Chapter 7: Low Power

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

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. Low Power Assistant consists of:

- Device Configurator: Used to configure all 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. The Application Note AN227910 - Low-Power System Design with CYW43012 and PSoC 6 MCU provides a detailed description of how to use the Low Power Assistant feature in general and specifically 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. It includes the references to the code examples which use the LPA middleware. The LPA middleware offers online documentation with the quick start guides that guide 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 you'd otherwise have started from if you didn't go through this class. Reading it after the class will help you to refresh and structure better the knowledge you have just gained.

- AN227910 - Low-Power System Design with CYW43012 and PSoC 6 MCU – link to internal doc system
- 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

7.3 Guide to this class

For this class, 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 Class

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

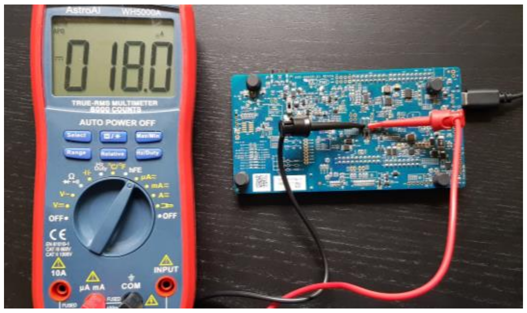
- Low Power Assistance for Bluetooth. You can go through the Bluetooth Low Power quick start guide in the Cypress Low Power Assistant Middleware Library 1.0.
- Using Power Estimator Tool. You can go through the https://github.com/cypresssemiconductorco/cype example, and use the Power Estimator tool available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

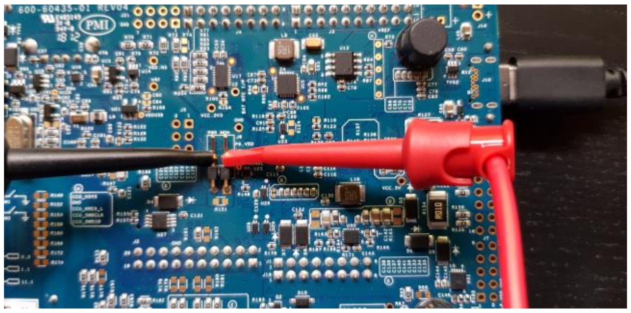
7.5 Hardware Setup and Power Consumption Measurements

For the purposes of this class, 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 the multimeter. If available, it is recommended to use a power analyzer such as KeySight N6705B. The use of the KeySight N6705B with CY8CKIT-062S2-43012 kit is explained in the AN227910 - Low-Power System Design with CYW43012 and PSoC 6 MCU.

If available, you could also use the 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 connect the probes to the J8 jumper on the bottom side of your board. You can use the KITPROG power supply, but ensure you disconnect the board from the power supply before connecting and detaching 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 KitProg. Keep everything connected throughout the exercises.

IMPORTANT: For Mbed OS exercises, you will be using the KitProg3 in DAPLink mode (Amber LED2 is blinking 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 assistance for Mbed OS

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 platforms 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, compile and flash the mbed-os-example-blinky example.

	`mbed import mbed-os-example-blinky`
	`cd mbed-os-example-blinky`
	`rm -rf mbed-os`
	`mbed add mbed-os`
	`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.

7.6.1 View the Default Device Configuration Shipped with Mbed OS

Mbed OS includes Board Support Packages (BSP) for the Cypress platforms out of the box. So, to use any of the Mbed OS examples, you should just specify the correct command line argument that should match the platform you are working on. In our case it is: `--target CY8CKIT_062_WIFI_BT`.

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

However, there is a way to override this 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 default 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.

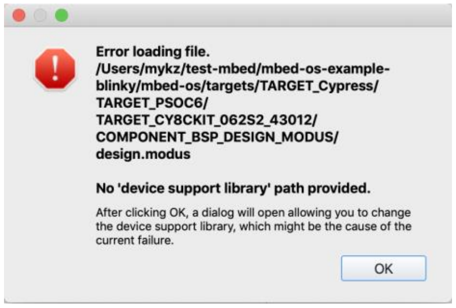
Find the design.modus file for the platform we are working on: CY8CKIT_062_WIFI_BT. What other design.* files do you see? What do you see in the GeneratedSources directory?

☐ 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

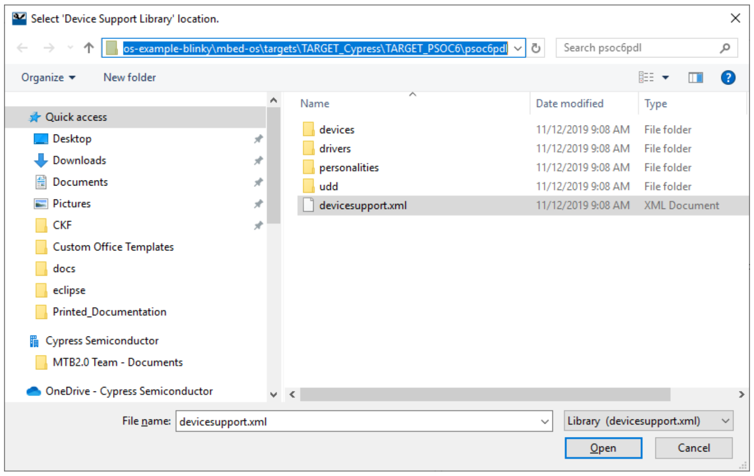


Error loading file.
/Users/mykz/test-mbed/mbed-os-example-blinky/mbed-os/targets/TARGET_Cypress/TARGET_PSOC6/TARGET_CY8CKIT_062S2_43012/COMPONENT_BSP_DESIGN_MODUS/design.modus

No 'device support library' path provided.

After clicking OK, a dialog will open allowing you to change the device support library, which might be the cause of the current failure.

☐ 4. Click **OK** and a dialog opens.



Select 'Device Support Library' location.

os-example-blinky\mbed-os\targets\TARGET_Cypress\TARGET_PSOC6\psoc6pdl

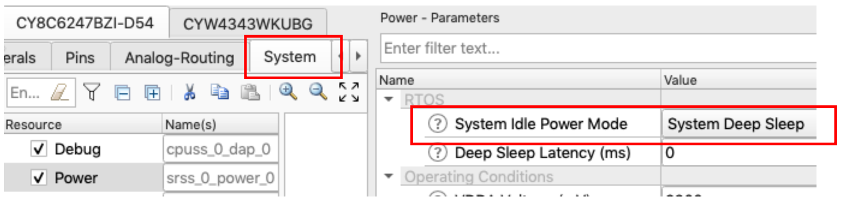
Name	Date modified	Type
devices	11/12/2019 9:08 AM	File folder
drivers	11/12/2019 9:08 AM	File folder
personalities	11/12/2019 9:08 AM	File folder
udd	11/12/2019 9:08 AM	File folder
devicesupport.xml	11/12/2019 9:08 AM	XML Document

File name: devicesupport.xml
Library (devicesupport.xml)

☐ 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.

