

Chapter 7: Low Power

After completing this chapter, you will understand how to use the ModusToolbox Low Power Assistant to improve power consumption of your device.

7.1	OVERVIEW	2
7.2	LOW POWER DOCUMENTATION AND COLLATERAL	3
7.2.1	APPLICATION NOTE	3
7.2.2	LOW POWER ASSISTANT DOCUMENTATION	3
7.2.3	MBED OS CODE EXAMPLES	3
7.2.4	MBED OS VERSION	3
7.3	GUIDE TO THIS CHAPTER.....	4
7.4	WHAT IS NOT COVERED IN THIS CHAPTER.....	4
7.5	HARDWARE SETUP AND POWER CONSUMPTION MEASUREMENTS	5
7.6	BASIC LOW POWER ASSISTANT FOR MBED OS.....	6
7.6.1	EXERCISE: SETUP BLINKY PROJECT AND MEASURE POWER CONSUMPTION	6
7.6.2	EXERCISE: VIEW THE DEFAULT DEVICE CONFIGURATION SHIPPED WITH MBED OS.....	6
7.6.3	EXERCISE: MODIFY THE DEFAULT DEVICE CONFIGURATION SHIPPED WITH MBED OS	8
7.7	WLAN LOW POWER ASSISTANT FOR MBED OS	12
7.7.1	EXERCISE: CONFIGURE HOST WAKE PIN FOR MBED-OS-EXAMPLE-WIFI.....	12
7.7.2	EXERCISE: MEASURE POWER FOR MBED-OS-EXAMPLE-WIFI: SENDING PING AND ARP REQUESTS	16
7.7.3	EXERCISE: CONFIGURING PACKET FILTER OFFLOAD	17
7.7.4	EXERCISE: ADDING ARP OFFLOAD	19
7.7.5	EXERCISE: LET THE ICMP (PING) GO THROUGH.....	21
7.8	EXPLORE MBED OS CODE EXAMPLES	22
7.8.1	WLAN Low-Power.....	22
7.8.2	ARP (ADDRESS RESOLUTION PROTOCOL) OFFLOAD	23
7.8.3	WLAN PACKET FILTER OFFLOAD DEMO.....	24
7.8.4	WLAN DISCARD PACKET FILTER OFFLOAD DEMO	24
7.9	LOW POWER ASSISTANT FOR PSOC 6 SDK	25
7.9.1	EXERCISE: MAKING HELLO WORLD ENERGY EFFICIENT	25
7.9.2	EXERCISE: IMPROVING POWER CONSUMPTION FOR THE CAPSENSE BUTTONS AND SLIDER EXAMPLE.....	30

7.1 Overview

If we think in the context of Low Power, not only do we want to be able to configure and control how every part of the system works, such as the MCU or Wi-Fi, but we also want configurability and control over how the parts of the system interact with each other. We also want seamless integration with the RTOS, peripherals, and connectivity subsystems.

Plus, it must work just fine in a real-world environment flooded with packets sent to and from hundreds of hotspots and Bluetooth devices around us, as well as other electromagnetic impulses we don't care about. "Works just fine" in the context of Low Power means the battery lasts as long as it is expected by the customer of your "Thing". Save more energy and you win.

ModusToolbox provides tools and middleware to set up and use the low power features of the PSoC 6 MCU and the connectivity devices on your board. We are referring to this set of tools and middleware as the Low Power Assistant. It consists of:

- **Device Configurator:** Used to configure the peripherals and system resources of the PSoC 6 MCU, configure the low power features of the connectivity device, such as wakeup pins, packet filters, offloads for Wi-Fi, as well as Bluetooth low power. The Device Configurator can also be used to configure the RTOS integration parameters such as the System Idle Power Mode for Mbed OS.
- **Low Power Assistant (LPA) middleware:** This consumes the configuration generated by the Device Configurator, then configures and handles the packet filters, offloads, host wake functionality, etc.

The Low Power Assistant feature is supported for all PSoC 6 MCUs as well as for the CYW43012 and 4343W connectivity modules. The combination of PSoC 6 and CYW43012 offers the lowest power consumption and is available on the CY8CKIT-062S2-43012.

7.2 Low Power Documentation and Collateral

The best starting point to learn about the Low Power Assistant feature and low power in general is the dedicated Application Note listed below. It includes the references to the code examples which use the LPA middleware. The LPA middleware offers online documentation with quick start guides that show you how to setup and test every feature.

Our exercises are mostly based upon the quick start guides offered in the [LPA Middleware API Reference Guide](#).

7.2.1 Application Note

The Application Note is an excellent source of information that you could have started from if you didn't go through this chapter. Reading it after this chapter will help you to refresh and reinforce the knowledge you have just gained.

- [AN227910 - Low-Power System Design with CYW43012 and PSoC 6 MCU](#) – link to public when available

7.2.2 Low Power Assistant Documentation

The [Cypress Low Power Assistant Middleware Library 1.0](#) includes set of Quick Start Guides for Mbed OS:

- [MCU Low Power](#)
- [Wi-Fi Host-wake Signal](#)
- [Wi-Fi Packet Filter Offload](#)
- [Wi-Fi Address Resolution Protocol Offload](#)
- [Bluetooth Low Power](#)

7.2.3 Mbed OS Code Examples

The following code examples provide low power examples for use with Mbed OS. Refer to the [Explore Mbed OS Code Examples](#) section for more details about these examples:

- <https://github.com/cypresssemiconductorco/mbed-os-example-wlan-lowpower>
- <https://github.com/cypresssemiconductorco/mbed-os-example-wlan-offload-arp>
- <https://github.com/cypresssemiconductorco/mbed-os-example-wlan-offload-packet-filter>
- <https://github.com/cypresssemiconductorco/mbed-os-example-wlan-offload-discard-packet>

Note: The file `mbed-os/targets/targets.json` contains a macro setting for `MBED_TICKLESS` for the `MCU_PSO4_M4` target. If you don't do this, the system will wake every 1ms when the tick timer expires. With this option, the tick timer is disabled before going to sleep and is re-enabled upon wake from sleep. For additional details on tickless mode, see: <https://os.mbed.com/docs/mbed-os/v5.15/porting/tickless.html>

7.2.4 Mbed OS Version

Mbed OS Version used for this documentation is **5.15.1**

7.3 Guide to this chapter

For this chapter, we will be performing exercises using the [CY8CKIT-062-WiFi-BT](#) kit, which includes the [PSoC 62 MCU](#) and the [Murata LBEE5KL1DX Module \(CYW4343W WiFi + Bluetooth Combo Chip\)](#).

The development kit guide explains how to measure power consumption and where to connect the probes. The first exercise will be to measure the power consumption of the Blinky Mbed OS example on the [CY8CKIT-062-WiFi-BT](#) kit.

The Low Power Assistant feature can be used in various flows and we are extending its capabilities:

- Mbed OS flow: Full feature functionality including PSoC 6 low power, RTOS integration, Packet Filters, and Offloads
- Other flows: PSoC 6 low power only

The general development flow in the context of Low Power remains the same as for any other design:

- Get Code Examples from <https://github.com/cypresssemiconductorco> or start from an empty project and add the Low Power Assistant middleware to your project.
- Configure the PSoC 6 MCU and connectivity low power using the Device Configurator.
- Flash the project and measure the power consumption.

7.4 What is not Covered in this chapter

This chapter is subject to further updates. As we continue to expand the Low Power Assistant solution, there will be additional features and exercises to learn. For now, these features are not covered:

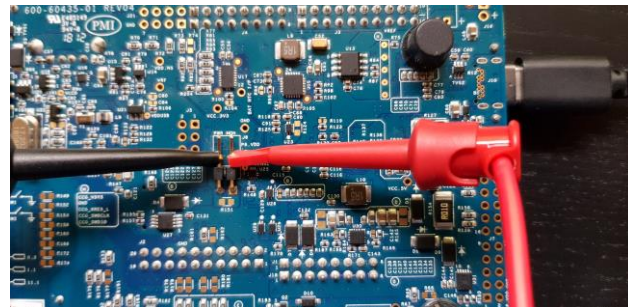
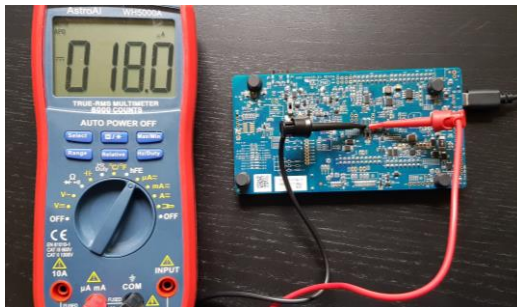
- Low Power Assistant for Bluetooth. You can go through the [Bluetooth Low Power](#) quick start guide in the [Cypress Low Power Assistant Middleware Library 1.0](#).
- Using Power Estimator Tool. You can go through the <https://github.com/cypresssemiconductorco/cype> example and use the Power Estimator tool available in your ModusToolbox Installation.

