-non-Poll) enables STAs 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 that 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 frames 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 that the STA is awake and sends buffered frames.
6. STA receives the frames, waits for a timeout period to receive additional frames (if available), and then returns to sleep as described in Step 1.

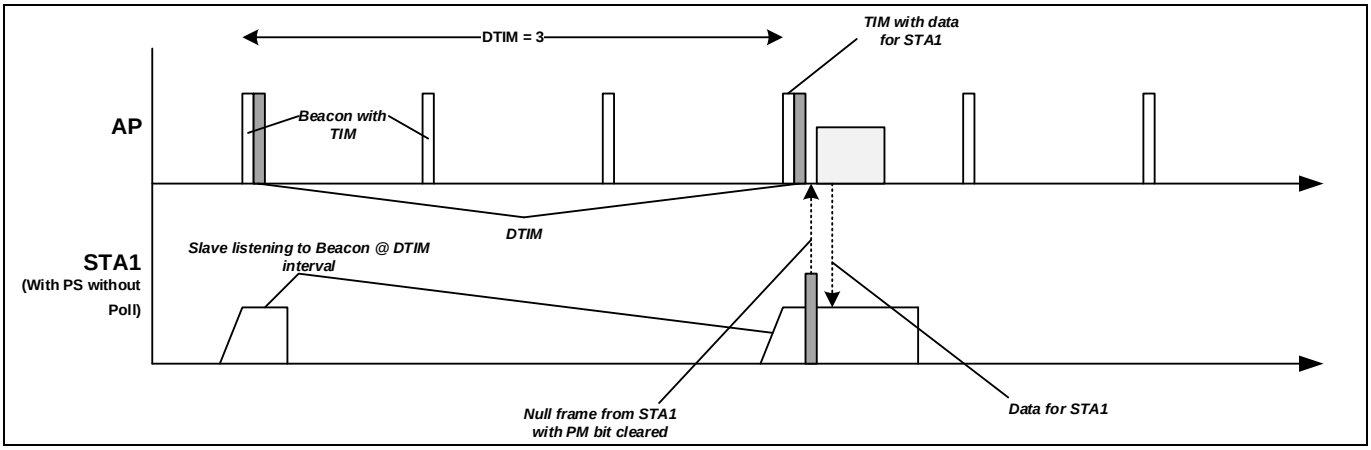

Figure 4 Power Save Without Poll

3.1.4.3 Association Timeout Limit

If a STA 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.

Because APs generally have an association timeout limit with a default value of 60 seconds, a STA that uses power save must wake up before the association timeout expires and send or receive a directly addressed frame to inform the AP that the STA is still associated. Failure to comply results in the AP de-authenticating the STA. 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.

3.1.5 802.11ac-friendly features

CYW43012 provides 802.11ac friendliness by supporting 256-QAM (for 20-MHz channels in the 5-GHz band) enabling data rates of up to 78 Mbps with 802.11ac APs. This enables superior throughput and reduced power consumption in the device by sending/receiving data and then going back to sleep faster. In addition, it supports 802.11ac's 'explicit beamforming' feature to offer significantly higher throughputs than 802.11n chipsets at any given range.

3.2 Wi-Fi power save implementation

3.2.1 WHD power save interface

The power save implementation is provided as part of the **Wireless Host Driver (WHD) library**. In addition, platforms such as Mbed OS and Amazon FreeRTOS implement their power save mechanisms on top of the WHD to provide users with a seamless low-power integration.

Table 4 describes the low-level functions available as part of WHD providing various power save features. Refer to the **WHD documentation** for details.

Table 4 WHD power save APIs

WHD API	Description
`whd_wifi_enable_powersave()`	Enables PS-Poll mode in the device (see **Power Save Poll**)
`whd_wifi_enable_powersave_with_throughput()`	Enables 802.11 PS-non-Poll mode in the device (see **802.11 Power Save Without Poll**). Use this mode when it is important to maintain throughput.

WLAN power optimization techniques

WHD API	Description
`whd_wifi_disable_powersave()`	Disables power save mode in the device; by default, power save is enabled in the WLAN device.
`whd_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 and the DTIM interval broadcasted by the AP is zero (i.e., no DTIM in the network).
`whd_wifi_get_listen_interval()`	Returns information about the number of beacons to be skipped while the Wi-Fi chip is sleeping

It should be noted that power save with throughput is enabled in the WLAN device by default. The enable/disable powersave APIs can be called to change or disable power save option in the WLAN. The WLAN automatically manages the entry and exit of DS0 after the power save is enabled. The **mbed-os-example-wlan-lowpower** code example demonstrates the use of WHD power save APIs.

An example implementation of a power save handler using WHD APIs in Mbed OS is shown below:

```
    /*
     The wifi_get_ifp() returns valid pointer in "ifp" only if the EMAC is
     powered on through "wifi->connect". So, it is recommended to call
     powersave_handler
     after wifi->connect is called.
    */

int32_t powersave_handler(WhdSTAInterface* wifi, wlan_powersave_mode_t mode)
{
    cy_rslt_t       result = CY_RSLT_SUCCESS;
    whd_interface_t ifp;

    /* Get the instance of the WLAN interface. This instance is used to obtain
       network parameters and to configure power save mode of the device.
    */
    result = wifi->wifi_get_ifp(&ifp);
    if( CY_RSLT_SUCCESS == result )
    {

            switch( mode )
            {
                case POWERSAVE_WITHOUT_THROUGHPUT:
                    result = whd_wifi_enable_powersave(ifp);
                    if( CY_RSLT_SUCCESS != result )
                    {
                        printf("Failed to enable PS-Poll mode\r\n");
                    }
                    break;
                case POWERSAVE_WITH_THROUGHPUT:
```

```
                    result = whd_wifi_enable_powersave_with_throughput(ifp,
RETURN_TO_SLEEP_MS);
                    if( CY_RSLT_SUCCESS != result )
                    {
                        printf("Failed to enable PS without poll mode\r\n");
                    }
                    break;
                case POWERSAVE_DISABLED:
                    result = whd_wifi_disable_powersave(ifp);
                    if( CY_RSLT_SUCCESS != result )
                    {
                        printf("Failed to disable power save\r\n");
                    }
                    break;
                default:
                    printf("Unknown mode\r\n" );
                    result = UNKNOWN_MODE;
                    break;
            }
    }
    else
    {
        ERR_INFO(( "WiFi interface is not initialized or connected.\r\n" ));
    }

    return result;
}
```

3.2.2 Cyclic timers in network stack

Platforms that implement network stacks, such as Lightweight TCP/IP stack (lwIP), run periodic timers for various network-related activities. These timers are used as ticks for network activity such as Address Resolution Protocol (ARP) cache table expiry and Dynamic Host Configuration Protocol (DHCP) lease renewal timeouts. The period of these timers can vary from 100 ms to 60 s. The host waking up periodically to service these timers is required for proper functioning. However, if the application desires to suspend these timers when it does not anticipate any network timeout activity and for a longer stay in low-power modes, there is an option to suspend and resume the network stack.

This feature is provided as part of the Low Power Assistant middleware helper files. The `wait_net_suspend()` API available in the *lpa/helpers/net_suspend/<platform>/network_activity_handler.h* file lets you suspend the network stack for a predefined duration or until an activity is detected, if the network remains inactive within a given time window. This API function lets you safely suspend the network and resume it whenever any network activity gets detected. **Figure 5** shows the `wait_net_suspend()` operation in different scenarios with network suspend duration of 5000 ms, network inactivity monitor window of 150 ms, and network inactivity duration of 100 ms.

WLAN power optimization techniques

Scenario 1: `wait_net_suspend()` detects no network activity for more than 100 ms within the 150-ms window and suspends the network. No network activity is detected for the network suspend duration of 5000 ms. Therefore, `wait_net_suspend()` resumes the network stack after 5000 ms.

Scenario 2: `wait_net_suspend()` detects network activity in the 150-ms window and inactivity is less than 100 ms. `wait_net_suspend()` does not suspend the network stack.

Scenario 3: `wait_net_suspend()` detects no network activity for more than 100 ms within the 150-ms window and suspends the network. Network activity is detected before the expiry of 5000 ms duration. Therefore, `wait_net_suspend()` resumes the network stack immediately.

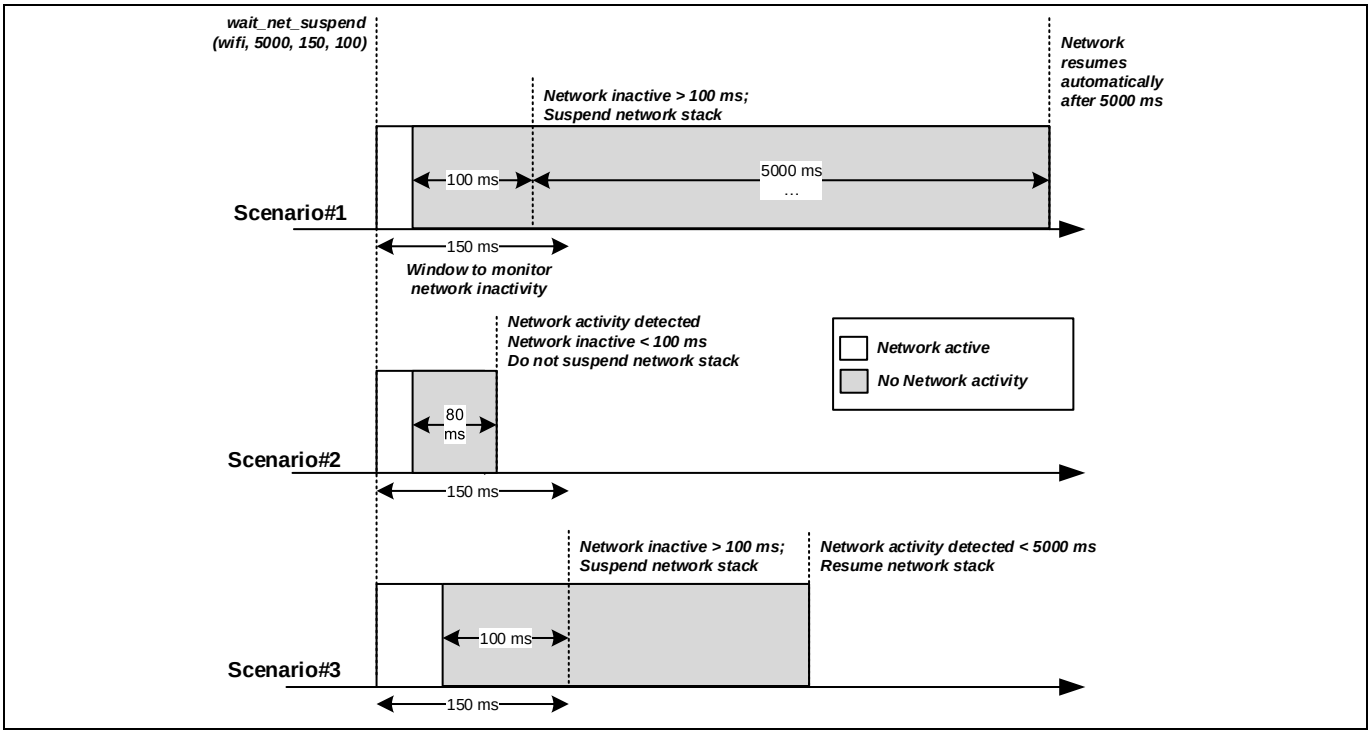

Figure 5 wait_net_suspend() operation

See the **mbed-os-example-wlan-lowpower** code example for details on using the network suspend feature safely. See **lpa middleware documentation** for additional details on these timers.