6. In the Device Configurator, go to the **System** tab and check the setting for the **System Idle Power Mode** in the RTOS category.



Resource	Name(s)
✓ Debug	cpuss_0_dap_0
✓ Power	srss_0_power_0

Name	Value
RTOS	
System Idle Power Mode	System Deep Sleep
Deep Sleep Latency (ms)	0
Operating Conditions	

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

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

   Now you know how to visualize the default configuration shipped for our platforms in Mbed OS.

7.6.2 Modify the Default Device Configuration Shipped with Mbed OS

The examples we provide for Cypress platforms at https://github.com/cypresssemiconductorco already assume that you might want to make the changes to the device configuration. So, they allow tweaks that override that default device configuration with those 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.

1. Copy the "target" directory to the root of your application:

   From: mbed-os-example-blinky/mbed-os/targets/TARGET_Cypress/TARGET_PSOC6/TARGET_CY8CKIT_062_WIFI_BT
   To: mbed-os-example-blinky

2. Delete all files in the mbed-os-example-blinky/TARGET_CY8CKIT_062_WIFI_BT directory, **except** COMPONENT_BSP_DESIGN_MODUS and its contents.

3. Rename the following directory:

   From: COMPONENT_BSP_DESIGN_MODUS
   To: CUSTOM_BSP_DESIGN_MODUS

4. Delete the design.cyqspi and design.cycapsense files from the CUSTOM_BSP_DESIGN_MODUS directory.

   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. Add/update the mbed_app.json file in the mbed-os-example-blinky directory:

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

Note: This specifies to ignore the BSP configuration shipped with Mbed OS.

6. Replace the code in main.cpp file with the following:

```
#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);
}
```

7. Connect 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 the terminal here because we don't need it. However, for later exercises, you will need to add the option `--sterm` to the following command:

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

8. Observe power consumption. Note min max and how they're changing.

   Now we can make further changes using the Device Configurator.

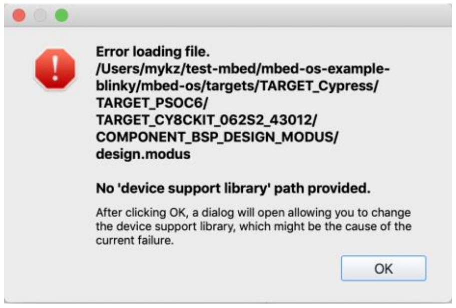
9. Start the Device Configurator located in the tools_2.0 directory of your ModusToolbox 2.0 installation.

10. Open the “custom” design file with the Device Configurator:

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

    Note: This is the design file at 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

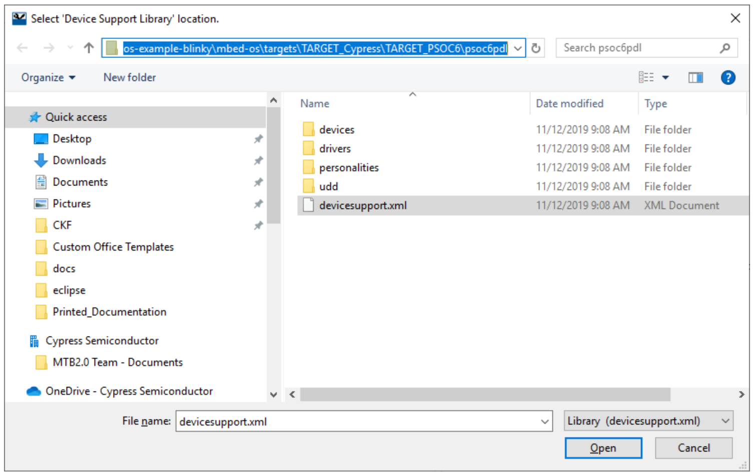


Error loading file.
/Users/mykz/test-mbed/mbed-os-example-blinky/mbed-os/targets/TARGET_Cypress/TARGET_PSOC6/TARGET_CY8CKIT_062S2_43012/COMPONENT_BSP_DESIGN_MODUS/design.modus

No 'device support library' path provided.

After clicking OK, a dialog will open allowing you to change the device support library, which might be the cause of the current failure.

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



Select 'Device Support Library' location.

os-example-blinky\mbed-os\targets\TARGET_Cypress\TARGET_PSOC6\psoc6pdl

Name	Date modified	Type
devices	11/12/2019 9:08 AM	File folder
drivers	11/12/2019 9:08 AM	File folder
personalities	11/12/2019 9:08 AM	File folder
udd	11/12/2019 9:08 AM	File folder
devicesupport.xml	11/12/2019 9:08 AM	XML Document

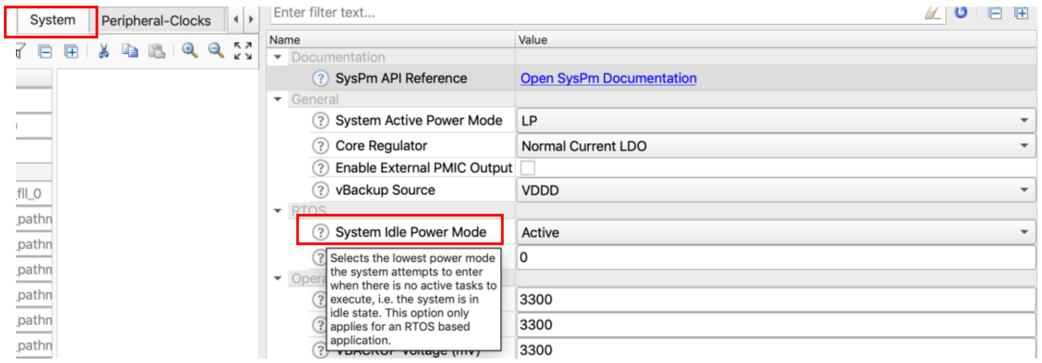
File name: devicesupport.xml

Library (devicesupport.xml)

12. 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.

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



Name	Value
Documentation	
SysPm API Reference	Open SysPm Documentation
General	
System Active Power Mode	LP
Core Regulator	Normal Current LDO
Enable External PMIC Output	☐
vBackup Source	VDDD
RTOS	
System Idle Power Mode	Active
	0
Opera	
	3300
	3300
VBACKUP Voltage (mV)	3300

Selects the lowest power mode the system attempts to enter when there is no active tasks to execute, i.e. the system is in idle state. This option only applies for an RTOS based application.