7.5 Hardware Setup and Power Consumption Measurements

For the purposes of this chapter, we will be using [CY8CKIT-062-WiFi-BT](#) board. We will be measuring the power consumption of the PSoC 6 MCU only. We will be performing the measurements using a multimeter. If available, it is recommended to use a power analyzer such as a KeySight [N6705B](#). The use of a KeySight N6705B with a CY8CKIT-062S2-43012 kit is explained in the low power system design application note referenced earlier.

If available, you could also use a CY8CKIT-062S2-43012 kit and measure the power consumption of both the PSoC 6 MCU and connectivity device.

In order to measure the power consumption on the [CY8CKIT-062-WiFi-BT](#) board configure the multimeter as an ammeter and connect the probes to the J8 jumper on the bottom side of the board in place of the shunt. You can use USB connection to the KitProg to supply power to the kit, but ensure you disconnect the board from the power supply before connecting and disconnecting the power measurement probes. Also detach the TFT shield from the board as we won't need it for the exercises.



Note: Do not disconnect the multimeter or USB connection to the KitProg. Keep everything connected throughout the exercises.

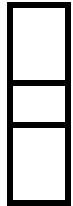
IMPORTANT: For Mbed OS exercises, you will be using the KitProg3 in DAPLink mode (Amber LED2 is ramping fast), and for the ModusToolbox IDE PSoC 6 SDK exercises you will be using KitProg3 in BULK or HID mode. Refer to Chapter 1 for more details about KitProg3 and the Firmware Loader.

7.6 Basic low power assistant for Mbed OS

In this section we will introduce the low power assistant and how it is used on Mbed OS.

7.6.1 Exercise: Setup Blinky Project and Measure Power Consumption

In this exercise we will measure the power consumption for the mbed-os blinky project just to learn in an example. We will also learn how to make modifications to the default device configuration of Cypress kits shipped within Mbed OS without modifications to the Mbed OS itself.



1. Disconnect the board from your PC and connect the multimeter probes as explained above.
2. Ensure KitProg3 is in DAPLink mode.
3. Import, update to the latest mbed-os, compile and flash the mbed-os-example-blinky example.

```
mbed import mbed-os-example-blinky
cd mbed-os-example-blinky
cd mbed-os
mbed update latest
cd ..
mbed deploy
mbed compile --toolchain GCC_ARM --target CY8CKIT_062_WIFI_BT --flash
```



4. Observe mA or uA (further referenced as “power consumption” for simplicity) displayed on the multimeter.

7.6.2 Exercise: View the Default Device Configuration Shipped with Mbed OS

Mbed OS includes Board Support Packages (BSPs) for the Cypress kits out of the box. So, to use any of the Mbed OS examples, you can just specify the correct command line argument to match the kit you are working with. In our case it is:

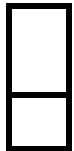
```
--target CY8CKIT_062_WIFI_BT
```

The BSP shipped with Mbed OS includes the design.modus file and the generated source files. This is the default configuration shipped for the kit. You don't want to modify this configuration directly because it will introduce conflicts in your Git repository, and it will complicate updates to a newer version of Mbed OS.

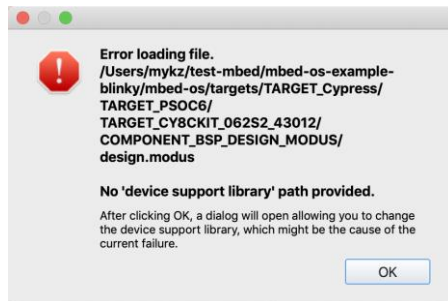
However, there is a way to override the default configuration, and this is what you need to do in order to adjust the configuration to your own needs, including but not limited to the configuration for low power. Later sections will explain how to properly make these changes to the configuration, but for this exercise we will just open the configuration using the Device Configurator, and we will **NOT** make any modifications to it.



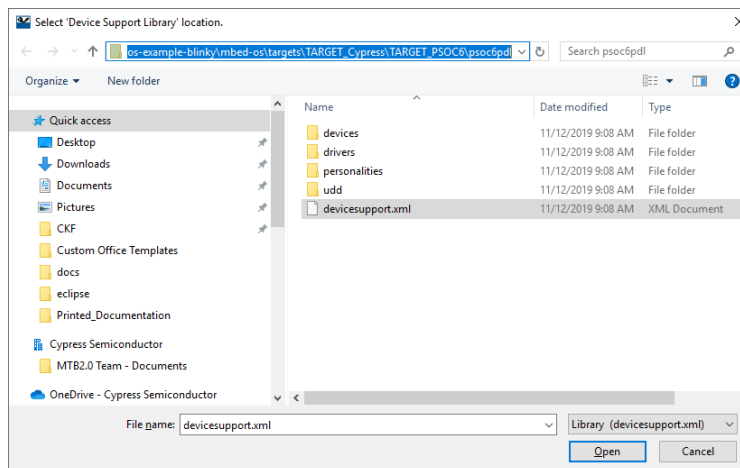
1. In the previously created mbed-os-blinky example, go to the mbed-os/targets directory.
Find the design.modus file for the kit we are working with: CY8CKIT_062_WIFI_BT. What other design.* files do you see? What do you see in the GeneratedSources directory?



2. Start the Device Configurator located in the tools_2.0 directory of your ModusToolbox 2.0 installation.
3. Open the design file with the Device Configurator:
`mbed-os-example-blinky/mbed-os/targets/TARGET_Cypress/TARGET_PSOC6/TARGET_CY8CKIT_062_WIFI_BT/COMPONENT_BSP_DESIGN_MODUS/design.modus`.
Notice the error that that Device Configurator cannot find the 'device support library'. This must be fixed manually for now.



4. Click **OK** and a dialog opens.

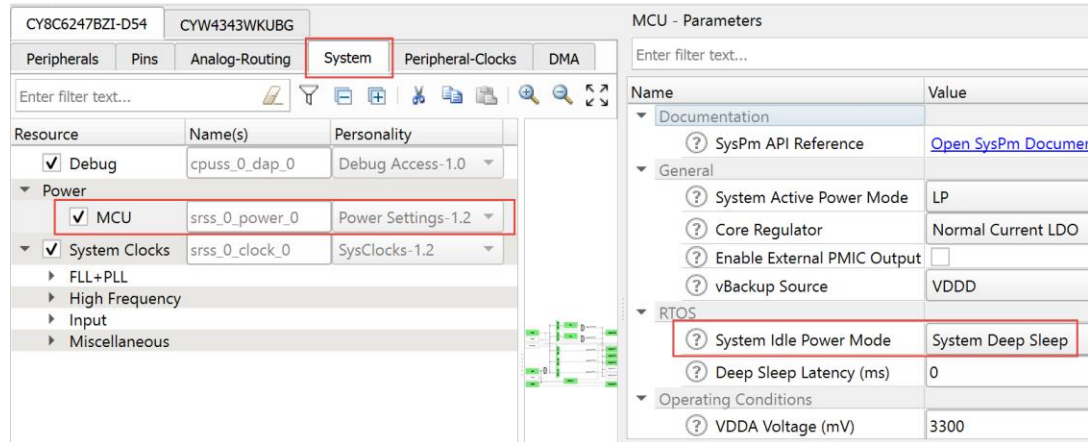


5. On the dialog, navigate to and select `<examples_dir>/mbed-os-example-blinky/mbed-os/targets/TARGET_Cypress/TARGET_PSOC6/psoc6pdl/devicesupport.xml`.

The Device Configurator will open and show the configuration of your device.



- In the Device Configurator, go to the **System** tab, select **Power > MCU** from the resource list and check the setting for the **System Idle Power Mode** in the RTOS category.