3.3 Host offloads to WLAN device

CYW43012 has an Arm® Cortex®-M3 core that performs all the WLAN device activities and supports various offloads that execute certain functionalities on behalf of the host without interrupting the host state.

Host offloads play a key role in determining host power consumption because offloads let the host go into deep sleep for extended periods of time while the WLAN device handles tasks such as 802.11 roaming, ARP, IPv6 neighbor resolution, key rotation, and TCP keepalives on behalf of the host. In addition, these offloads free the host CPU for other, more powerful tasks such as audio or sensor data processing. This in turn improves the overall system efficiency and power.

The following sections describe offloads supported by CYW43012. The offloads can be enabled using the Cypress' Low Power Assistant (LPA) middleware. Using the LPA middleware in Mbed OS is explained in **Low Power Assistant (LPA)**.

3.3.1 ARP offload

ARP is a data link layer protocol for mapping an IP address to a physical MAC address of a device. Such mapping is necessary for any WLAN device to keep updated on [IP:MAC] details of all nearby devices in its network. Using the ARP protocol, a device can detect duplicate IP addresses in a network. This is also useful in notifying peer devices of an IP address change.

Because ARP request and response packets are a common activity in any WLAN network, the host device usually gets flooded by many requests; the host needs to remain active to respond to such requests, especially in a crowded network. ARP request packets from a peer usually wake the host if it is sleeping. The ARP offloading feature in CYW43012 handles responding to ARP requests from a peer so that these requests will not disturb the host. The CYW43012 device holds a cache (of 8 entries); the host will be woken up only if there is a cache miss as shown in **Figure 6**.

In addition to auto-replying to peers, CYW43012 can use the cache to reply to the host (PSoC 6 MCU) as well when the host sends an ARP request to the network. This host auto reply feature works similar to a Peer auto reply, where a cache miss is forwarded to the network and a cache hit returns an ARP response immediately to the host. This feature is complemented by the ability of the CYW43012 device to snoop ARP info from host-network communication, i.e., whenever a host sends an ARP response packet over to the network, the [IP:MAC] information is cached into the CYW43012 ARP cache table.

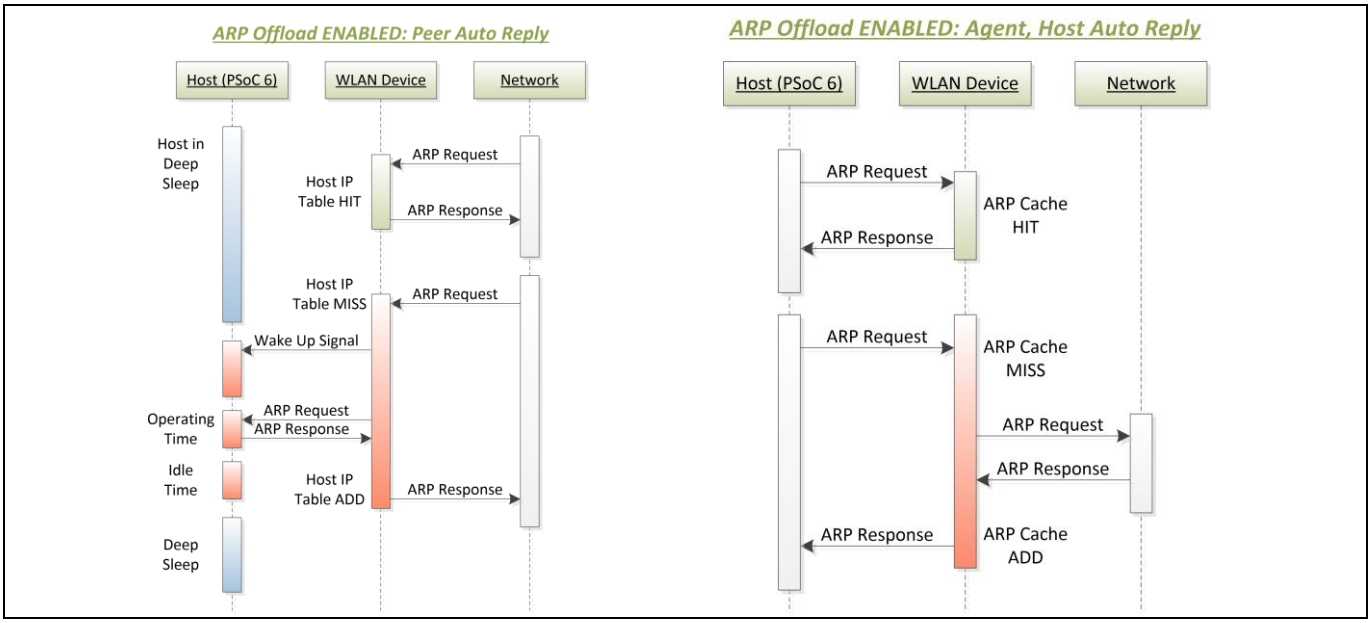

Figure 6 ARP offload feature

3.3.2 Packet filter offload

Packet filters allow the host to limit the types of packets that get passed up to the host processor from the WLAN device. This is useful to keep out unwanted packets from the network that might otherwise wake the host out of a power-saving deep sleep mode or prevent it from entering deep sleep mode.

In general, whenever a WLAN packet is destined for the host, the WLAN device must awaken the host (if it is asleep) so that the host can retrieve the packet for processing. Often, the host network stack processes the packet only to discover that the packet should be thrown away because it is not needed such as when the packet is destined for a port or service that is not being used. Packet filters allow these types of packets to be filtered and discarded by the WLAN processor itself so that the host is not interrupted.

WLAN power optimization techniques

In short, packet filters are useful when:

- You want to keep the host processor in deep sleep for as long as possible (or enter deep sleep as soon as possible) by filtering unwanted traffic.
- You want to keep the host processor involvement to a minimum by filtering and passing only the wanted traffic to it.

Filters can be configured to be active either when the host processor is active or asleep (or both). Multiple filters may be configured to operate simultaneously. One set can be functioning while awake and a separate set can function while sleeping and can be used to wake the host. In addition, filters can be configured to either discard or keep (send to host). When configured to discard, only the specified packets are discarded, while any others not specifically filtered are passed to the host. Conversely, when configured to keep packets, only the specified packets are passed to the host and the rest discarded.

A typical model would be to use only 'keep' filters, where the application is aware of all the types of packets it needs to process. In other words, use 'keep' filters to specify the complete list of packet types the host is interested in and discard the rest. This is preferred as it is much simpler to list the packets the host wants to receive versus creating a complete list of packets it doesn't want. When using keep filters, care must be taken to allow enough packets through for networking protocols to work properly.

For example, while awake, the processor must be able to join a network, get a DHCP address, run ARP and possibly share network keys. Not creating enough keep filters to allow these types of packets through will prevent the host from even joining the network. A reasonable minimal set of keep filters should include:

- ARP (allow ARP packets)
- IEEE 802.1X (allow security packets)
- DHCP (UDP port 68)
- DNS (UDP port 53)

There is no minimum set of filters required when enabling discard filters. However, it should be noted that all the filter types in the offload must be either keep or discard and not a mix of both. This is because the complementary nature of the keep and discard filters will result in unexpected behavior.

Three types of packet filters are supported in CYW43012 based on a standard network stack as seen in **Figure 7**, which shows the relationships between Stack layers, Protocols, and Packet Filters.

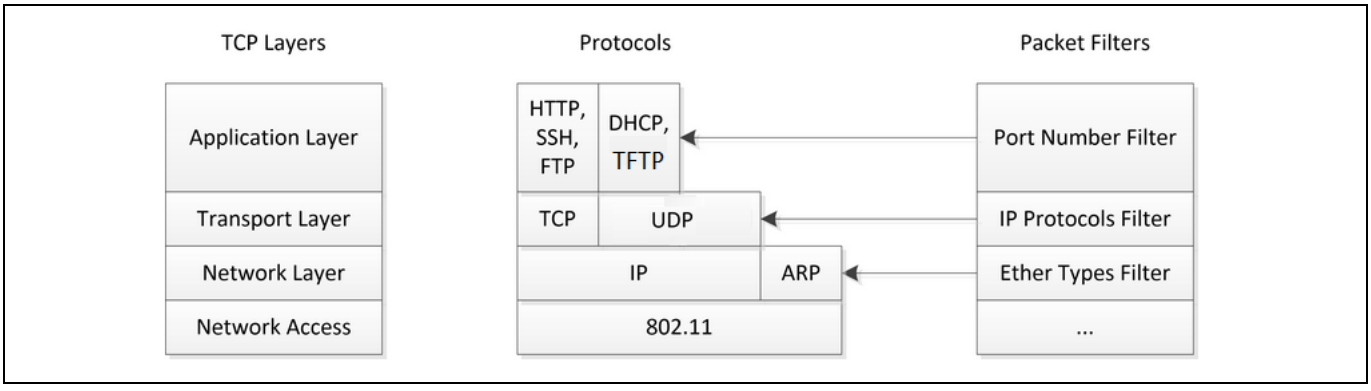

Figure 7 Packet filters

Note: *In the CYW43012 device, the number of discard packet filter offload is limited to one, i.e., only one discard packet filter can be enabled in the design; enabling more than one discard filter will let all packets to pass through (no filter).*

3.3.2.1 Network layer or EtherType filter

The EtherType filter filters packets based on a 16-bit EtherType field present in Ethernet packets seen at the network layer (see **Internet Assigned Numbers Authority, IANA)**. The most commonly used protocols (and most useful filters) are:

- IP (EtherType = 0x800)
- ARP (EtherType = 0x806)
- IEEE 802.1X or EAP over LAN (EtherType = 0x888E)

Filtering on IP EtherType would match all IP packets coming from the network. This is a very coarse filter and will include all ICMP, TCP, and UDP packets as shown in **Figure 7**. Filtering on ARP or IEEE 802.1X is finer; it will only match respective packets. Filtering all IP packets will have an enormous impact due to the substantial number of packets it will match and is generally not recommended for typical usage. Valid EtherType filters consist of a 16-bit number greater or equal to 0x800.