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

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

17. Observe power consumption during active and sleep.
18. Change System Idle Power Mode back to System Deep Sleep.
19. Measure power consumption again. How did it change for the sleep period?

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 platform with Mbed OS remains intact and you can use it as the reference in your further experiments.

7.7 WLAN Low Power Assistance for Mbed OS

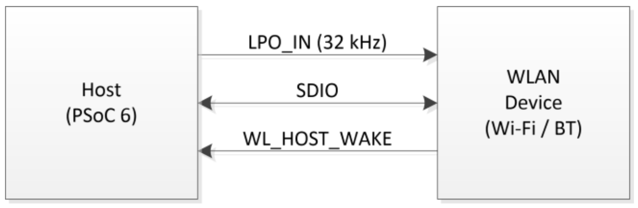
In this chapter we will learn how the Low Power Assistant helps you to minimize the MCU power consumption in system connected to the Wi-Fi. First, we will learn how to configure the HOST WAKE PIN, so that the network traffic can wake up the MCU. Then we will try to minimize the amount of time MCU needs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

7.7.1 Configure Host Wake Pin and Measure Power Consumption 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 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 addirional 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.

```
rm -rf mbed-os
mbed add mbed-os
cd mbed-os
mbed update latest
cd ..
mbed deploy
```

☐ 3. Copy the target directory to the root of your application:

From: mbed-os-example-blinky/mbed-os/targets/TARGET_Cypress/ TARGET_PSOC6/TARGET_CY8CKIT_062_WIFI_BT
To: mbed-os-example-blinky

☐ 4. Delete all files in mbed-os-example-blinky/TARGET_CY8CKIT_062_WIFI_BT directory, **except** COMPONENT_BSP_DESIGN_MODUS and its contents.

5. Rename the following directory:

   From: COMPONENT_BSP_DESIGN_MODUS
   To: CUSTOM_BSP_DESIGN_MODUS

6. Delete the design.cyqspi and design.cycapsense files from the CUSTOM_BSP_DESIGN_MODUS directory.

   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.

7. Add/update the mbed_app.json file in the mbed-os-example-blinky directory:

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

   Note: This specifies to ignore the BSP configuration shipped with Mbed OS.

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

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

9. Copy 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
```

10. Include WhdSTAInterface.h and network_activity_handler.h

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

☐ 11. Add the following declarations:

```
// 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)
```

☐ 12. 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);
}
```

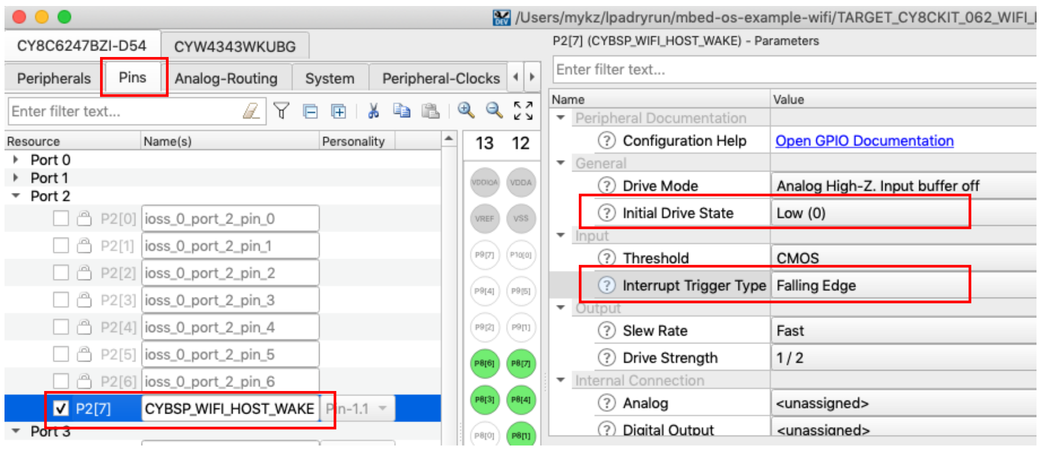
☐ 13. Open the design file with the Device Configurator (the one at the application level):

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

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

☐ 14. Switch to the **Pins** tab.

Ensure P2[7] pin specifies its name as "CYBSP_WIFI_HOST_WAKE".
In the **Parameters** pane, change **Initial Drive State** set to "Low (0) " and **Interrupt Trigger Type** to "Failing Edge".



/Users/mykz/lpadryrun/mbed-os-example-wifi/TARGET_CY8CKIT_062_WIFI_

CY8C6247BZI-D54 | CYW4343WKUBG

Peripherals | Pins | Analog-Routing | System | Peripheral-Clocks

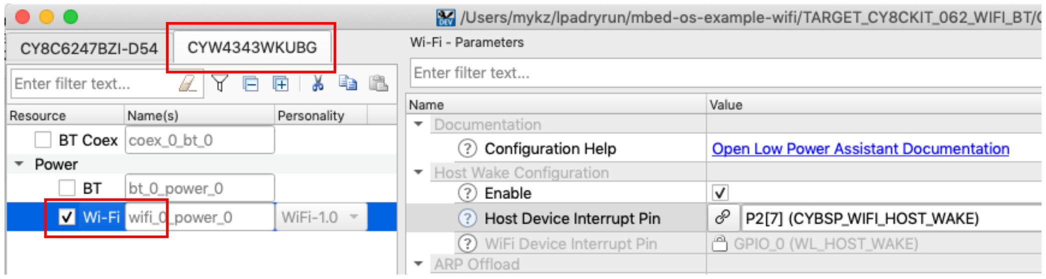
Resource	Name(s)	Personality
Port 0		
Port 1		
Port 2		
☐ P2[0]	ioss_0_port_2_pin_0	
☐ P2[1]	ioss_0_port_2_pin_1	
☐ P2[2]	ioss_0_port_2_pin_2	
☐ P2[3]	ioss_0_port_2_pin_3	
☐ P2[4]	ioss_0_port_2_pin_4	
☐ P2[5]	ioss_0_port_2_pin_5	
☐ P2[6]	ioss_0_port_2_pin_6	
✓ P2[7]	CYBSP_WIFI_HOST_WAKE	Pin-1.1
Port 3		

P2[7] (CYBSP_WIFI_HOST_WAKE) - Parameters

Name	Value
Peripheral Documentation	
Configuration Help	Open GPIO Documentation
General	
Drive Mode	Analog High-Z. Input buffer off
Initial Drive State	Low (0)
Input	
Threshold	CMOS
Interrupt Trigger Type	Falling Edge
Output	
Slew Rate	Fast
Drive Strength	1 / 2
Internal Connection	
Analog	<unassigned>
Digital Output	<unassigned>

☐ 15. 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 "CYBSP_WIFI_HOST_WAKE".