Note that it is already set to ensure minimal power consumption: System Deep Sleep.



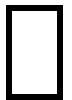
- Do not save any changes. Close the Device Configurator.

Now you know how to see the default configuration shipped for our kits in Mbed OS.

7.6.3 Exercise: Modify the Default Device Configuration Shipped with Mbed OS

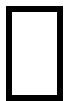
The examples we provide for Cypress kits at <https://github.com/cypresssemiconductorco> already assume that you might want to make the changes to the device configuration. So, they allow you to override that default device configuration with one stored at the application level.

If you start from the general Mbed OS examples (mbed-os-example-blinky is one of those) you must make these tweaks to get the project ready for configuration changes. Now we'll modify our Blinky project so that we can introduce further changes using the Device Configurator.



- Make a directory for your target at the root of your application and change to that directory.

```
mkdir TARGET_CY8CKIT_062_WIFI_BT
cd TARGET_CY8CKIT_062_WIFI_BT
```



- Copy all files from the COMPONENT_BSP_DESIGN_MODUS folder in the Mbed BSP to your new folder.

```
cp -r ../mbed-
os/targets/TARGET_Cypress/TARGET_PSOC6/TARGET_CY8CKIT_062_WIF
I_BT/COMPONENT_BSP_DESIGN_MODUS/* .
```



- Delete the design.cyqspi and design.cycapsense files.

Note: You don't need these files for this exercise. Later on, you can pull them from the original BSP shipped with Mbed OS, if needed.

4. Go back to the main application directory (up one level) and add a file called `mbed_app.json` with the following contents:

```
{
  "target_overrides": {
    "*": {
      "target.components_remove": ["BSP_DESIGN_MODUS"],
      "target.components_add": ["CUSTOM_DESIGN_MODUS"]
    }
  }
}
```

Note: The first line specifies to ignore the BSP configuration shipped with Mbed OS. The custom one will be added since the folder it is in matches the TARGET name that you will use when building the project. The second line is required so that the `init_cycfg_all` function is still called from the `cybsp_init` function. The `init_cycfg_all` function is used to initialize the hardware that was setup in the configuration files

5. Replace the code in `main.cpp` file with the following. This will alternate between active mode (LED ON) and sleep mode (LED OFF) every 5 seconds.

```
#include "mbed.h"
#include "platform/mbed_thread.h"

int main()
{
  DigitalOut led(LED1);
  // Duration of Active/Idle states in milliseconds.
  const int stateTime = 5000;
  while (true) {
    // Turn on the onboard Led to indicate the thread Active state.
    led = 0;
    // The Cy_SysLib_Delay() implements a loop that blocks further
    // code execution for the specified period of time.
    // This call is used to keep the thread in the Active state long
    // enough to evaluate the PSoC power consumption in the Active state.
    Cy_SysLib_Delay(stateTime);
    // Turn off the onboard LED to indicate thread Idle state.
    led = 1;
    // Put the thread into the Idle state to evaluate the PSoC power
    // consumption in the configured System Idle Power mode state.
    ThisThread::sleep_for (stateTime);
  }
}
```

6. Connect the multimeter for power measurements and connect the board to USB.

You can keep the multimeter connected while reprogramming the board and throughout the exercises.

We won't start a terminal here because we don't need it. However, for later exercises, you will need to add the option `--stern` to the following command:

```
mbed compile --target CY8CKIT_062_WIFI_BT --toolchain GCC_ARM -flash
```

7. Observe power consumption during active and sleep (remember, the LED will be ON during active and OFF during sleep).

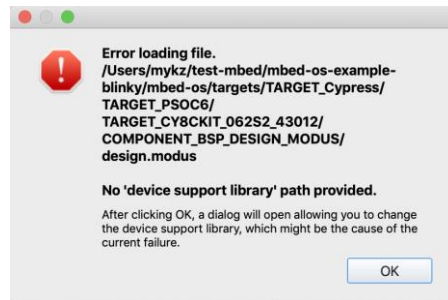
Now we can make further changes using the Device Configurator.

8. Start the Device Configurator located in the tools_2.0 directory of your ModusToolbox 2.0 installation.
9. Open the “custom” design file with the Device Configurator:

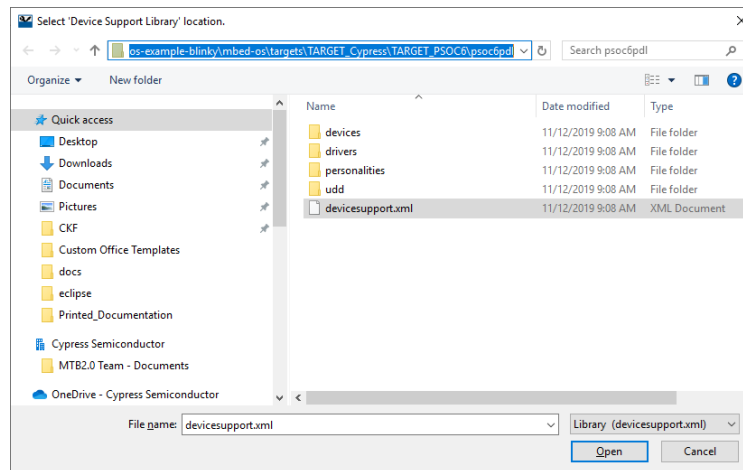
mbed-os-example-blinky/TARGET_CY8CKIT_062_WIFI_BT/design.modus

Note: This is the design file in your application, not the default one shipped with Mbed OS.

Notice the error that that Device Configurator cannot find the ‘device support library’. This must be fixed manually for now.



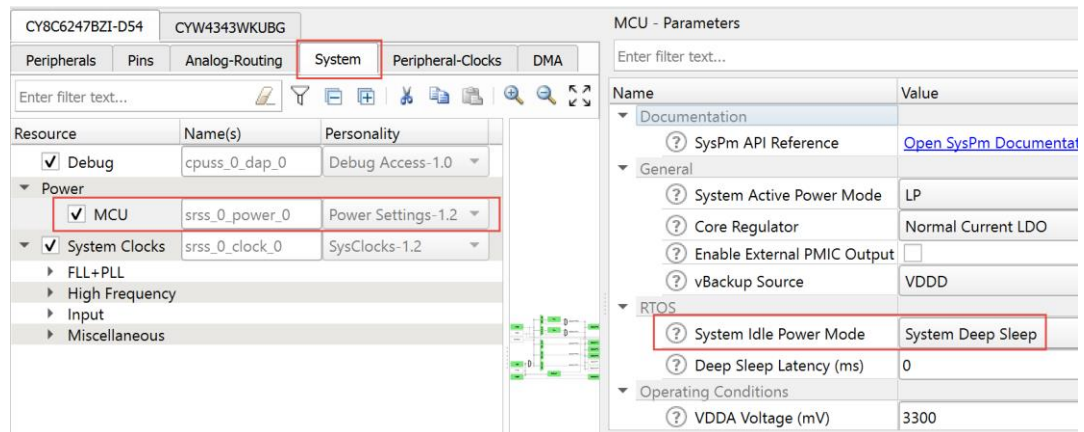
10. Click **OK** and a dialog opens.



11. On the dialog, navigate to and select <examples_dir>/mbed-os-example-blinky/mbed-os/targets/TARGET_Cypress/TARGET_PSO6/psoc6pdl/devicesupport.xml.

The Device Configurator will open and show the configuration of your device.

12. Go to the **System** tab and check the setting for the **System Idle Power Mode** in the RTOS category.



13. Change it to “CPU Sleep”.
14. Save the configuration to generate the necessary code.
15. Compile and flash your project:

```
mbd compile --target CY8CKIT_062_WIFI_BT --toolchain GCC_ARM --flash
```

16. Observe power consumption during active and sleep. How did it change from the previous measurements?

You now see the effect of your changes to the copy of the design.modus you have created at the application level. The original configuration shipped for this kit with Mbed OS remains intact and you can use it as the reference in your further experiments.