3.3.2.2 Transport layer/IP protocols

The next layer up the stack is the Transport layer, which consists of various IP-based protocols such as TCP, UDP, and ICMP. Discussions of the protocols themselves are outside the scope of the application note but are widely available and list of protocol numbers can be found in numerous sources such as **IANA**.

IP Protocol filters consist of a single 8-bit number. The filters do not perform any validation check on this 8-bit number and simply filter the packets matching the number. This is because it is possible for vendors to use proprietary numbers and protocols at this layer. It should be noted that filtering on TCP/UDP protocols is still coarse and will likely include most packets destined for the host processor (depending on the application). Application layer filters that are based on port numbers are the next level of filter refinement providing finer control on the majority of packets received in the network.

3.3.2.3 Applications layer/TCP and UDP port numbers

Application layer filters or simply port filters perform packet filtering based on the source or destination port of a transport layer packet. These port numbers are well-known numbers used to identify various TCP- and UDP-based protocols. (see **IANA**). These are 16-bit port numbers. For example, with a port filter, a user can filter on only SSH packets (port 22) or only on FTP packets (port 20), or any other of the many applications supported. Due to the large number and constantly changing port definitions, the filters do not validate the port numbers against available applications.

Transport layer packets have both source and destination ports. Destination port are the well-known port numbers described in the IANA link and generally the most useful. Source ports describe temporary, ephemeral port numbers used by the host sending the packets and are generated on-the-fly and are not well known. Because source ports are not known ahead of time, creating a source port filter is difficult. Filters can be designed to filter a range of ports (between a start and end port) to match a wide range of source ports to cover this case. Both TCP and UDP use port numbers; therefore, filters can be designed to select TCP or UDP. If both TCP and UDP need to be filtered for a port, two filters can be created.

4 Bluetooth power optimization techniques

CYW43012 supports a Bluetooth 5.0 BR/EDR/Bluetooth LE radio. This section describes the power optimization techniques in designs involving Mbed OS Cordio stack that provides Bluetooth LE support to CYW43012 in Mbed OS. For details on Cordio stack and related documentation, see the Mbed OS **documentation page**.

Common operations related to power consumption in a Bluetooth LE application include advertisement and connection events. We will discuss how to optimize various parameters related to both advertisement and connection events to reduce power consumption. This section assumes that you are familiar with Bluetooth LE fundamentals. See Vol 6: “Core System package [Low Energy Controller]” of the latest **Bluetooth Core specification** for Bluetooth LE fundamentals.

4.1 Advertisement events

Advertisement events broadcast the capabilities of a Bluetooth LE Peripheral device to Bluetooth LE Central devices listening to the events. These events can constitute the major portion of a Bluetooth device’s battery consumption, if not carefully chosen. The interval between two successive advertisement events from the Peripheral forms the advertising interval; it can be fixed or a random value between a min and max interval.

To achieve lowest possible power during advertisement events, you should maximize the advertising interval. However, a larger advertising interval will lead to a longer detection time at the Central device and a subsequently longer time for getting connected to the Central. The advertising interval should be between 20 ms and 10.24 seconds. In addition, having the advertisement interval between a min and max not only improves the tradeoff between power and latency, but also improves the detection probability in noisy environments by randomizing the time at which an advertisement packet is sent out.

Table 5 Advertising Interval selection

Application requirement	Suggested Advertisement Interval
The application requires shorter detection and connection establishment periods such as audio devices or devices that send real-time data.	20 to 100 ms
The application does not have aggressive connection requirements but still prefers low latency such as fitness trackers or smart watches.	100 to 1000 ms
The application does not care about latency; power consumption is of more importance, such as in the case of beacons.	1000 to 10000 ms

In addition to advertisement interval, the amount of data transmitted during advertisement impacts the power consumption during advertisement events. The smaller the advertisement payload, the lower is the power consumption. In Mbed OS, the `gap().setAdvertisingParameters()` API function can be used to set the advertisement interval. See the **Mbed OS Bluetooth LE API documentation** for details and **mbed-os-example-ble** for examples that show advertisement parameter configuration.

4.2 Connection events

4.2.1 Connection Interval

The connection interval for a Bluetooth LE link is set by the Central device when a connection is created with the Peripheral device. It is preferable that the Peripheral has a connection interval that matches the rate at which the sensor data from the Peripheral is sent to the Central device. If the initial connection interval set by the Central is lower than necessary, the Peripheral should request the Central to increase the connection

interval to reduce current consumption. This can be done after a connection is established by sending a "connection parameter update request" to the Central.

4.2.2 Use slave latency

Slave latency is a Bluetooth LE feature that allows a Peripheral to listen to the Central device on connection events at a reduced rate. This is useful if the Peripheral device is inactive for some time and has to wait for some activity to occur to send data to the Central device. If no activity is detected for some time, the Peripheral can enter slave latency by sending a connection update request to the Central. The Peripheral can extend the interval between connection events at which it listens to the Central. This allows the Bluetooth LE device and the host to be put into deep sleep mode for longer durations.

The key advantage of using slave latency is that when the Peripheral application detects an activity, it can switch the Bluetooth LE device back ON to listen to the Central at the original connection interval until the data transfer is completed. This avoids the latency in sending the data upon the first activity.

4.2.3 Connection parameter update in Mbed OS

In Mbed OS, you can update the connection parameters by calling the `updateConnectionParams()` function as shown in the code snippet below. You can also set the preferred connection parameters before the connection by calling `setPreferredConnectionParams()`. However, it is always a good practice to check the actual connection parameters post connection to see whether the Central has honored the requested connection parameters. In any case, it is finally up to the Central device to accept or reject the preferred or updated connection parameters.

Here is the snippet to update connection parameters after connection:

```
void connection_callback(const Gap::ConnectionCallbackParams_t *params)
{
	Gap::Handle_t gap_handle = params->handle;
	Gap::ConnectionParams_t new_params;

	new_params.minConnectionInterval = BLE_GAP_CP_MIN_CONN_INTVL_MAX;
	new_params.maxConnectionInterval = BLE_GAP_CP_MAX_CONN_INTVL_MAX;
	new_params.slaveLatency = BLE_GAP_CP_SLAVE_LATENCY_MAX;
	new_params.connectionSupervisionTimeout =
	BLE_GAP_CP_CONN_SUP_TIMEOUT_MAX;

	ble.gap().updateConnectionParams(gap_handle, &new_params);
}

int main(void)
{
	ble.init();
	ble.gap().onConnection(connection_callback);
	……
}
```

Bluetooth power optimization techniques

Here is the snippet for updating preferred connection parameters:

```
Gap::ConnectionParams_t new_params;

ble.gap().getPreferredConnectionParams(&new_params);

new_params.minConnectionInterval = BLE_GAP_CP_MIN_CONN_INTVL_MAX;
new_params.maxConnectionInterval = BLE_GAP_CP_MAX_CONN_INTVL_MAX;
new_params.slaveLatency = BLE_GAP_CP_SLAVE_LATENCY_MAX;
new_params.connectionSupervisionTimeout = BLE_GAP_CP_CONN_SUP_TIMEOUT_MAX;

ble.gap().setPreferredConnectionParams(&new_params);
```

See **Mbed OS Bluetooth LE APIs** for more details.

5 PSoC 6 MCU power optimization techniques

5.1 Core voltage and operating frequency

PSoC 6 MCU supports two core regulators – LDO or Buck – either of which can be used to power the CPU core. In addition, the system supports two active power modes – System LP and System ULP. In the System LP power mode, the CPU frequency can reach up to 150 MHz, whereas in the System ULP mode, the CPU frequency is limited to 50 MHz. The power consumed by the CPU is much lower in System ULP mode.

By default, the LDO powers the core and it is in System LP mode. The LDO regulator consumes a higher power than the buck regulator in active mode (>50%) for the same CPU clock frequency irrespective of the LP or ULP mode. However, in LP mode, the leakage in deep sleep is lower (~20%) using the LDO regulator as compared to the buck regulator. Unless the application spends < 0.01% time in active mode (1 ms out of 10 s), it is recommended to use the buck regulator as the core regulator because the added leakage in deep sleep using the buck regulator is outweighed by the power saving in active mode. When using ULP mode for active power mode, it is always recommended to use the buck regulator because the leakage in ULP mode is lower than the LDO in System ULP mode.

Table 6 Core regulator and active power mode selection

Application use case	Core regulator	System active power mode
(CPU Clock > 50 MHz OR Peri Clock > 25 MHz) AND CPU active/sleep time < 0.01 %. 0.01 % = 1 ms out of 10 s.	LDO	LP
(CPU Clock > 50 MHz OR Peri Clock > 25 MHz) AND CPU active/sleep time > 0.01 %. 0.01 % = 1 ms out of 10 s.	Buck	LP
(CPU Clock < 50 MHz AND Peri Clock < 25 MHz)	Buck	ULP

See **AN219528** for details on the power modes and **PSoC 6 MCU datasheet** for details on the power parameters.

5.2 Reducing the leakage in deep sleep

PSoC 6 MCU supports selective retention of SRAM in sizes of 32 KB when the device enters deep sleep. The smaller the amount of SRAM retained in deep sleep, the lower the deep sleep power consumption will be. If the application knows the amount of SRAM it plans to use, the unused SRAM blocks can be disabled to improve the deep sleep power consumption. However, some guidelines must be kept in mind while disabling the SRAM blocks:

1. Some platforms, such as Mbed OS, allocate the unused SRAM to heap. As a result, the amount of SRAM consumed by the heap (or dynamically allocated objects in the code) should be considered and added to the budget.
2. System calls supported in PSoC 6 MCU use 2 KB of SRAM available at the end of the SRAM region present in the device. As a result, if the application uses/requires any system calls, such as flash write/erase, it must not disable the last block of the SRAM (block 8 in PSoC 6 MCU devices with 288 KB RAM). However, the block can still be turned off before entering deep sleep (because system calls are executed in active mode only).
3. All projects should reserve 8 KB (0x2000) of SRAM at the start of the SRAM region for the CM0+ core; this should be accounted into the overall SRAM consumption of the application.

Do the following to disable an unused SRAM block in the code:

1. Find out the SRAM consumed by the application. CPU Stack statistics and other memory analyzer tools supported in the platforms can be used to determine the memory usage.
2. Update the linker script to reduce the RAM size to the value calculated above. Make it a multiple of 32 KB and fitting the size calculated in Step 1. This step is compiler-dependent as well.