/Users/mykz/lpadryrun/mbed-os-example-wifi/TARGET_CY8CKIT_062_WIFI_BT/

CY8C6247BZI-D54 | CYW4343WKUBG

Resource	Name(s)	Personality
☐ BT Coex	coex_0_bt_0	
Power		
☐ BT	bt_0_power_0	
✓ Wi-Fi	wifi_0_power_0	WiFi-1.0

Wi-Fi - Parameters

Name	Value
Documentation	
Configuration Help	Open Low Power Assistant Documentation
Host Wake Configuration	
Enable	✓
Host Device Interrupt Pin	P2[7] (CYBSP_WIFI_HOST_WAKE)
WiFi Device Interrupt Pin	GPIO_0 (WL_HOST_WAKE)
ARP Offload	

☐ 16. Save the configuration to generate the necessary code.

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

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

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

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

In this exercise we will learn how to send the PING and ARP packets.

☐ 1. Install the arp-ping tool for your system.

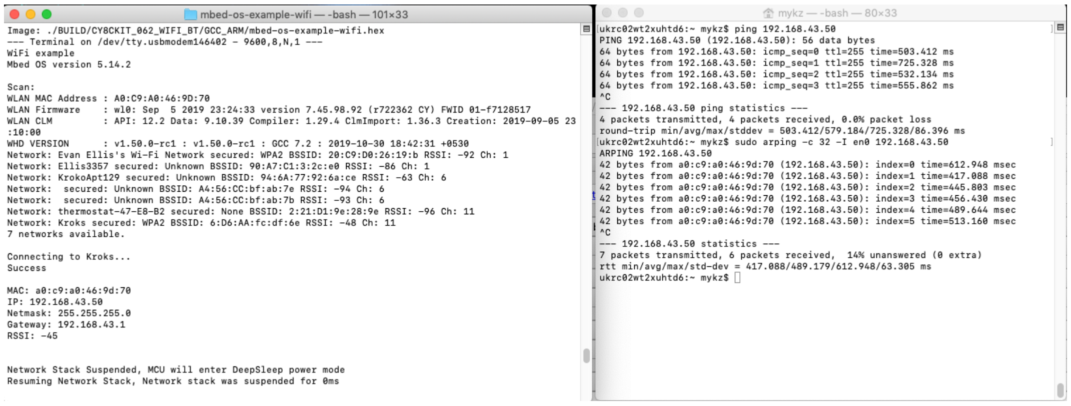
- Windows: https://www.elifulkerson.com/projects/arp-ping.php
- Mac : http://macappstore.org/arping/
- linux : sudo apt install arping

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

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

☐ 3. Open two terminal windows, side by side. On the left one you'll observe how device responds to the packets sent from the left terminal.

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



```
Image: ./BUILD/CY8CKIT_062_WIFI_BT/GCC_ARM/mbed-os-example-wifi.hex
--- Terminal on /dev/tty.usbmodem146402 - 9600,8,N,1 ---
WiFi example
Mbed OS version 5.14.2

Scan:
WLAN MAC Address : A0:C9:A0:46:9D:70
WLAN Firmware    : wl0: Sep  5 2019 23:24:33 version 7.45.98.92 (r722362 CY) FWID 01-f7128517
WLAN CLM         : API: 12.2 Data: 9.10.39 Compiler: 1.29.4 ClmImport: 1.36.3 Creation: 2019-09-05 23
:10:00
WHD VERSION      : v1.50.0-rc1 : v1.50.0-rc1 : GCC 7.2 : 2019-10-30 18:42:31 +0530
Network: Evan Ellis's Wi-Fi Network secured: WPA2 BSSID: 20:C9:D0:26:19:b RSSI: -92 Ch: 1
Network: Ellis3357 secured: Unknown BSSID: 90:A7:C1:3:2c:e0 RSSI: -86 Ch: 1
Network: KrokoApt129 secured: Unknown BSSID: 94:6A:77:92:6a:ce RSSI: -63 Ch: 6
Network:  secured: Unknown BSSID: A4:56:CC:bf:ab:7e RSSI: -94 Ch: 6
Network:  secured: Unknown BSSID: A4:56:CC:bf:ab:7b RSSI: -93 Ch: 6
Network: thermostat-47-E8-B2 secured: None BSSID: 2:21:D1:9e:28:9e RSSI: -96 Ch: 11
Network: Kroks secured: WPA2 BSSID: 6:D6:AA:fc:df:6e RSSI: -48 Ch: 11
7 networks available.

Connecting to Kroks...
Success

MAC: a0:c9:a0:46:9d:70
IP: 192.168.43.50
Netmask: 255.255.255.0
Gateway: 192.168.43.1
RSSI: -45


Network Stack Suspended, MCU will enter DeepSleep power mode
Resuming Network Stack, Network stack was suspended for 0ms
```

```
ukrc02wt2xuhtd6:~ mykz$ ping 192.168.43.50
PING 192.168.43.50 (192.168.43.50): 56 data bytes
64 bytes from 192.168.43.50: icmp_seq=0 ttl=255 time=503.412 ms
64 bytes from 192.168.43.50: icmp_seq=1 ttl=255 time=725.328 ms
64 bytes from 192.168.43.50: icmp_seq=2 ttl=255 time=532.134 ms
64 bytes from 192.168.43.50: icmp_seq=3 ttl=255 time=555.862 ms
^C
--- 192.168.43.50 ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 503.412/579.184/725.328/86.396 ms
ukrc02wt2xuhtd6:~ mykz$ sudo arping -c 32 -I en0 192.168.43.50
ARPING 192.168.43.50
42 bytes from a0:c9:a0:46:9d:70 (192.168.43.50): index=0 time=612.948 msec
42 bytes from a0:c9:a0:46:9d:70 (192.168.43.50): index=1 time=417.088 msec
42 bytes from a0:c9:a0:46:9d:70 (192.168.43.50): index=2 time=445.803 msec
42 bytes from a0:c9:a0:46:9d:70 (192.168.43.50): index=3 time=456.430 msec
42 bytes from a0:c9:a0:46:9d:70 (192.168.43.50): index=4 time=489.644 msec
42 bytes from a0:c9:a0:46:9d:70 (192.168.43.50): index=5 time=513.160 msec
^C
--- 192.168.43.50 statistics ---
7 packets transmitted, 6 packets received,  14% unanswered (0 extra)
rtt min/avg/max/std-dev = 417.088/489.179/612.948/63.305 ms
ukrc02wt2xuhtd6:~ mykz$
```

☐ 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, for Mac (use the IP of your device, send 15 times for example)

```
sudo arping -c 32 -I en0 192.168.43.50
```

☐ 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 Configuring Packet Filter Offload

Packet filters allow the host processor to limit which types of packets get passed up to the host processor 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 awaken the host (if it is asleep) so it can retrieve the packet for processing. Often times 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 interrupted.