7.7 WLAN Low Power Assistant for Mbed OS

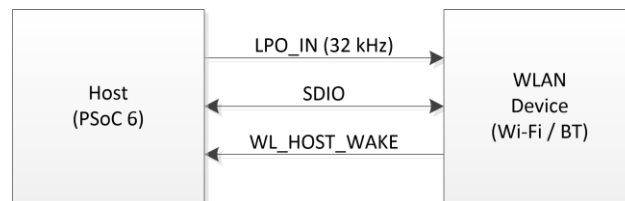
In this section we will learn how the Low Power Assistant helps you to minimize the MCU power consumption in a system connected to Wi-Fi. First, we will learn how to configure the HOST WAKE PIN, so that the network traffic can wake up the MCU. Then we will try to minimize the amount of time the MCU needs to be awake by filtering certain packets and handling them without the MCU involvement.

7.7.1 Exercise: Configure Host Wake Pin for mbed-os-example-wifi

In this exercise we will setup the Host Wake signal. We will also prepare the project for further modifications just like we did for the mbed-os-example-blinky. During this and further exercises keep the multimeter connected to your system. Also keep the USB connection.

Host-wake provides a way for a WLAN device to wake up the Host MCU from its low-power state. Host-wake is implemented using a GPIO on the MCU that is connected to the WLAN device. The WLAN device asserts the host-wake GPIO to bring the host MCU out of its low-power state. This interrupt is called an out-of-band (OOB) interrupt.

Refer to the [Wifi-Host Wake Quick Start Guide](#) for additional information.



1. Import the mbed-os-example-wifi example project and switch to the example directory:

```

mbed import mbed-os-example-wifi
cd mbed-os-example-wifi

```



2. Download latest Mbed OS, the LPA middleware, and connectivity utilities, and then deploy.

```

cd mbed-os
mbed update latest
cd ..
mbed add http://github.com/cypresssemiconductorco/lpa
mbed add http://github.com/cypresssemiconductorco/connectivity-utilities
mbed deploy

```



3. Make a directory for your target at the root of your application and change to that directory.

```

mkdir TARGET_CY8CKIT_062_WIFI_BT
cd TARGET_CY8CKIT_062_WIFI_BT

```



4. Copy all files from the COMPONENT_BSP_DESIGN_MODUS folder in the Mbed BSP to your new folder.

```
cp -r ../mbed-
os/targets/TARGET_Cypress/TARGET_PSOC6/TARGET_CY8CKIT_062_WIF
I_BT/COMPONENT_BSP_DESIGN_MODUS/* .
```

Delete the design.cyqspi and design.cycapsense files.

Note: You don't need these files for this exercise. Later on, you can pull them from the original BSP shipped with Mbed OS, if needed.



5. Go back to the main application directory (one level up) and edit the existing mbed_app.json file.

Exclude the BSP_DESIGN_MODUS folder and add CUSTOM_DESIGN_MODUS by adding the two lines shown in bold below to the target overrides section. Note you must also add a comma at the end of the line that says *"platform.stdio-convert-newlines": true*.

```
. . .
{
  "target_overrides": {
    "**": {
      "platform.stdio-convert-newlines": true,
      "target.components_remove": ["BSP_DESIGN_MODUS"],
      "target.components_add": ["CUSTOM_DESIGN_MODUS"]
    }
  }
}
```



6. Modify the mbed_app.json file to update the "wifi-ssid" and "wifi-password" values as needed:

```
. . .
  "wifi-ssid": {
    "help": "WiFi SSID",
    "value": "\"SSID\""
  },
  "wifi-password": {
    "help": "WiFi Password",
    "value": "\"PASSWORD\""
  }
. . .
```



7. Copy the following files from the lpa directory to the same directory as the main.cpp file.

This step is necessary for the first release of LPA and will be eliminated in the future. These files provide functions to suspend the network stack when there is no network activity and resumes it when there is network activity.

```
lpa/helpers/net_suspend/mbedos/network_activity_handler.cpp
lpa/helpers/net_suspend/mbedos/network_activity_handler.h
```



8. Include WhdSTAInterface.h and network_activity_handler.h at the top of main.cpp

```
#include "WhdSTAInterface.h"
#include "network_activity_handler.h"
```



9. Add the following declarations to main.cpp:

```
// This macro specifies the interval in milliseconds that the device monitors
// the network for inactivity. If the network is inactive for duration lesser
// than INACTIVE_WINDOW_MS in this interval, the MCU does not suspend the network
// stack and informs the calling function that the MCU wait period timed out
// while waiting for network to become inactive.
#define INACTIVE_INTERVAL_MS    (300u)
// This macro specifies the continuous duration in milliseconds for which the
// network has to be inactive. If the network is inactive for this duration, the
// MCU will suspend the network stack. Now, the MCU will not need to service the
// network timers which allows it to stay longer in sleep/deepsleep.
#define INACTIVE_WINDOW_MS      (200u)
```



10. Modify main.cpp to go to deep sleep after connecting to the Wi-Fi Access Point.

Mbed OS uses lwIP as the network stack. lwIP has periodic timers (smallest timer ~100ms), which wake up the Host MCU periodically. Do the following to suspend these timers. This is needed to demonstrate/verify that WLAN_HOST_WAKE pin is properly configured, and that the Host MCU does not wake up from any other reason other than OOB from WLAN device due to Network activity.

Modify main.cpp to allow the host MCU to go to deep-sleep and stay in deep sleep until woken up by an external event. with the following code:

Replace the line: `wifi->disconnect();`

With this code:

```
while(true) {
    wait_net_suspend(static_cast<WhdSTAInterface*>(wifi),
        osWaitForever, INACTIVE_INTERVAL_MS, INACTIVE_WINDOW_MS);
}
```



11. Open the design file with the Device Configurator (the one at the application level):

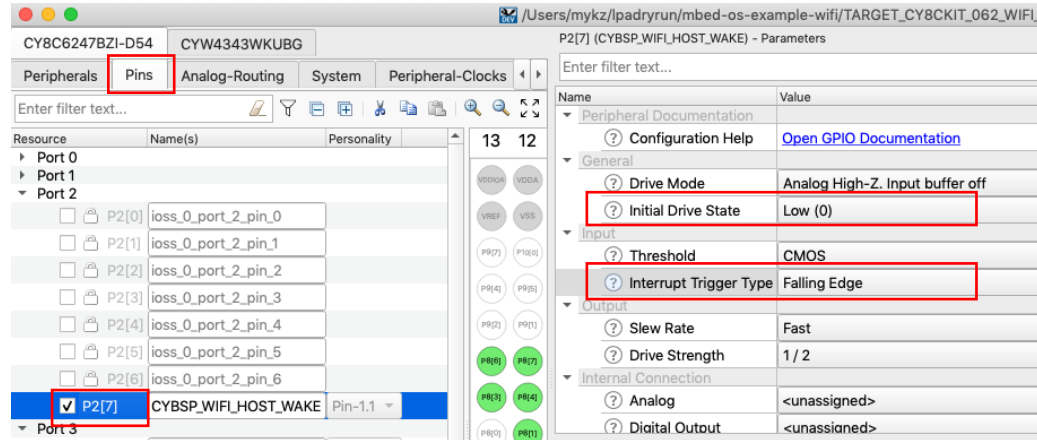
`mbed-os-example-wifi/TARGET_CY8CKIT_062_WIFI_BT/design.modus`

The Device Configurator will open and show the configuration of your device.



12. Switch to the **Pins** tab.

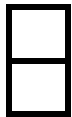
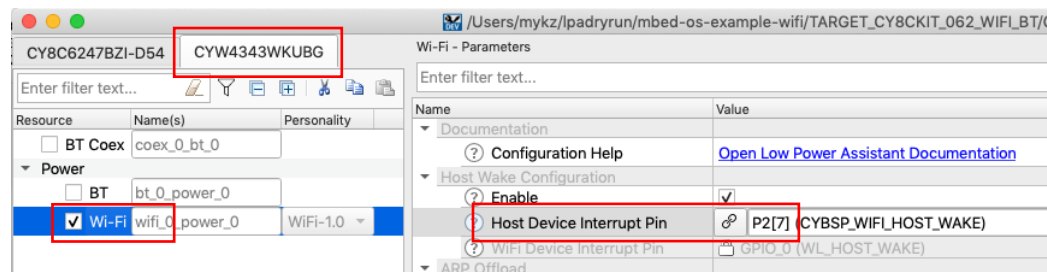
Enable pin P2[7] if it isn't already enabled. In the **Parameters** pane, change **Initial Drive State** set to "Low (0)" and **Interrupt Trigger Type** to "Falling Edge".