   GCC (cy8c6xx*_cm4_dual.ld file):

   ```
   ram                (rwx)    : ORIGIN = 0x08002000, LENGTH = 0x76000
   ```

   ARM (cy8c6xx*_cm4_dual.sct file):

   ```
   #define RAM_SIZE                    0x00076000
   ```

   IAR (cy8c6xx*_cm4_dual.icf file):

   ```
   define symbol __ICFEDIT_region_IRAM1_start__ = 0x08002000;
   define symbol __ICFEDIT_region_IRAM1_end__   = 0x08078000;
   ```

   CLANG (cy8c6xx*_dual.mk file):

   ```
   export RAM_SIZE_CM4       := 0x00076000
   ```

3. Add code to disable SRAM blocks and/or controllers depending on the device and memory consumed. For example, consider the following cases:

a) PSoC 6 MCU with 1 MB of SRAM has three controllers – SRAM0 (512 KB), SRAM1 (256 KB) and SRAM2 (256 KB). As an example, if your application requires only 450 KB of SRAM (= 480 KB, multiple of 32 KB), you can completely disable SRAM1 and SRAM2 controllers and the last block of SRAM0 using the following code:

```
CPUSS->RAM0_PWR_MACRO_CTL[15]= 0x05FA0000; //Disable block 15 in SRAM0
CPUSS->RAM1_PWR_CTL= 0x05FA0000;//Disable SRAM1
CPUSS->RAM2_PWR_CTL= 0x05FA0000;//Disable SRAM2
```

b) If you want to add system call support in the same application, you should not disable the last block of the SRAM region, which is the last 32 KB block in SRAM2:

```
CPUSS->RAM0_PWR_MACRO_CTL[15]= 0x05FA0000;
CPUSS->RAM1_PWR_CTL= 0x05FA0000;
for(int32 i = 0; i < 15; i++) //Do not disable block 15 alone.
CPUSS-> RAM0_PWR_MACRO_CTL[i]= 0x05FA0000;
```

5.3 RTOS tickless mode

All RTOS require a clock or a "tick" for timing and managing tasks. These ticks are usually small RTOS kernel tasks that wake up periodically and update the RTOS tick count used by delays, timeouts, and other timing-related tasks. Often, these ticks are configured in ms. This results in the CPU waking up often just to service the tick task even if it is not running any useful tasks.

Applications that do not use delays or timeouts or use large delays need not wake up this often and can enter a tickless mode provided by the platform. In tickless mode, the RTOS suspends the tick task and lets the CPU enter a low-power state (deep sleep) for longer durations. This duration is typically determined by the time the next task that depends on a delay needs to be triggered. If there are no tasks dependent on delays, the device can remain in the low-power state indefinitely until an external event (such as a GPIO interrupt) wakes up the device and triggers the task.

When delays are involved while entering tickless mode, a timer that is active in the System deep sleep state is configured to generate an interrupt after the delay expires.

PSoC 6 MCU power optimization techniques

All RTOS platforms provide a tickless mode of operation. For example, in Mbed OS, you can define the `MBED_TICKLESS` macro to enable tickless operation. See the Mbed OS **Tickless mode documentation** for details. In addition, you can control whether the OS enters sleep or deep sleep when idle. This is useful when the OS (CPU) can be idle but peripherals such as an ADC or UART still operate and require the system to not enter deep sleep. You can "lock" deep sleep entry in such cases allowing the OS to enter sleep instead and 'unlock' the deep sleep entry after the peripheral completes its operation. See Mbed OS **Power management section** on how to control sleep or deep sleep entry.

5.4 Additional power optimization techniques

See **AN219528** for additional power optimization techniques in PSoC 6 MCU.

6 Low Power Assistant (LPA)

The Infineon Low Power Assistant (LPA) allows configuring PSoC 6 MCU Host and WLAN (Wi-Fi/Bluetooth Radio) devices to provide low-power features. Key highlights of the LPA include the following:

- Self-aware firmware that detects configurations automatically and enables appropriate low-power features without any additional API calls from the user
- Supports multiple platforms such as Mbed OS and FreeRTOS
- GUI-based configuration for ease of use
- Supports low-power configuration for PSoC 6 MCU, Wi-Fi, and Bluetooth

LPA lets you optimize various parts of your design to be energy-efficient. The LPA configuration is split into three main parts:

- **PSoC 6 MCU (Host) Low-Power Configuration:** Includes PSoC 6 MCU-specific low-power configurations such as core voltage, regulator selection, and RTOS idle power mode
- **Wi-Fi Low-Power Configuration:** Includes Wi-Fi-specific low-power configurations such as Wi-Fi host wake signal selection and host offload configurations (ARP and packet filters)
- **Bluetooth Low-Power Configuration:** Includes Bluetooth low-power configurations such as host wake interrupt signal (device to host) and related device integration

These configurations can be generated through ModusToolbox's Device Configurator or added manually through code. Given the simplicity of the configurations, adding them through code is not difficult. However, using the configurator lets you take advantage of a GUI-based configurator in addition to making the generated configuration easily upgradable in the future; it is the recommended way to create low-power configurations.

See **LPA documentation** for details on the middleware, configurators, and quick start. The following sections provide a brief on the PSoC 6 MCU, Wi-Fi, and Bluetooth low-power configurations available and their usage in different platforms.

6.1 LPA configuration

To use the LPA in your application, you need to input some configurations. It is recommended to use the *design.modus* device configuration file shipped with the kit BSP for configuring LPA settings. The LPA settings are available under **design.modus** > **[PSoC 6 device in target]** > **System** > **Power** for PSoC 6 MCU-related power settings and **design.modus** > **[CYW43012 device in target]** > **Power** > **WiFi or BT** for Wi-Fi and Bluetooth settings. **Figure 8** shows the PSoC 6 MCU and Wi-Fi LPA (power) settings for the CY8CKIT-062S2-43012 target. You need to enable the "Power" setting to enable LPA configuration generation. For a detailed description of all the configurations, see the **LPA documentation**.

Low Power Assistant (LPA)

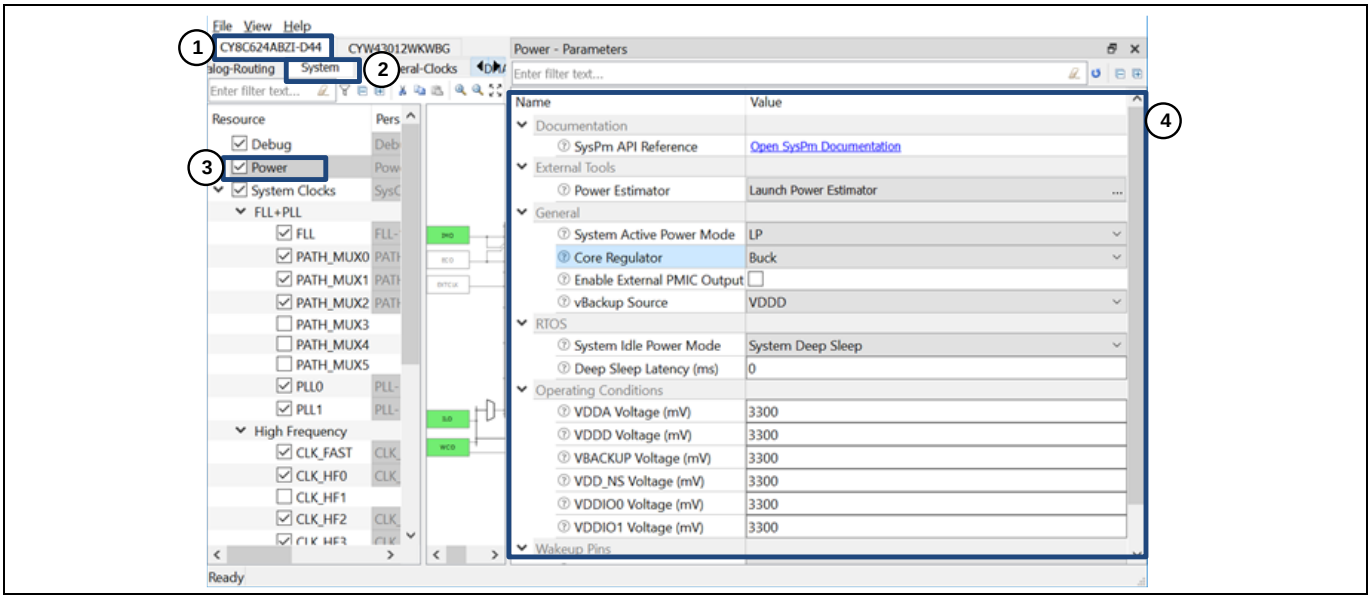

Figure 8 PSoC 6 MCU LPA configurations in *design.modus*

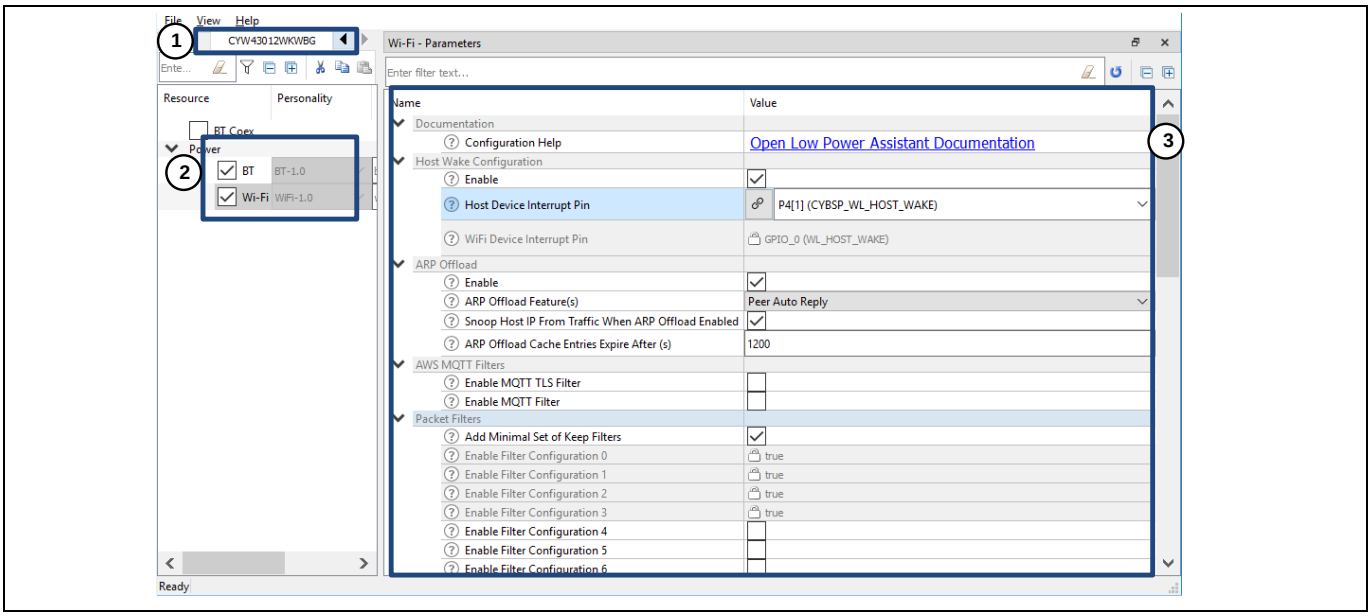

Figure 9 Wi-Fi/Bluetooth LPA configuration in *design.modus*

Table 7 Guidelines for LPA configuration

Parameter	Description	Recommended configuration
PSoC 6 MCU (Host)		
System active power mode	Selects the power mode the core operates in when active – LP or ULP	See **Table 6**.
Core regulator	Selects the type of regulator that powers the core	
System idle power mode	Selects the power mode the core operates in when idle – active, sleep, or deep sleep	Deep sleep (for maximum power saving) Active (for minimum latency)