Refer to the Packet Filters Quick Start Guide for additional information.

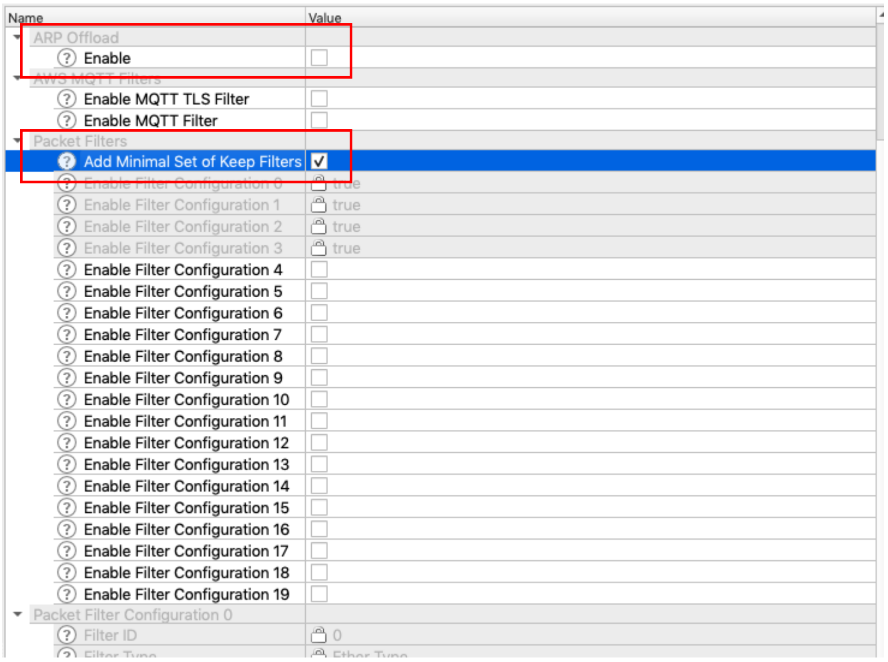
☐

1. Open the design file with the Device Configurator:

   mbed-os-example-wifi/TARGET_CY8CKIT_062_WIFI_BT/
   CUSTOM_BSP_DESIGN_MODUS/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, and are needed to connect to Wi-Fi Access point. Any other Wi-Fi packets are dropped by the WLAN chip and not forwarded to the Host MCU (PSoC 6).



Name	Value
ARP Offload	
Enable	☐
AWS MQTT Filters	
Enable MQTT TLS Filter	☐
Enable MQTT Filter	☐
Packet Filters	
Add Minimal Set of Keep Filters	✓
Enable Filter Configuration 0	true
Enable Filter Configuration 1	true
Enable Filter Configuration 2	true
Enable Filter Configuration 3	true
Enable Filter Configuration 4	☐
Enable Filter Configuration 5	☐
Enable Filter Configuration 6	☐
Enable Filter Configuration 7	☐
Enable Filter Configuration 8	☐
Enable Filter Configuration 9	☐
Enable Filter Configuration 10	☐
Enable Filter Configuration 11	☐
Enable Filter Configuration 12	☐
Enable Filter Configuration 13	☐
Enable Filter Configuration 14	☐
Enable Filter Configuration 15	☐
Enable Filter Configuration 16	☐
Enable Filter Configuration 17	☐
Enable Filter Configuration 18	☐
Enable Filter Configuration 19	☐
Packet Filter Configuration 0	
Filter ID	0
Filter Type	Ether Type

3. Save the configuration to generate the necessary code.
4. Build the project and program the board, also start terminal:

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

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

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

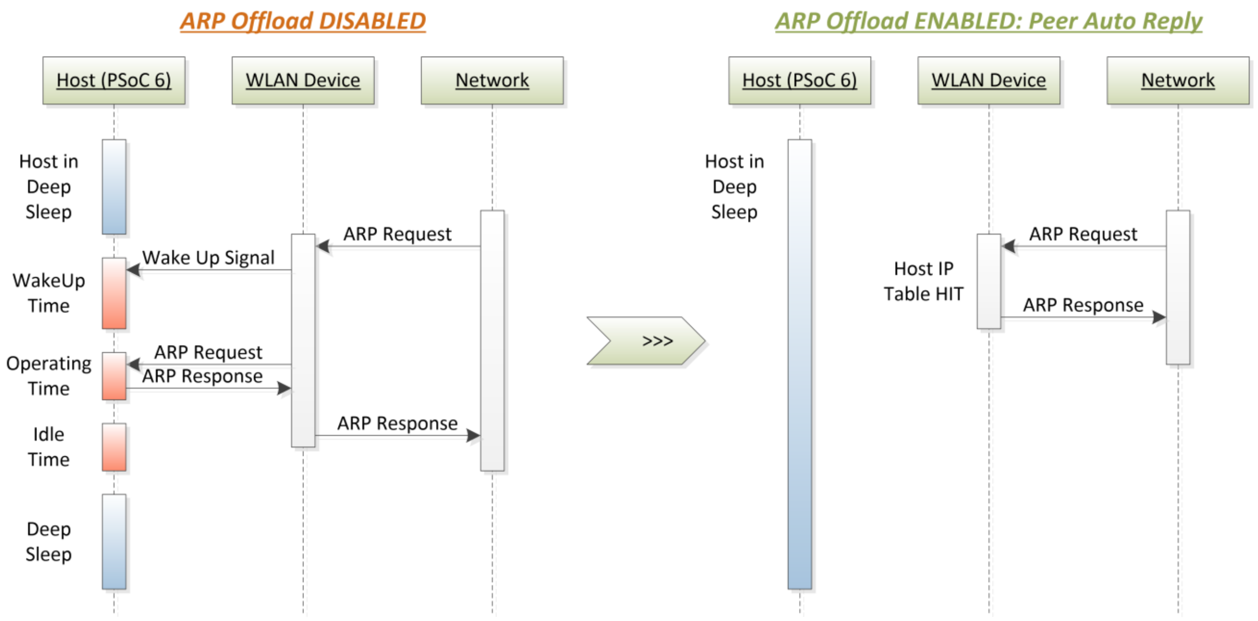
6. Send an "arping" command and observe that:
   - You get the responses
   - PSoC 6 MCU wakes up (observe power consumption).