13. Switch to the **CYW4343WKUBG** connectivity device tab.

Enable **Power->Wi-Fi**.

In the **Wi-Fi - Parameters** pane, ensure **Host Wake Configuration** is enabled and set **Host Device Interrupt Pin** to P2[7]



14. Save the configuration to generate the necessary code.

15. Build the project and program the board, also start a terminal:

```
mbd compile --target CY8CKIT_062_WIFI_BT -DMBED_CPU_STATS_ENABLED
--toolchain GCC_ARM --flash --stern
```

Note: If you forget to open the terminal, just use this command once programming is done:

```
mbd stern
```

MBED_CPU_STATS_ENABLED: This helps to understand how much time MCU Remained in Sleep state



16. Note the IP address and get ready for the next exercise (sending PING and ARP packets).

7.7.2 Exercise: Measure Power for mbed-os-example-wifi: sending PING and ARP Requests

In this exercise we will learn how to send the PING and ARP (Address Resolution Protocol) packets.



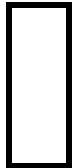
1. Download the arp-ping tool for your system.

- o Windows: <https://www.elifulkerson.com/projects/arp-ping.php>
- o Mac : <http://macappstore.org/arping/>
- o Linux : `sudo apt install arping`



2. Skip this step if you still have the board programmed, connected and terminal running

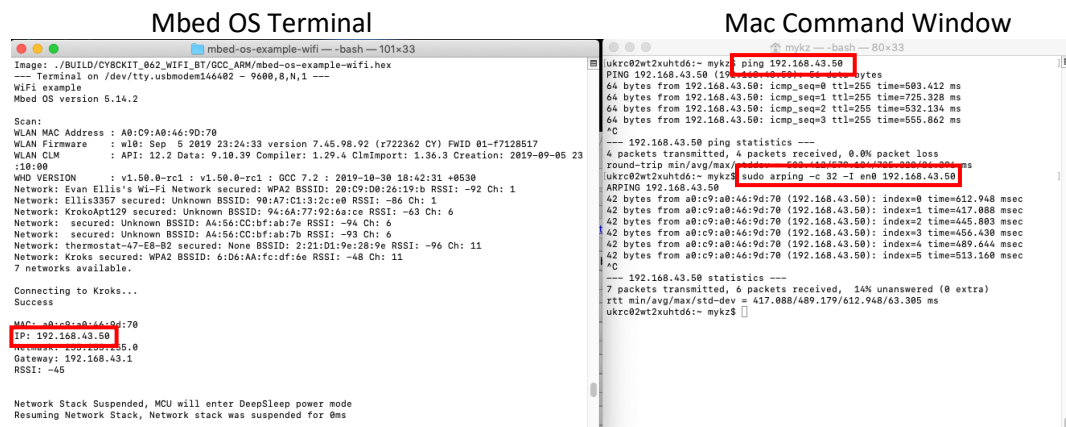
```
mbed compile --target CY8CKIT_062_WIFI_BT -DMBED_CPU_STATS_ENABLED -
-toolchain GCC_ARM --flash --sterm
```



3. For Mac/Linux, open a terminal . For Windows, open a command window or power shell window from the folder where you download the arp-ping tool. You will send commands from this window and will observe in the Mbed OS terminal how the device responds to the packets sent .

IMPORTANT: Ensure your laptop and device are both connected to the same network.

IMPORTANT: Ensure your laptop and device are both connected to a 2.4 GHz network.



4. Send ping packets to your device, for Mac (use the IP of your device)

```
ping 192.168.43.50
```



5. Send ARP packets to your device (use the IP of your device, send 15 times for example)

Windows:

```
.\arp-ping.exe -n 15 192.168.43.50
```

Mac/Linux:

```
sudo arping -c 15 -i en0 192.168.43.50
```

Note: Do “ifconfig” in terminal and find the interface for Wi-Fi connection and modify “en0” to proper interface name.



6. Did the device respond to both PING and ARP? _____



7. How does the power consumption change in response to you sending packets? What happens with the power consumption if you are sending the packets continuously?
-

7.7.3 Exercise: Configuring Packet Filter Offload

Packet filters allow the host processor to limit which types of packets get passed up to the host from the WLAN subsystem. This is useful to keep out unwanted / unneeded packets from the network that might otherwise wake the host out of a power-saving System Deep Sleep mode or prevent it from entering System Deep Sleep mode.

Packet filters are useful when:

- Trying to keep the host processor in System Deep Sleep for as long as possible.
- Trying to get the host processor into System Deep Sleep as soon as possible.

Whenever a WLAN packet is destined for the host, the WLAN processor must wake the host (if it is asleep) so it 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. For example, it 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, so the host is not woken.

Refer to the [Packet Filters Quick Start Guide](#) for additional information.



1. Open the design file with the Device Configurator:
mbed-os-example-wifi/TARGET_CY8CKIT_062_WIFI_BT/design.modus



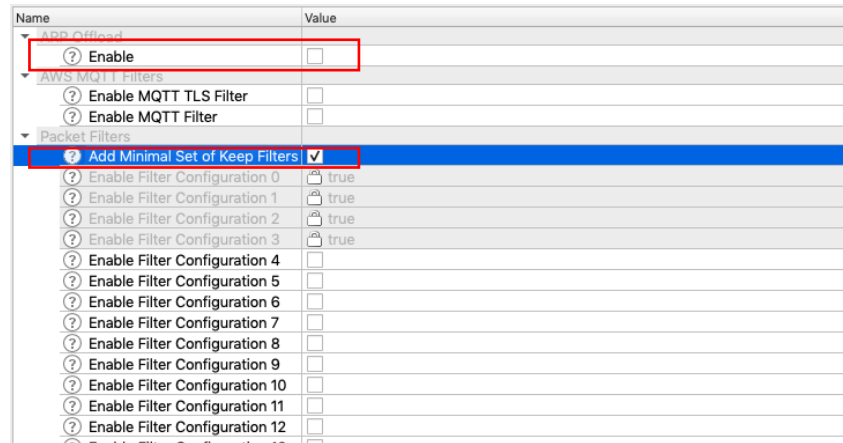
2. Switch to the connectivity device tab.

Ensure **ARP offload** is disabled.

Enable **Add minimal set of keep filters**.

These filters allow ARP, DNS, DHCP and 802.11x security packets to wake up the Host.

These are needed to connect to a Wi-Fi Access point. Any other Wi-Fi packets are dropped by the WLAN chip and not forwarded to the Host MCU (PSoC 6).



3. Save the configuration to generate the necessary code.



4. Build the project and program the board, also start a terminal:

```
mbd compile --target CY8CKIT_062_WIFI_BT -DMBED_CPU_STATS_ENABLED -
-toolchain GCC_ARM --flash -sterm
```

MBED_CPU_STATS_ENABLED: This helps to understand how much time MCU remained in Sleep state



5. Send a "ping" command to the board.

Observe in the serial terminal that it does not wake up the PSoC 6 device since there is no "keep" packet filter for ICMP pings. Also observe that there is no response for the pings. Observe the power consumption to see that the PSoC 6 MCU remains in Deep Sleep mode.