Low Power Assistant (LPA)

Parameter	Description	Recommended configuration
Deep sleep latency (ms)	Selects the minimum time for which CPU must be idle before entering deep sleep	0 (enters deep sleep immediately in idle) because the deep sleep wakeup latency is <50 us, it is OK to enter deep sleep as soon as the device hits idle. Increase this parameter only if the application involves tasks that need to wake up/run every 2-3 ms at times.
Wi-Fi		
Host wake configuration	Enables the capability of the WLAN device to wake up the host from deep sleep	Enable (Check-box ticked)
Host device interrupt pin	Selects the pin from host to be configured as the host wake signal from the WLAN device	First, configure the pin in PSoC 6 MCU that is connected to WL_HOST_WAKE (see board schematics). The pin configuration includes setting the drive mode to **Digital Hi-Z** and interrupt trigger type to **Falling Edge**. The interrupt trigger type selects the polarity of the WL_HOST_WAKE signal from CYW43012. Falling selects “Active LOW” and Rising selects “Active HIGH” configuration. For example, in CY8CKIT-062S2-43012, P4[1] is connected to WL_HOST_WAKE of the CYW43012 device. So, enable the pin, set the drive mode to **Digital Hi-Z,** interrupt trigger type to **Falling Edge** and select it from the drop-down.
ARP offload	Enables ARP offload feature	Enabled
ARP offload features	Selects the functionality of ARP offload	Peer Auto Reply is sufficient for most use cases.
Snoop host IP from traffic	Selects whether the CYW43012 device snoops ARP information from host ARP responses	Enabled
Packet filters	Configure multiple packet filters	Depends on the application. See **Packet filter offload**. If using “keep” filters, enable the **Add Minimal Set of Keep Filters** option.
Bluetooth		
Enable	Enables low-power Bluetooth operation	Enabled

Low Power Assistant (LPA)

Parameter	Description	Recommended configuration
Host-wake-up configuration	Selects the pin from host to be configured as the wakeup signal from the Bluetooth device.	Board-dependent and similar to "Host Device Interrupt Pin" configuration in Wi-Fi.
Device-wake-up configuration	Selects the pin from host to be configured as the wakeup signal to the Bluetooth device	

6.2 Using LPA in Mbed OS

Do the following to include the Low Power Assistant in any Mbed OS example (supported for Infineon targets):

1. Ensure that you are using Mbed OS 5.14.2 or later.
2. Import the *lpa* and *connectivity-utilities* libraries into the Mbed OS example as explained below; see **Table 9** for LPA examples.
3. Create *lpa.lib* inside the example folder with the following path:

   `https://github.com/cypresssemiconductorco/lpa.git`
4. Create *connectivity-utilities.lib* with the following path:

   `https://github.com/cypresssemiconductorco/connectivity-utilities.git`
5. Run `mbed deploy` to download all the libraries.

Note: Alternatively, you can do `git clone` the libraries inside the example folder.

Do the following to open *design.modus* for the Infineon target used in the example:

1. Install ModusToolbox 2.0 from **here**.
2. Open *[*Install *Directory]\ModusToolbox\tools_2.0\device-configurator\device-configurator.exe.*
3. In the Device Configurator, select **File** > **Open**.
4. Navigate to the *design.modus* file for your target available inside the following Mbed OS folder:

   [Example_Directory]\mbed-os\targets\TARGET_Cypress\TARGET_PSOC6\TARGET_[CY_BOARD]\COMPONENT_BSP_DESIGN_MODUS\design.modus
5. If an error message as shown in **Figure 10** appears, click **OK** and then navigate to *[Example_Directory]\mbed-os\targets\TARGET_Cypress\TARGET_PSOC6\psoc6pdl* and select **devicesupport.xml** file.

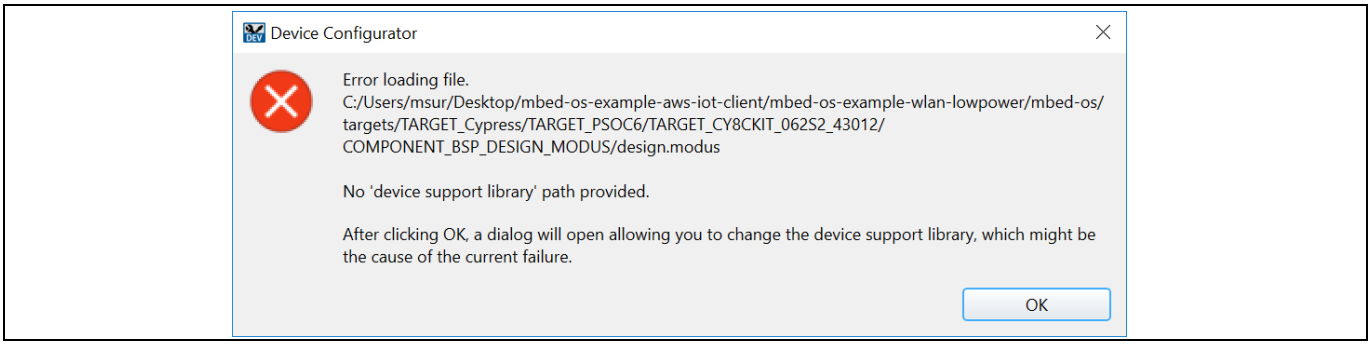

Figure 10 No 'device support library' path found error

6. Back up the existing *design.modus* file if required. Open *design.modus* and configure the LPA as explained in **LPA configuration**. For example, to enable low power in the AWS Subscriber or Publisher example available in **Cypress GitHub**, you can use the configuration provided in **Table 8**.

Low Power Assistant (LPA)

Table 8 LPA Configuration for AWS Subscriber example

Parameter	Configuration	Comment
PSoC 6 MCU (Host)		
System active power mode	LP	You need >50 MHz M0+ for Wi-Fi applications
Core regulator	Buck	CPU will wake up often (>0.1%) to service Wi-Fi activities
System Idle Power mode	Deep sleep	For lowest power consumption during idle
Deep sleep latency (ms)	0	Enters deep sleep immediately in idle
Wi-Fi		
Host wake configuration	Enable	Required for low power
Host device interrupt pin	<P4.1 for CY8CKIT_062S2_43012>	-
ARP offload	Enabled	Reduces the number of ARP requests to host and improves power consumption
ARP offload features	Peer Auto Reply	
Snoop host IP from traffic	Enabled	
Packet filters	Enable "**Add Minimal Set of Keep Filters**" option	Ensures that the device can connect to a Wi-Fi AP and obtain an IP address
	Enable "**MQTT TLS filter**" Enable "**MQTT filter**" Other configuration - *Action*: **Keep** *Protocol*: **TCP** *Direction*: **Source Port**	Ensures that all MQTT packets reaches the host. After connection to the AP, the host receives only the MQTT packets.

7. Disable peripherals such as CapSense, Serial Communications Block, and Quad Serial Memory Interface in PSoC 6 MCU if they are enabled but not used in the example, as shown in **Figure 11**.

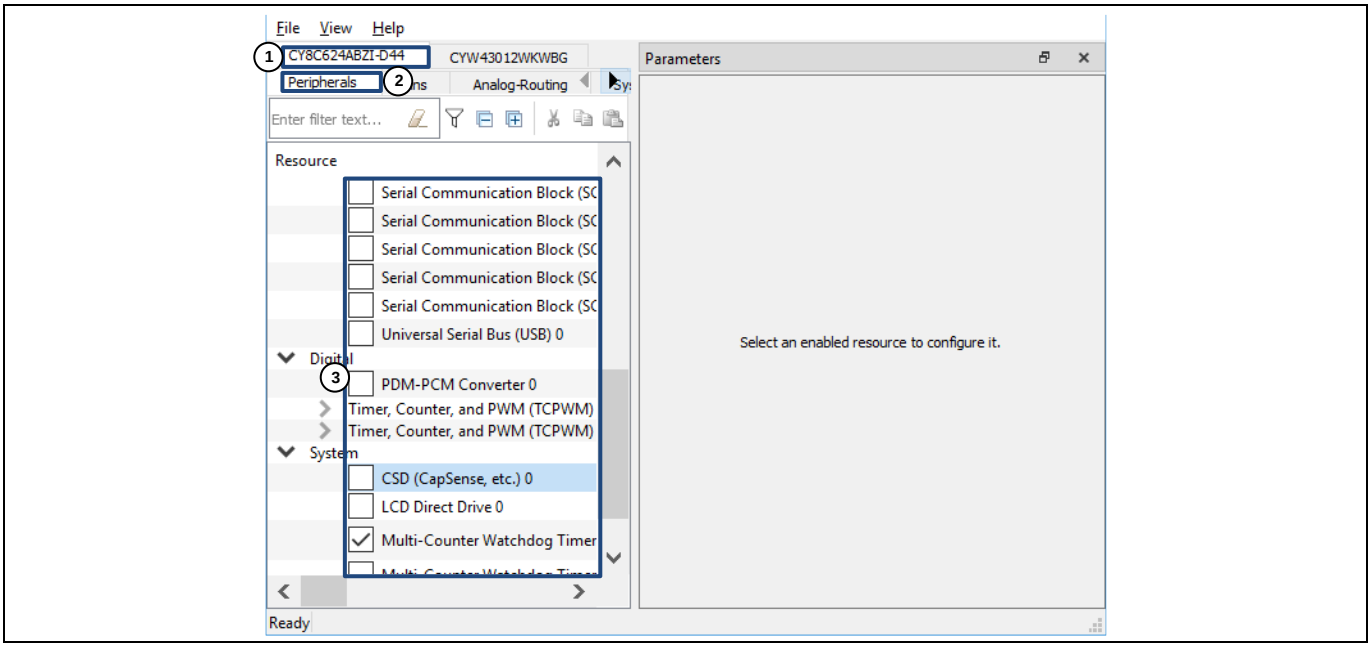

Figure 11 Disable peripherals in *design.modus*

8. Review the system clock settings and disable unused clock paths and PLLs. For example, in CY8CKIT-062S2-43012 *design.modus*, PLL1 can be disabled if USB is not used. To save further power, PLL0 can be disabled and FLL can be routed to CLK_HF0 (CPU clocks). See **Figure 12**.