So 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 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) lookup. 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.

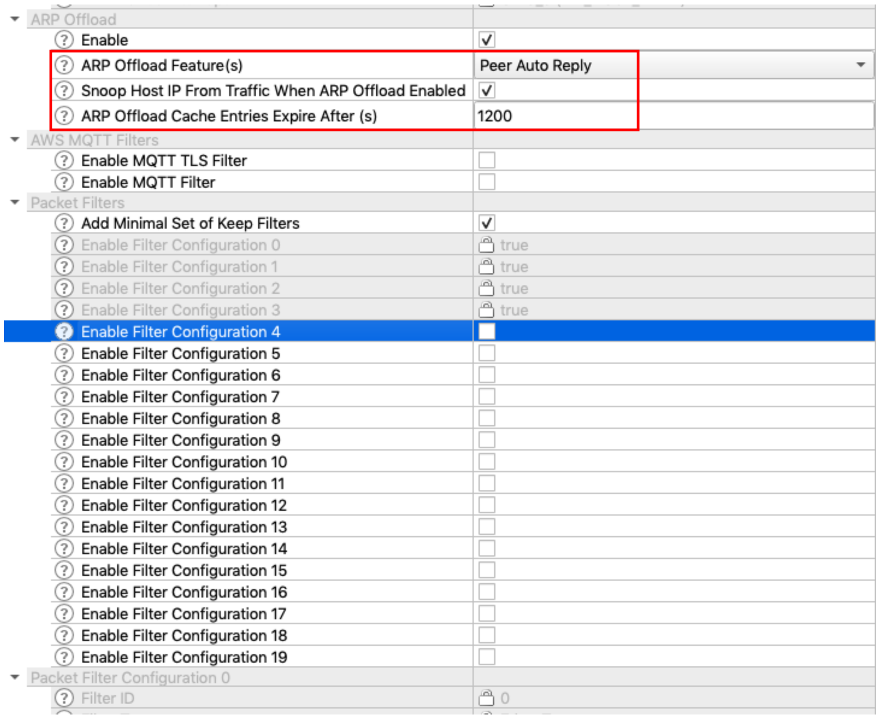


☐ 1. Open the design file with the Device Configurator.

mbed-os-example-wifi/TARGET_CY8CKIT_062_WIFI_BT/
CUSTOM_BSP_DESIGN_MODUS/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	✓
ARP Offload Feature(s)	Peer Auto Reply
Snoop Host IP From Traffic When ARP Offload Enabled	✓
ARP Offload Cache Entries Expire After (s)	1200
AWS MQTT Filters	
Enable MQTT TLS Filter	☐
Enable MQTT Filter	☐
Packet Filters	
Add Minimal Set of Keep Filters	✓
Enable Filter Configuration 0	true
Enable Filter Configuration 1	true
Enable Filter Configuration 2	true
Enable Filter Configuration 3	true
Enable Filter Configuration 4	☐
Enable Filter Configuration 5	☐
Enable Filter Configuration 6	☐
Enable Filter Configuration 7	☐
Enable Filter Configuration 8	☐
Enable Filter Configuration 9	☐
Enable Filter Configuration 10	☐
Enable Filter Configuration 11	☐
Enable Filter Configuration 12	☐
Enable Filter Configuration 13	☐
Enable Filter Configuration 14	☐
Enable Filter Configuration 15	☐
Enable Filter Configuration 16	☐
Enable Filter Configuration 17	☐
Enable Filter Configuration 18	☐
Enable Filter Configuration 19	☐
Packet Filter Configuration 0	
Filter ID	0

3. Save the configuration to generate the necessary code.

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

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

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

   In serial terminal that it does not wake up the PSoC 6 device since there is no "keep" packet filter for ICMP pings, there is no response for the pings. PSoC 6 MCU remains in Deep Sleep mode (measure power consumption).

6. Send an "arping" command and observe that:
   - You still get the responses
   - PSoC 6 MCU no longer wakes up (measure power consumption).

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

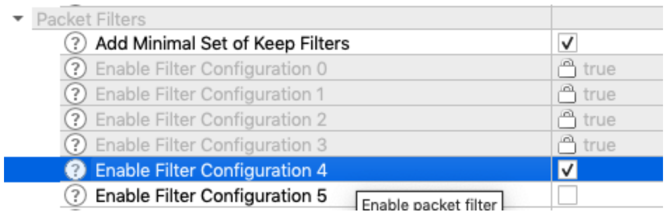
7.7.5 Let the ICMP (Ping) Go Through

☐ 1. Open the design file with the Device Configurator.

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

☐ 2. Switch to the connectivity device tab.

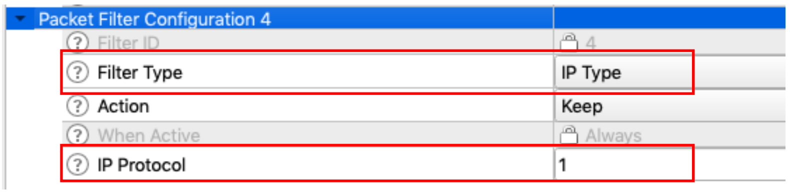
Enable new filter (Set check box at **Enable Filter Configuration 4**.)



Packet Filters	
Add Minimal Set of Keep Filters	✓
Enable Filter Configuration 0	true
Enable Filter Configuration 1	true
Enable Filter Configuration 2	true
Enable Filter Configuration 3	true
Enable Filter Configuration 4	✓
Enable Filter Configuration 5	

☐ 3. Scroll down and set parameters as follows:

Filter Type set to “IP Type”
IP Protocol set to “1’