6. Send an "arping" command and observe that:

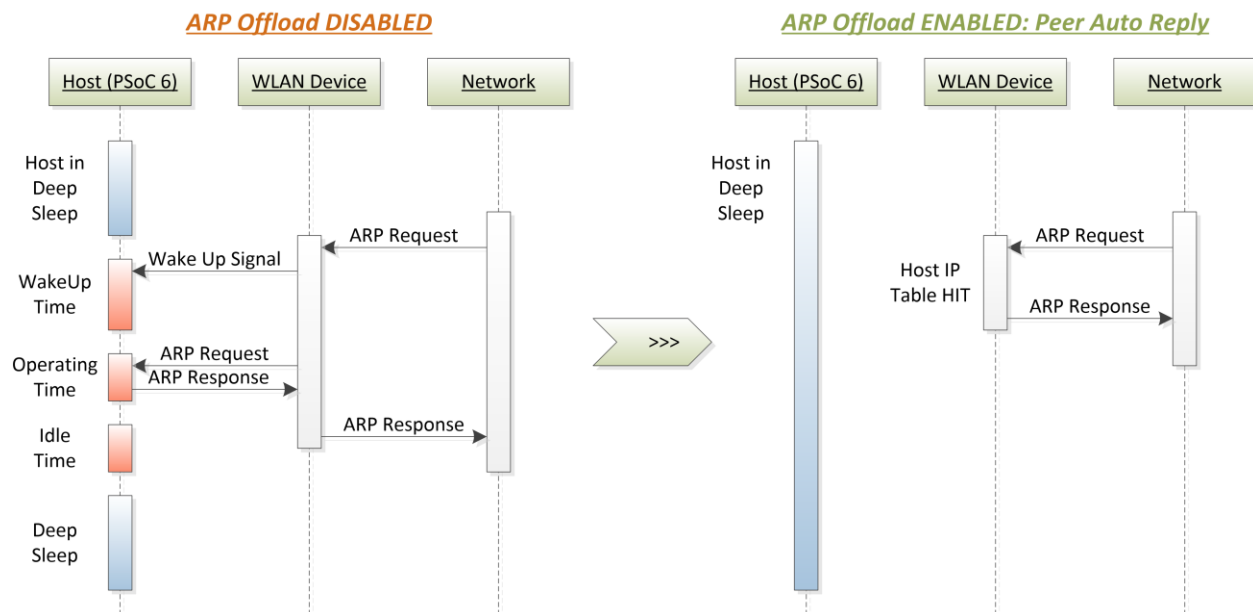
- You get the responses
- Observe the power consumption to see that the PSoC 6 MCU wakes up

You can see that the packet filter is not letting PING packets through, but the ARP packets wake up the PSoC 6 MCU.

7.7.4 Exercise: Adding ARP Offload

ARP is used for mapping an Internet Protocol address (IP address; ex: 192.168.1.1) to a physical machine address (MAC; ex: ac:32:df:14:16:07). ARP uses broadcast frames to accomplish this.

ARP broadcast traffic is normally forwarded from the Network to the Device (Wi-Fi Radio) and then to the Host (Application CPU) Network Stack. If the Host is sleeping, the Device wakes it up. Having the Device handle some of the ARP traffic will reduce the frequency that the Host sleeps/wakes up, reducing Host power consumption by staying in CPU Sleep and System Deep Sleep states longer. Having the Device handle some of the ARP requests from the Host to the network will reduce Network traffic (Host Auto Reply).



Refer to the [ARP Offload features](#) to understand different configurations.

Host Auto Reply

Host Auto Reply is a power-saving and network traffic reduction feature. Using the ARP Offload Host Auto Reply feature, the Device will answer Host ARP Requests, without broadcasting to the Network.

Peer Auto Reply

Peer Auto Reply is another power-saving feature so that the Device can respond to ARP requests from Peers (Network) without waking up the Host Processor.

Host IP Snoop

When enabled, the Snoop facility watches for ARP Responses from the Host to the Network, and caches them in the WLAN Device Host IP Table. The size of this table is 8 entries, which allows for the Device to support multiple IP addresses.

ARP Offload Cache Entry Expiry

This is an "age out" value that you can set in the ARP Offload configuration to determine the length of time the ARP cache entry is valid. This ensures that the WLAN ARP cache is updated appropriately.

Steps

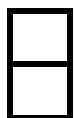


1. Open the design file with the Device Configurator.
mbed-os-example-wifi/TARGET_CY8CKIT_062_WIFI_BT/design.modus



2. Switch to the connectivity device tab.
 Enable **ARP offload**.
 Set **ARP offload Feature(s)** to "Peer Auto Reply".
 Enable **Snoop Host IP From Traffic When ARP Offload Enabled**.
 Set **ARP Offload Cache Entries Expire after (s)** to "1200".
 Keep Packet Filters as configured.

ARP Offload	
Enable	<input checked="" type="checkbox"/>
ARP Offload Feature(s)	Peer Auto Reply
Snoop Host IP From Traffic When ARP Offload Enabled	<input checked="" type="checkbox"/>
ARP Offload Cache Entries Expire After (s)	1200
AWS MQTT Filters	
Enable MQTT TLS Filter	<input type="checkbox"/>
Enable MQTT Filter	<input type="checkbox"/>
Packet Filters	
Add Minimal Set of Keep Filters	<input checked="" type="checkbox"/>
Enable Filter Configuration 0	<input checked="" type="checkbox"/>
Enable Filter Configuration 1	<input checked="" type="checkbox"/>
Enable Filter Configuration 2	<input checked="" type="checkbox"/>
Enable Filter Configuration 3	<input checked="" type="checkbox"/>
Enable Filter Configuration 4	<input checked="" type="checkbox"/>
Enable Filter Configuration 5	<input type="checkbox"/>
Enable Filter Configuration 6	<input type="checkbox"/>
Enable Filter Configuration 7	<input type="checkbox"/>
Enable Filter Configuration 8	<input type="checkbox"/>
Enable Filter Configuration 9	<input type="checkbox"/>
Enable Filter Configuration 10	<input type="checkbox"/>
Enable Filter Configuration 11	<input type="checkbox"/>



3. Save the configuration to generate the necessary code.



4. Build the project and program the board, also start a terminal:

```
mbed compile --target CY8CKIT_062_WIFI_BT -DMBED_CPU_STATS_ENABLED -
-toolchain GCC_ARM --flash -sterm
```

MBED_CPU_STATS_ENABLED: This helps to understand how much time MCU remained in Sleep state



5. Send a "ping" command to the board and observe no change in behavior.

In serial terminal, you will see that it does not wake up the PSoC 6 device since there is no "keep" packet filter for ICMP pings. Observe the power consumption to see that the PSoC 6 MCU remains in Deep Sleep mode.



6. Send an "arping" command.

- Observe that you still get the responses.
- Observe the power consumption to see that the PSoC 6 MCU no longer wakes up.

The PSoC 6 (Host) is offloaded from ARP packet handling and can remain in sleep.

7.7.5 Exercise: Let the ICMP (Ping) Go Through

This explains how to modify Packet filters so that ICMP Ping packets are not filtered by WLAN and sent to host.



1. Open the design file with the Device Configurator.

mbed-os-example-wifi/TARGET_CY8CKIT_062_WIFI_BT/design.modus



2. Switch to the connectivity device tab.

Enable new filter (Set check box at **Enable Filter Configuration 4.**)

▼ Packet Filters	
ⓘ Add Minimal Set of Keep Filters	<input checked="" type="checkbox"/>
ⓘ Enable Filter Configuration 0	<input type="checkbox"/> true
ⓘ Enable Filter Configuration 1	<input type="checkbox"/> true
ⓘ Enable Filter Configuration 2	<input type="checkbox"/> true
ⓘ Enable Filter Configuration 3	<input type="checkbox"/> true
ⓘ Enable Filter Configuration 4	<input checked="" type="checkbox"/>
ⓘ Enable Filter Configuration 5	<input type="checkbox"/>



3. Scroll down and set the parameters as follows. Refer to section : [Filter Types](#).

Filter Type set to "IP Type" (IP Protocol Filter)

IP Protocol set to "1" (1 refers to ICMP packet : [List of IP protocol numbers](#))

▼ Packet Filter Configuration 4	
ⓘ Filter ID	4
ⓘ Filter Type	IP Type
ⓘ Action	Keep
ⓘ When Active	Always
ⓘ IP Protocol	1



4. Save the configuration to generate the necessary code.



5. Build the project and program the board, also start terminal:

```
mbed compile --target CY8CKIT_062_WIFI_BT -DMBED_CPU_STATS_ENABLED
--toolchain GCC_ARM --flash -sterm
```

MBED_CPU_STATS_ENABLED: This helps to understand how much time MCU remained in Sleep state



6. Send a "ping" command to the board and observe in serial terminal.

It now wakes the Host up because you have just added a "keep" packet filter for ICMP pings. You can see the response in the terminal and can observe increased power consumption upon receiving the ping packet.



7. Send an "arping" command.

Behavior did not change; you get the responses to ARP in terminal while the PSoC 6 MCU remains in sleep.



8. Why did we set the IP Protocol to 1? Refer to the [Packet Filters Quick Start Guide](#).
-

7.8 Explore Mbed OS Code Examples

In addition to the Quick Start Guides in the [Cypress Low Power Assistant Middleware Library 1.0](#) online documentation, we also offer a set of the examples for you to exercise the Low Power Assistant in Mbed OS. Examples include more explanation of the features you now know how to use. They implement webpages hosted on a device allowing you to configure and test things.