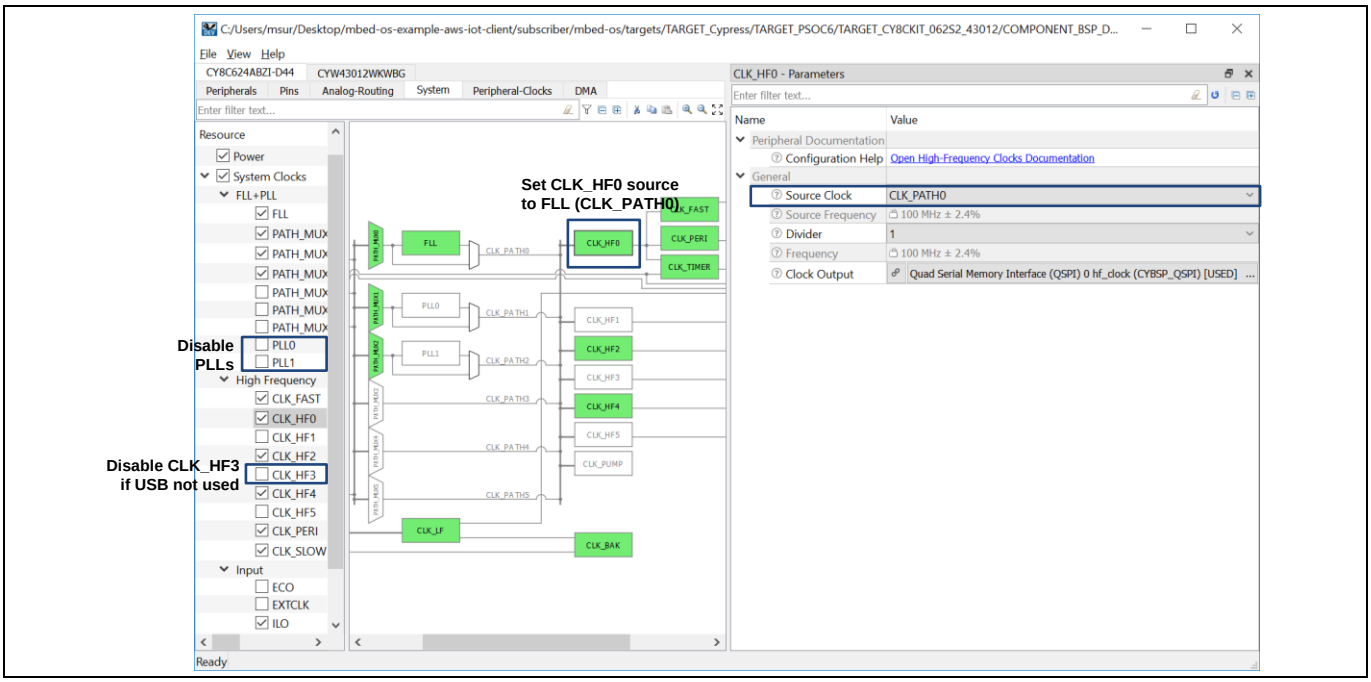

Figure 12 CY8CKIT-062S2-43012 clock setup for low power

9. Select **File** > **Save** to save the configuration.
10. Compile your Mbed OS example as follows:

```
mbed compile -DMBED_TICKLESS --target [CY_BOARD] --toolchain [TOOLCHAIN]
```

11. Program the target.

Low Power Assistant (LPA)

Table 9 Mbed OS examples for LPA

Example	Description	LPA features
mbed-os-example-wlan-lowpower	Demonstrates WLAN low-power modes and host network suspend features	Wi-Fi wake host signal
mbed-os-example-wlan-offload-arp	Demonstrates the WLAN ARP offload feature	Wi-Fi wake host signal, ARP offload
mbed-os-example-wlan-offload-packet-filter	Demonstrates WLAN packet filter offload with an option to evaluate filters in run-time	Wi-Fi wake host signal, Packet filters
mbed-os-example-wlan-offload-discard-packet	Demonstrates a simple WLAN discard packet filter offload that discards ping packets	Wi-Fi wake host signal, Packet filters

7 Power measurement using CY8CKIT-062S2-43012

7.1 Hardware setup

For power measurement, you should use a power analyzer such as KeySight **N6705B** because it gives insights on power transitions and a better estimate of power saving achieved using offloads. If you do not have access to a power analyzer, a simple 6 ½-digit multimeter can be used to monitor the PSoC 6 MCU and WLAN average power individually. **Figure 13** shows various power measurement jumpers available on the CY8CKIT-062S2-43012 kit.

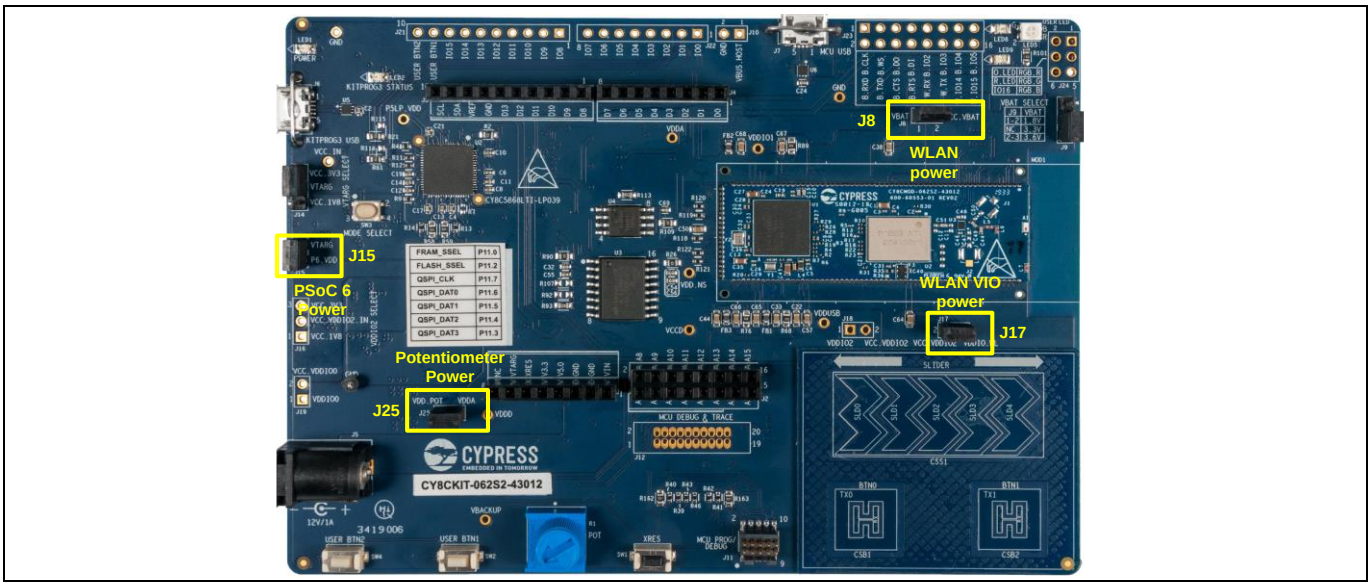

Figure 13 Power measurement jumpers in CY8CKIT-062S2-43012

Do the following for power measurement in CY8CKIT-062S2-43012:

1. Measure current for PSoC 6 at J15 across VTARG and P6.VDD.
2. Ensure that J25 is removed. This eliminates the leakage current from VDDA through potentiometer R1.
3. Measure the current for CYW43012 VBAT at J8 across VBAT and VCC.VBAT.
4. Measure the current for CYW43012 VDDIO at J17 across VDDIO.WL and VCC.VDDIO2.
5. See **Figure 14** for a sample setup using N6705B. Make sure that you select the PSoC 6 MCU (VTARG) and CYW43012 (VBAT) voltage as the same as shown in the setup. The setup does not measure CYW43012 VDDIO because it usually varies and depends on the activity on SDIO lines and the impact is minimal during active power measurement.

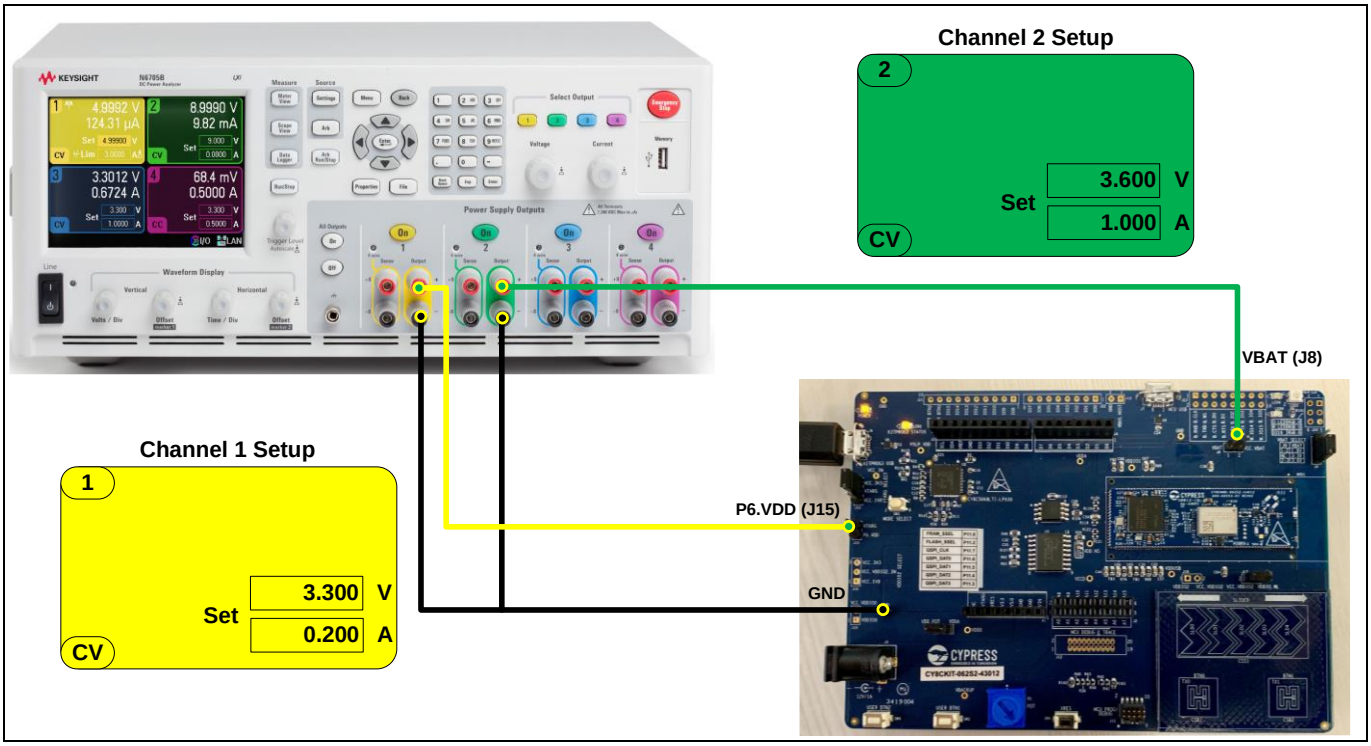

Figure 14 Power measurement setup using Keysight N6705B

7.2 mbed-os-example-wlan-lowpower

This section provides details on power measurement in CY8CKIT-062S2-43012 with the **mbed-os-example-wlan-lowpower** code example. This example can be used to measure CYW43012 power consumption as documented in the datasheet.