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 -DMBED_TICKLESS --target CY8CKIT_062_WIFI_BT --toolchain
GCC_ARM --flash --sterm
```

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

It now wakes the Host up because you have just added "keep" packet filter for ICMP pings. You can see the response in 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 Assistance in Mbed OS. Examples include more explanation of the features you now know how to use. They implement the 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 WIFI credentials.

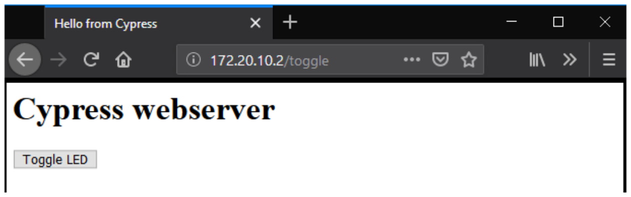
After programming the example, jump to the "Verify" section available.

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 extended wait state.

Figure 2. HTTP Server Webpage

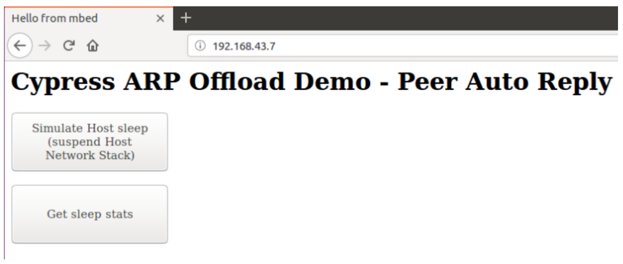


Cypress webserver

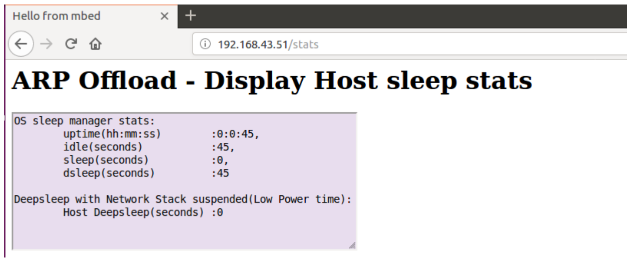
7.8.2 ARP (Address Resolution Protocol) Offload

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.



Cypress ARP Offload Demo - Peer Auto Reply



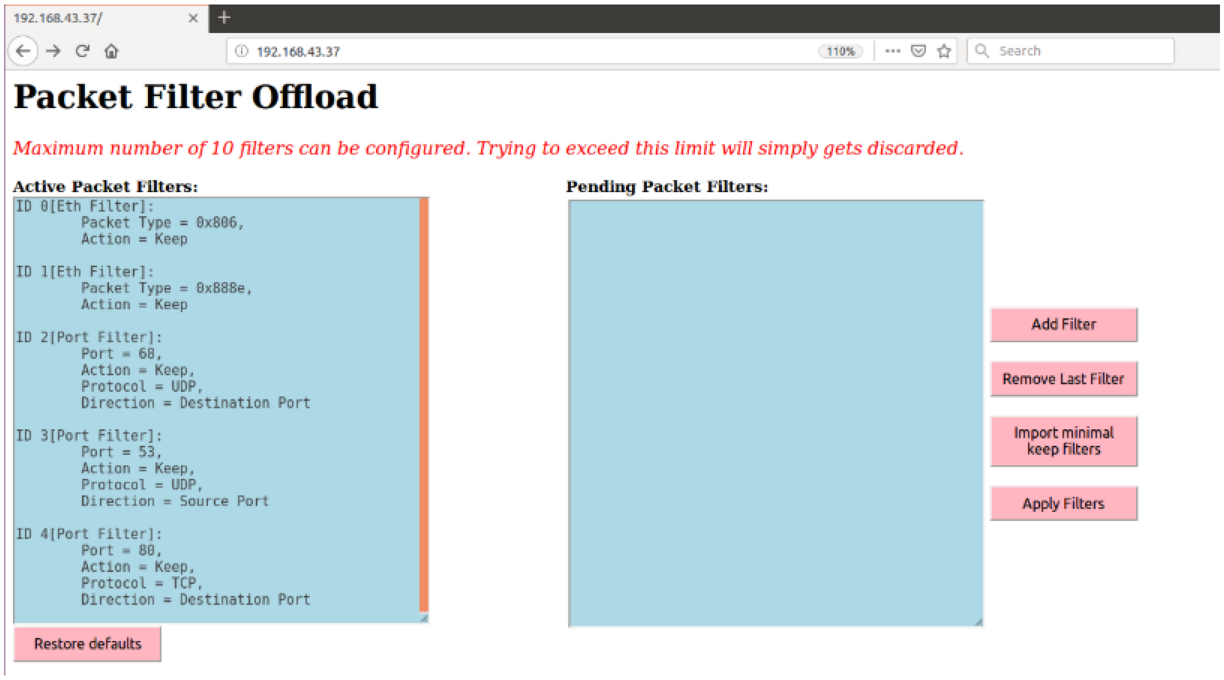
ARP Offload - Display Host sleep stats

```
OS sleep manager stats:
        uptime(hh:mm:ss)        :0:0:45,
        idle(seconds)           :45,
        sleep(seconds)          :0,
        dsleep(seconds)         :45

Deepsleep with Network Stack suspended(Low Power time):
        Host Deepsleep(seconds) :0
```

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 a "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.



Packet Filter Offload

Maximum number of 10 filters can be configured. Trying to exceed this limit will simply gets discarded.

Active Packet Filters:

```
ID 0[Eth Filter]:
        Packet Type = 0x806,
        Action = Keep

ID 1[Eth Filter]:
        Packet Type = 0x888e,
        Action = Keep

ID 2[Port Filter]:
        Port = 68,
        Action = Keep,
        Protocol = UDP,
        Direction = Destination Port

ID 3[Port Filter]:
        Port = 53,
        Action = Keep,
        Protocol = UDP,
        Direction = Source Port