The code examples work out of the box (they already have all the design.modus customization done for you.) All you need to do is to follow the simple steps available in the Readme of each code example. Just remember to update the mbed_app.json file with the Wi-Fi credentials.

After programming the example, jump to the "Verify" section available.

7.8.1 WLAN Low-Power

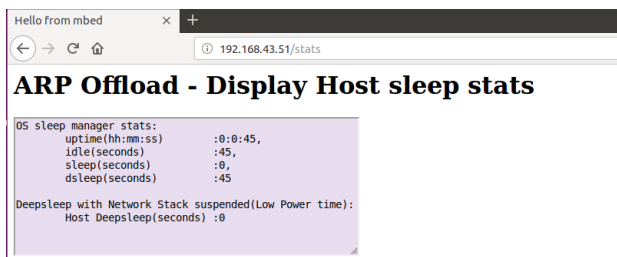
<https://github.com/cypresssemiconductorco/mbed-os-example-wlan-lowpower>

This code example demonstrates low power operation of the Host MCU and the WLAN device. The example connects to a network whose credentials are provided in mbed_app.json. After connecting to the network successfully, the example starts an HTTP server. The example configures the WLAN device in the specified power-save mode and sets the host MCU in an extended wait state, waiting for HTTP requests. The host MCU is in Deep Sleep or Sleep mode during the extended wait state.

7.8.2 ARP (Address Resolution Protocol) Offload

<https://github.com/cypresssemiconductorco/mbed-os-example-wlan-offload-arp>

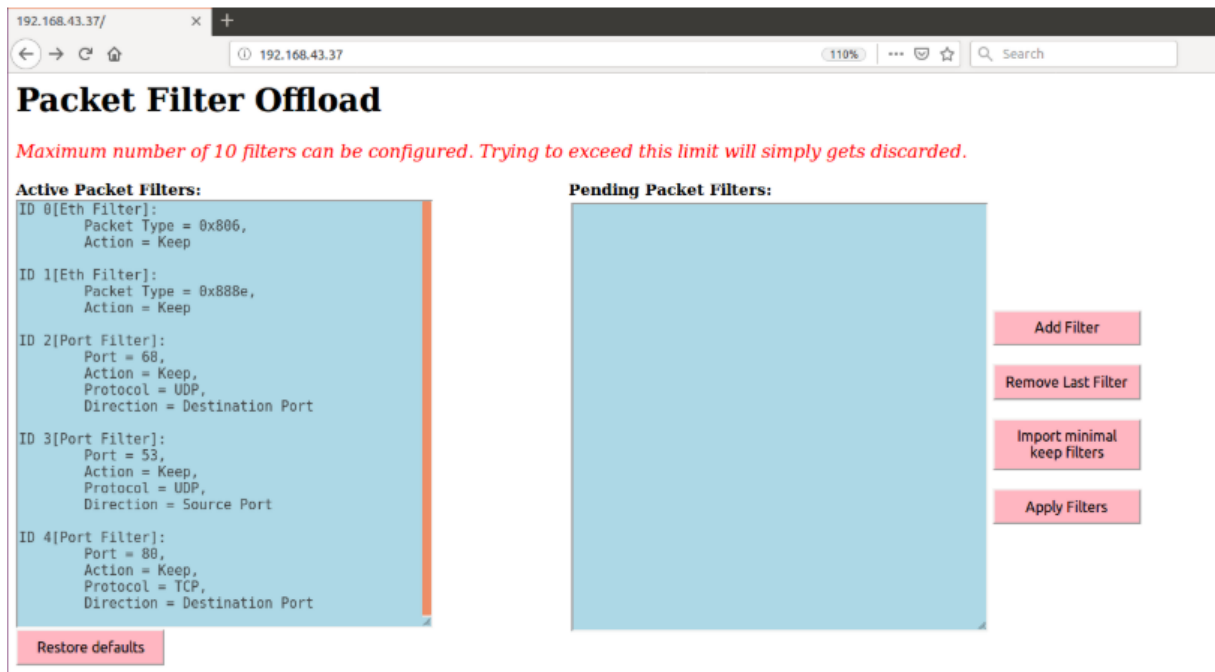
This code example demonstrates the ARP Offload functionality offered by the Cypress WLAN device while the PSoC 6 MCU is in sleep/deep sleep. Try the home webpage for the ARP Offload application that is hosted on your device. The home page has two web buttons: Simulate Host Sleep and Get Sleep Stats. Get Sleep Stats to see the Mbed OS sleep statistics such as uptime, idle, sleep, and deep sleep time.



7.8.3 WLAN Packet Filter Offload Demo

<https://github.com/cypresssemiconductorco/mbed-os-example-wlan-offload-packet-filter>

This code example demonstrates the Packet Filter Offload functionality offered by the Cypress Wi-Fi chips. This application uses "Ping-Pong" buffer logic. One of the buffers is used to hold Active packet filters configuration that is applied to the WLAN device. The other buffer is used to keep track of the Pending packet filters which you can add to the list and apply the configuration.



7.8.4 WLAN Discard Packet Filter Offload demo

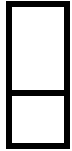
<https://github.com/cypresssemiconductorco/mbed-os-example-wlan-offload-discard-packet>

This application demonstrates the discard packet filter offload. The application is configured with a discard packet filter for the ICMP packet type using ModusToolbox 2.0 Device Configurator tool so that the WLAN will discard ICMP packets that are trying to reach the host.



5. Build and Program the board. Ensure KitProg3 is in HID or BULK mode.

After programming, the application starts automatically. Verify that the application works as expected: LED blinks.



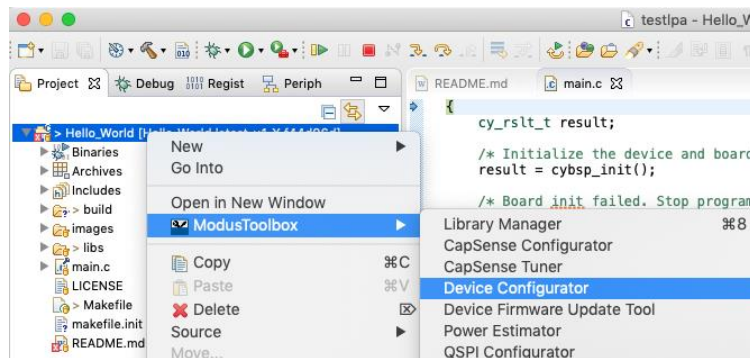
6. Launch a serial terminal (baud rate = 115200), connect to the kit, and press Enter. Observe the LED stops blinking. Press Enter again and observe the LED resumes blinking.
7. Measure power consumption of the original application on J8.

Refer to section 7.5 for details how to do that. What is the consumption you see?

Now do the following steps to improve power consumption of the application.



8. Open the Device Configurator.



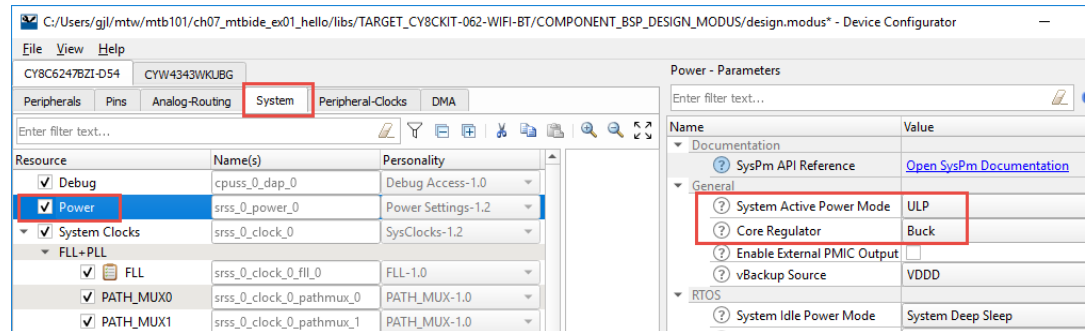
9. Select the **CY8C6247BZI-D54** > **System** tab.