By default, the example configures CYW43012 in Power Save Without Poll (PS-non-Poll) mode for better throughput, enables `MBED_TICKLESS`, hosts an HTTP server, and suspends the network stack so that PSoC 6 MCU remains in deep sleep for as long as no activity is detected in the network. Follow the example webpage for details on configuring your access point name/password and for programming the code example into the kit.

1. Set up the CY8CKIT-062S2-43012 kit as explained in **Hardware setup**. Note that this section uses a Keysight N6705B as the reference setup to measure power.
2. Turn on the channels to power PSoC 6 MCU and CYW43012 from N6705B. You can connect the on-board KitProg for viewing logs in a terminal software through the COM port.
3. Make sure that the configured Wi-Fi AP is in range and the device connects to it (**Figure 15**).

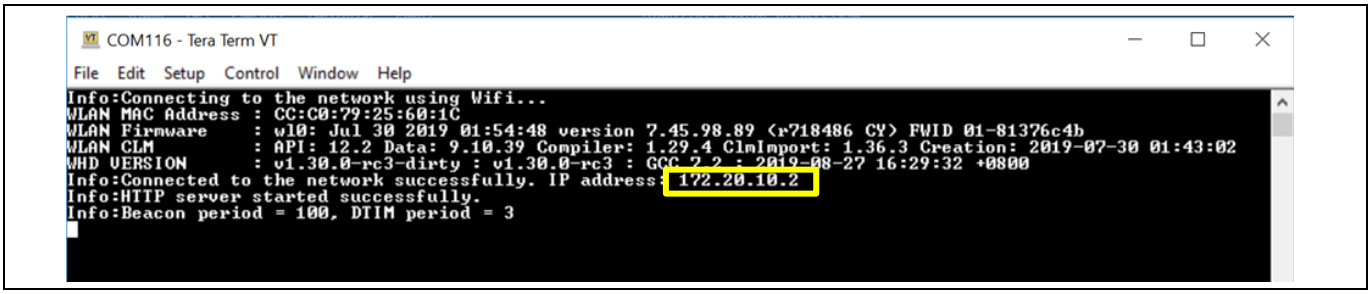

Figure 15 Terminal output after connection with AP

4. Monitor the power consumption at P6.VDD and VBAT. **Figure 16** shows the power consumption of both PSoC 6 MCU (P6.VDD) and CYW43012 (VBAT) after board initialization and association with the AP. The AP is configured for a beacon interval of 102.4 ms and DTIM = 1.

   It should be noted that the average power consumption will vary between DTIM intervals. This is because of other activities in the Wi-Fi spectrum (2.4 GHz or 5 GHz) such as other Wi-Fi networks, Bluetooth (2.4 GHz) or Long-Term Evolution (LTE in 2.4 GHz) that increase the duration for which CYW43012 listens for DTIM packet from the AP. This type of activity will result in an increase in the average power consumption. To obtain the datasheet numbers, measure power in a shielded environment to avoid any interference.

   Figure 16 shows a measurement taken in an open environment and therefore shows slightly higher consumption than the values provided in the datasheet (shielded environment).

Figure 16 CY8CKIT-062S-43012 power consumption (Associated, DTIM=1, Beacon Interval = 102.4 ms)

5. From a device (PC or mobile) connected to the same AP, enter `http://<ip_address>/` in a web browser, where `<ip_address>` is the one listed in the UART terminal window. The web page hosted in the device loads (**Figure 17**). **Figure 18** shows the power consumption when the user interacts with the HTTP server (such as loading the page or clicking the **Toggle LED** button) hosted by the board.

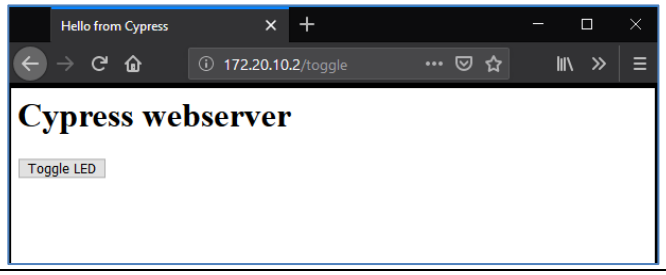

Figure 17 HTTP web page

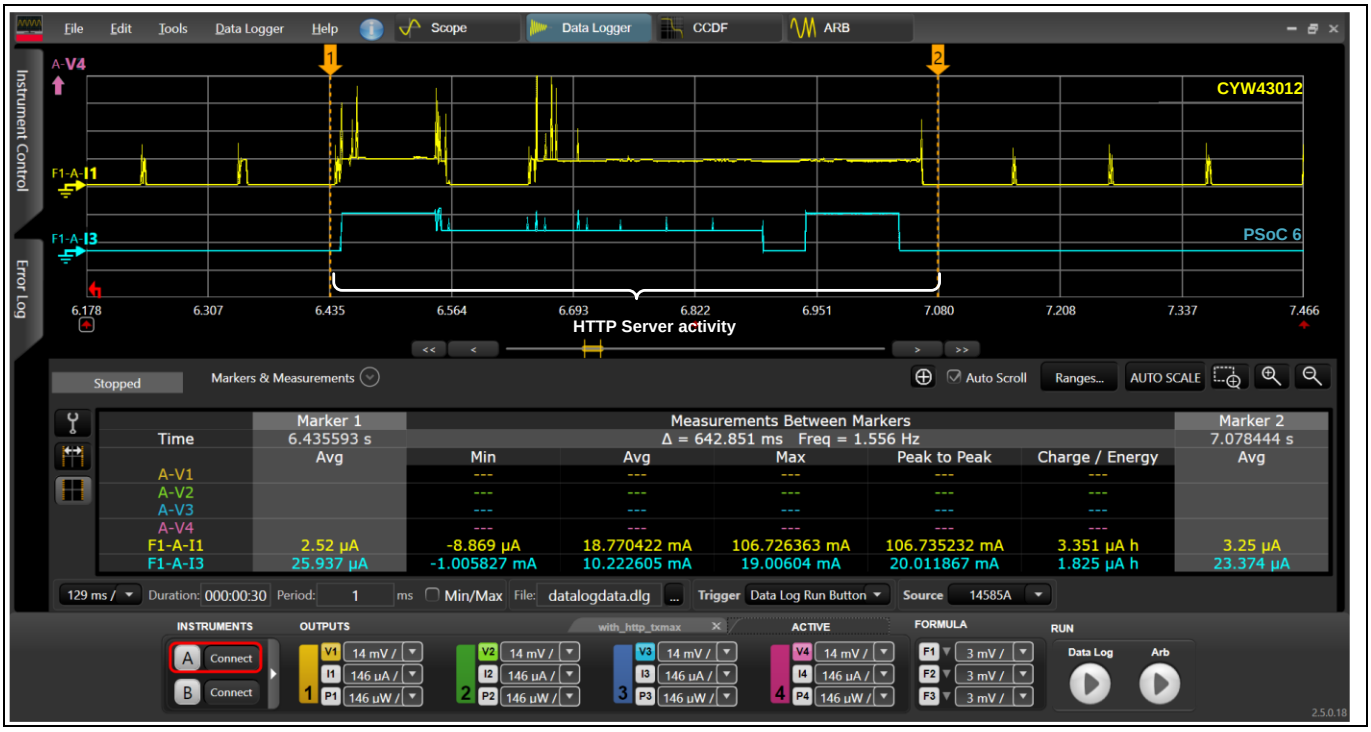

Figure 18 CY8CKIT-062S-43012 power consumption during HTTP server activity

7.3 mbed-os-example-wlan-offload-arp

This section provides details on power measurement in CY8CKIT-062S2-43012 with the **mbed-os-example-wlan-offload-arp** code example. This example uses the ARP offload feature.

By default, the example hosts a webpage and enables ARP offload to control the PSoC 6 MCU (host) deep sleep state.

1. Set up the CY8CKIT-062S2-43012 kit as explained in **Hardware setup**. Note that this section uses a Keysight N6705B as a reference setup to measure power.
2. Turn on the channels to power PSoC 6 MCU and CYW43012 from N6705B. You can connect the on-board KitProg for viewing logs in a terminal software through the COM port.
3. Make sure that the configured Wi-Fi AP is in range and the device connects to it.
4. From a device (PC or mobile) connected to the same AP, enter `http://<ip_address>/` in a web browser, where `<ip_address>` is the one listed in the UART terminal window. The web page hosted in the device loads (**Figure 19**).