10. Select the Resource **Power**

In the in the **Power – Parameters** pane, change the following parameters:

- System Active Power Mode: LP -> ULP
- Core Regulator: Normal Current LDO -> Buck

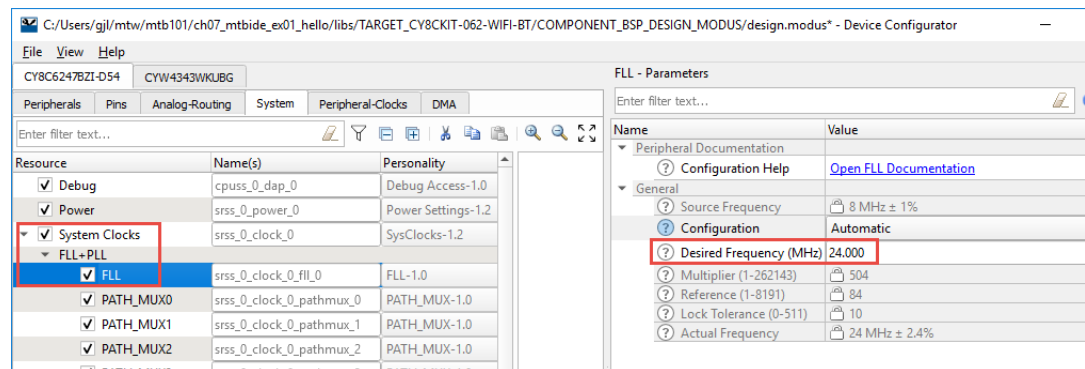


11. Select **System Clocks > FLL + PLL > FLL**

In the FLL – Parameters pane, change the parameter:

- Desired Frequency (MHz): 100 -> 24

This reduces the main system clock frequency from 100MHz to 24MHz.



12. Click **File > Save** and close the configurator.



13. Build and program the application.



14. Verify that the application works as expected.



15. Measure power consumption. What is the consumption you see?

Next, we will disable Unused Resources.



16. Open the Device Configurator.

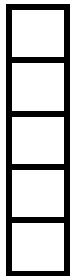


17. Navigate to **CY8C6247BZI-D54 > System** tab



18. Expand the Resource **System Clocks**, ensure the following listed clocks are selected (uncheck all others):

- FLL + PLL
 - FLL
 - PATH_MUX0
- High Frequency
 - CLK_FAST
 - CLK_HF0
 - CLK_PERI
- Input
 - ILO
 - IMO



19. Uncheck (Disable) Resource **Debug**.

20. Navigate to **CY8C6247BZI-D54 > Peripherals** tab.

21. Uncheck Resource **CSD (CapSense, etc.)**.

22. Navigate to **CY8C6247BZI-D54 > Pins** tab.

23. Uncheck all pins.

The pins reserved in the BSP are grayed out, so you won't be able to uncheck them.

Note: Click the "+" button to expand all ports.



24. Navigate to **CY8C6247BZI-D54 > Peripheral-Clocks** tab.

25. Uncheck Resource **8-bit Divider 3**.

26. Click **File > Save**.

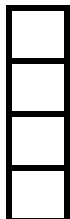
27. In the ModusToolbox IDE menu, select **Project > Properties**.

28. On the dialog, select **C/C++ Build** and change the build configuration to Release.

29. Click **Apply and Close**

30. On your disk (or directly from the Eclipse project explorer) navigate to the following:

`<project folder>\libs\TARGET_CY8CKIT-062-WIFI-BT\COMPONENT_BSP_DESIGN_MODUS\GeneratedSource\`



31. Delete `cycfg_capsense.c` and `cycfg_capsense.h` files

32. Build and program the application.

33. Verify that the application works as expected.

34. Measure power consumption. What is the consumption you see?

Now that we are done with the configuration of `design.modus`, we'll do some code changes.



35. Open the main.c file and add the following line before the main() function:

```
bool uart_command_flag = false;
```



36. Replace the contents of the infinite loop with the following code:

```
for(;;)
{
    Cy_SysPm_CpuEnterSleep(CY_SYSPM_WAIT_FOR_INTERRUPT);
    if(cyhal_uart_getc(&cy_retarget_io_uart_obj, &uart_read_value, 1) \
        == CY_RSLT_SUCCESS)
    {
        if (uart_read_value == '\r')
        {
            led_blink_active_flag ^= 1;
            uart_command_flag = true;
        }
    }

    if(timer_interrupt_flag)
    {
        timer_interrupt_flag = false;
        if (uart_command_flag)
        {
            uart_command_flag = false;
            if (led_blink_active_flag)
            {
                printf("LED blinking resumed\r\n");
            }
            else
            {
                printf("LED blinking paused \r\n");
            }
            printf("\x1b[1F");
        }
        if (led_blink_active_flag)
        {
            cyhal_gpio_toggle((cyhal_gpio_t) CYBSP_USER_LED);
        }
    }
}
```



37. Click **File > Save**.



38. Build and program the application.



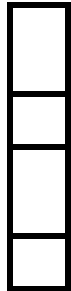
39. Verify that the application works as expected.



40. Measure power consumption. What is the consumption you see?

7.9.2 Exercise: Improving Power Consumption for the CapSense Buttons and Slider Example

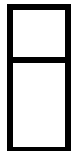
There are certainly more interesting things to do other than the Blinky, and CapSense is one of these. Let's see how to improve the power consumption for the CapSense code example.



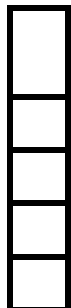
1. Create the CapSense Buttons and Slider Example for the CY8CKIT-062-WIFI-BT board using the ModusToolbox IDE. Name the application **ch07_mtbide_ex02_capsense**.
2. Build and Program the board. After programming, the application starts automatically.
3. Verify that the application works as expected by touching the slider, the LED brightness should correspond to the touch position.
4. Measure power consumption. What is the consumption you see?

Note: The current consumption slightly differs depending on whether the LED is on, off or controlled by the PWM.

Now do the following steps to improve power consumption of the application.

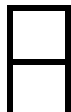


5. Open Device Configurator and select the **CY8C6247BZI-D54 > System** tab
6. Select the Resource **Power** and in the **Power – Parameters** pane change the following parameters:
 - System Active Power Mode: LP -> ULP
 - Core Regulator: Normal Current LDO -> Buck

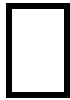


7. Select **Resource System Clocks > FLL + PLL > FLL** and change Desired Frequency (MHz): 100 -> 24
8. Click **File > Save**.
9. Build and program the application.
10. Verify that the application works as expected by touching the slider and buttons.
11. Measure power consumption. What is the consumption you see?

Now let's disable the chip resources we don't use.

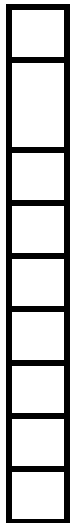


12. Open Device Configurator
13. Navigate to **CY8C6247BZI-D54 > System** tab



14. Expand Resource System Clocks, and ensure the following clocks are selected (unselect all others):

- FLL + PLL
 - FLL
 - PATH_MUX0
- High Frequency
 - CLK_FAST
 - CLK_HF0
 - CLK_PERI
- Input
 - ILO
 - IMO



15. Unselect (Disable) Resource **Debug**.

16. Navigate to **CY8C6247BZI-D54 > Pins** tab and unselect pins P6[4], P6[6] and P6[7]. These are the debug pins.

17. Click **File > Save**.

18. In the ModusToolbox IDE menu, select **Project > Properties**.

19. On the dialog, select **C/C++ Build** and change the build configuration to Release.

20. Click **Apply and Close**

21. Build and program the application.

22. Verify that the application works as expected by touching the slider and buttons.

23. Measure power consumption. What is the consumption you see?

Now let's optimize the application code.



24. Open main.c file and add the highlighted line as follows:

```
/* Start first scan */
Cy_CapSense_ScanAllWidgets(&cy_capsense_context);

for (;;)
{
    Cy_SysPm_CpuEnterSleep(CY_SYSPM_WAIT_FOR_INTERRUPT);
    if(CY_CAPSENSE_NOT_BUSY == Cy_CapSense_IsBusy(&cy_capsense_context))
    {
        /* Process all widgets */
        Cy_CapSense_ProcessAllWidgets(&cy_capsense_context);
    }
}
```



25. Click **File > Save**.

26. Build and program the application.

27. Verify that the application works as expected by touching the slider and buttons.



28. Measure power consumption. What is the consumption you see?