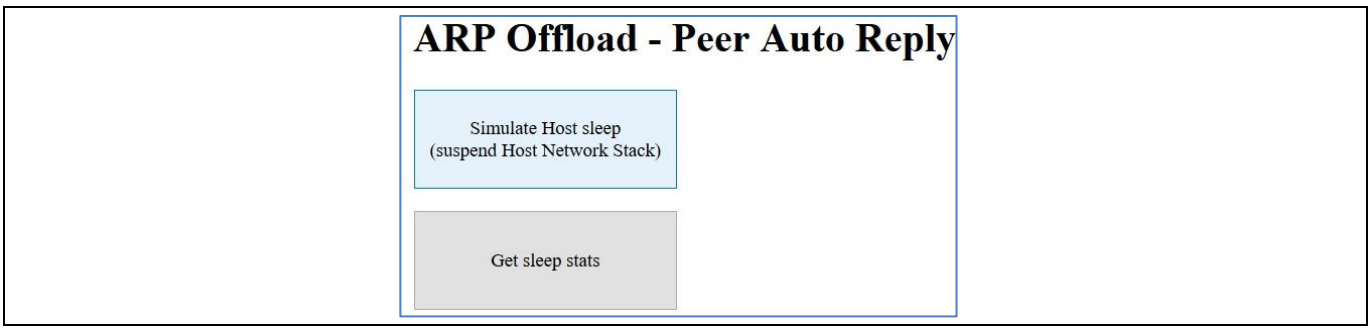

Figure 19 ARP offload web page

Power measurement using CY8CKIT-062S2-43012

5. Click the **Simulate Host Sleep** button on the webpage to suspend the network stack and put PSoC 6 MCU to deep sleep.
6. Do `arp-ping <ip-address>` from a command line or shell. With ARP offload enabled, PSoC 6 MCU should remain in deep sleep for arp-pings (**Figure 20**). Any other activity such as `ping <ip-address>` should wake the host. For comparison, **Figure 21** shows the scenario with ARP offload disabled. As seen in **Figure 20** and **Figure 21**, ARP offload reduces the power consumption of the host (PSoC 6 MCU) by over a 100-fold.

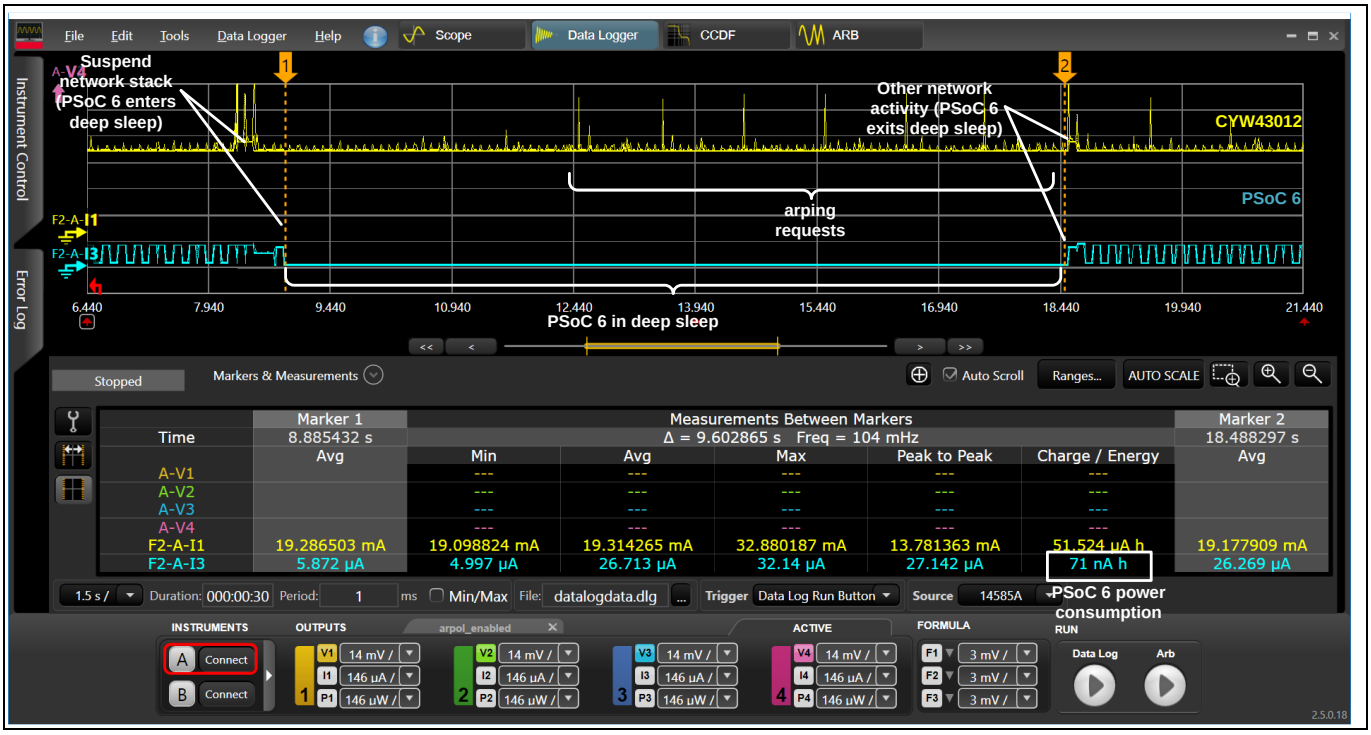

Figure 20 CY8CKIT-062S-43012 power consumption with ARP offload

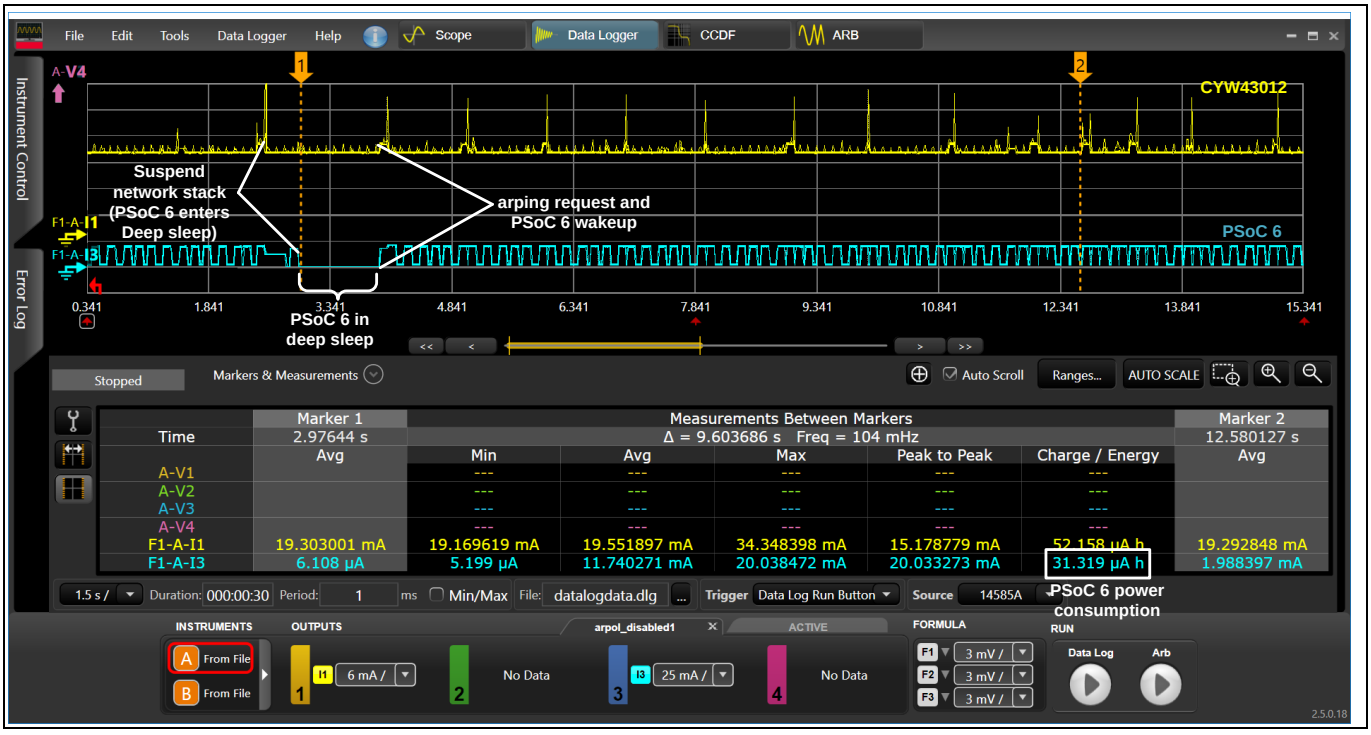

Figure 21 CY8CKIT-062S-43012 power consumption without ARP offload

8 Summary

PSoC 6 MCU and CYW43012 provide an industry-leading low-power IoT platform that provide many power management options. The Infineon Low Power Assistant middleware enables adding low-power to your IoT design with ease. By following proper methods and design techniques, you can optimize your system for the lowest possible power consumption without degrading performance.

Related Documents

For a comprehensive list of PSoC 6 MCU resources, see **KBA223067** in the Infineon community.

Application Notes

[1] **AN221774** – Getting Started with PSoC 6 MCU: Introduces the PSoC 6 MCU, a dual-CPU programmable system-on-chip

[2] **AN218241** – PSoC 6 MCU Hardware Design Considerations: How to design a hardware system around a PSoC 6 MCU device, including package selection, power, clocking, reset, I/O usage, and more

[3] **AN219528** – PSoC 6 MCU Low-Power Modes and Power Reduction Techniques: Describes how to use the PSoC 6 MCU power modes to optimize power consumption.

[4] **Getting Started with PSoC 6 MCU and CYW43xx in Mbed OS**: Describes how to develop applications in Mbed OS using PSoC 6 MCU and CYW43xx WLAN/Bluetooth combo devices.

Code Examples

[5] **mbed-os-example-wlan-lowpower**: Simple example that demonstrates WLAN low-power modes and host network suspend features

[6] **mbed-os-example-wlan-offload-arp**: Example demonstrates WLAN ARP offload feature

[7] **mbed-os-example-wlan-offload-packet-filter**: Example demonstrates WLAN packet filter offload with an option to evaluate filters in run-time

[8] **mbed-os-example-wlan-offload-discard-packet**: Example demonstrates a simple WLAN discard packet filter offload that discards ping packets

Visit the **Cypress GitHub site** for a comprehensive collection of code examples using Mbed OS

Device Documentation

[9] **PSoC 6 MCU: PSoC 62 Datasheet**

[10] **PSoC 6 MCU: PSoC 62 Architecture Technical Reference Manual**

[11] CYW43012 Datasheet

Development Kits and Documentation

[12] **CY8CKIT-062S2-43012 PSoC 6 Wi-Fi Bluetooth Pioneer kit (with CYW43012)**

Tools and Documentation

[13] **ModusToolbox Software**: The Infineon IDE for IoT designers

[14] PDL and Middleware: *<ModusToolbox install>libraries\psoc6sw-<version>\docs*

[15] **Low Power Assistant**: Low Power Assistant Middleware library

Revision history

Document version	Date of release	Description of changes
**	2019-12-06	Initial release
*A	2021-02-24	Updated in Infineon template

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

Edition 2021-02-24

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Go to cypress.com/support

Document reference
002-27910 Rev. *A

IMPORTANT NOTICE

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

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

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.