ID 4[Port Filter]:
        Port = 80,
        Action = Keep,
        Protocol = TCP,
        Direction = Destination Port
```

Pending Packet Filters:

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.

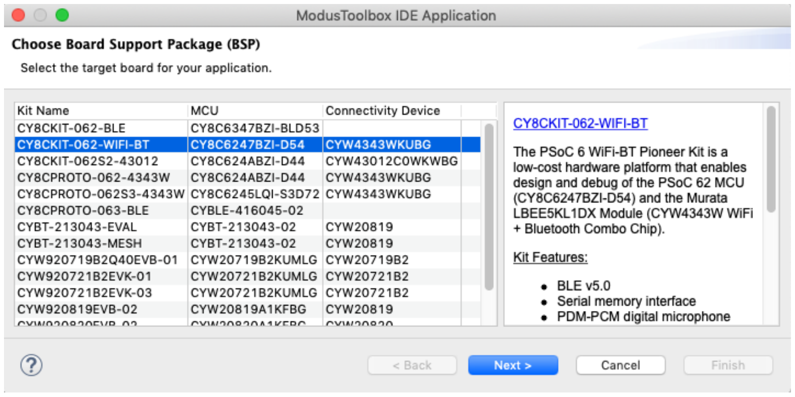
7.9 Low Power Assistance for PSoC 6 SDK

So far, we've been exercising the Low Power Assistance functionality with the Mbed OS projects. This section will teach you how to improve the power consumption in the bare metal projects, using the ModusToolbox IDE. Keep in mind that with ModusToolbox we provide the set of tools that enable you to work in various kinds of environments, IDEs, and RTOS ecosystems. The Low Power Assistant is not an exception.

In this part of the training we will select two ModusToolbox code examples and will try to improve the power consumption to the extent possible. Our goal will be to preserve the original functionality as much as possible. We first start from the blinky example, and then will improve the power consumption for the CapSense code example available in ModusToolbox. We will use bare metal (non RTOS versions of the code example). We will use the same CY8CKIT-062-WIFI-BT board we have used in previous exercises.

7.9.1 Making Hello World Energy Efficient

1. Start ModusToolbox IDE
2. Create new application: File > New > ModusToolbox IDE Application
3. Select the CY8CKIT-062-WIFI-BT target board



Choose Board Support Package (BSP)

Select the target board for your application.

Kit Name	MCU	Connectivity Device
CY8CKIT-062-BLE	CY8C6347BZI-BLD53	
CY8CKIT-062-WIFI-BT	CY8C6247BZI-D54	CYW4343WKUBG
CY8CKIT-062S2-43012	CY8C624ABZI-D44	CYW43012C0WKWBG
CY8CPROTO-062-4343W	CY8C624ABZI-D44	CYW4343WKUBG
CY8CPROTO-062S3-4343W	CY8C6245LQI-S3D72	CYW4343WKUBG
CY8CPROTO-063-BLE	CYBLE-416045-02	
CYBT-213043-EVAL	CYBT-213043-02	CYW20819
CYBT-213043-MESH	CYBT-213043-02	CYW20819
CYW920719B2Q40EVB-01	CYW20719B2KUMLG	CYW20719B2
CYW920721B2EVK-01	CYW20721B2KUMLG	CYW20721B2
CYW920721B2EVK-03	CYW20721B2KUMLG	CYW20721B2
CYW920819EVB-02	CYW20819A1KFBG	CYW20819
CYW920820EVB-02	CYW20820A1KFBG	CYW20820

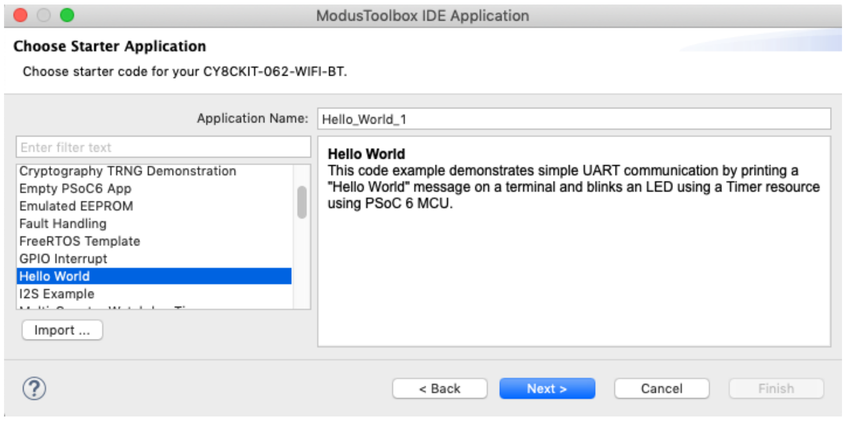
CY8CKIT-062-WIFI-BT

The PSoC 6 WiFi-BT Pioneer Kit is a low-cost hardware platform that enables design and debug of the PSoC 62 MCU (CY8C6247BZI-D54) and the Murata LBEE5KL1DX Module (CYW4343W WiFi + Bluetooth Combo Chip).

Kit Features:

- BLE v5.0
- Serial memory interface
- PDM-PCM digital microphone

☐ 4. Then select the **Hello World** starter application and click **Next** and the **Finish** buttons


ModusToolbox IDE Application

Choose Starter Application

Choose starter code for your CY8CKIT-062-WIFI-BT.

Application Name: Hello_World_1

Hello World
This code example demonstrates simple UART communication by printing a "Hello World" message on a terminal and blinks an LED using a Timer resource using PSoC 6 MCU.

☐ 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.

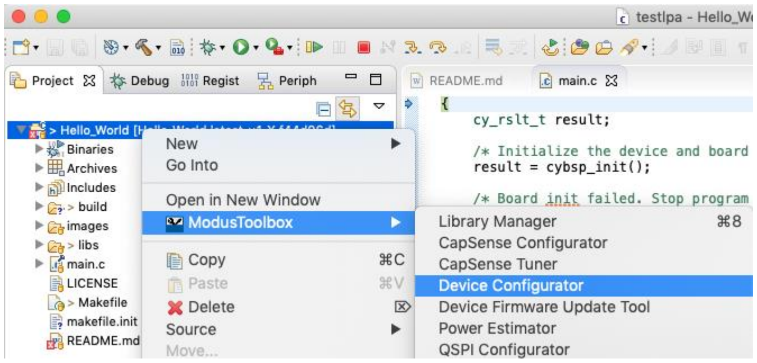
☐ 6. Launch a terminal and press Enter. Observe the LED stops blinking. Press Enter again and observe LED resumed blinking.

☐ 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.

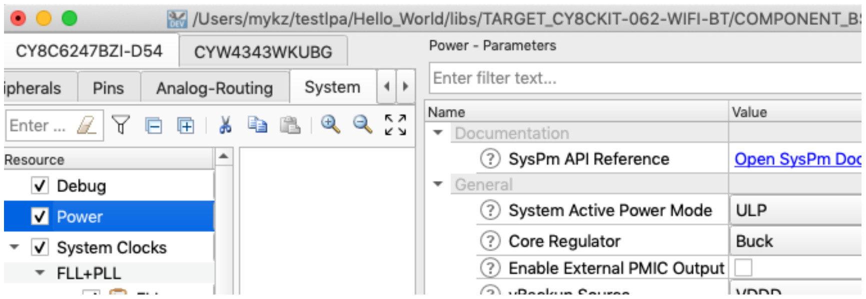

```
    {
        cy_rslt_t result;

        /* Initialize the device and board
        result = cybsp_init();

        /* Board init failed. Stop program
```

☐ 9. Select to **CY8C6247BZI-D54 > System** tab

☐ 10. Select Resource **Power** and

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

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



/Users/mykz/testlpa/Hello_World/libs/TARGET_CY8CKIT-062-WIFI-BT/COMPONENT_B

CY8C6247BZI-D54 | CYW4343WKUBG

ipherals | Pins | Analog-Routing | System

Resource: ✓ Debug, ✓ Power, ✓ System Clocks, FLL+PLL

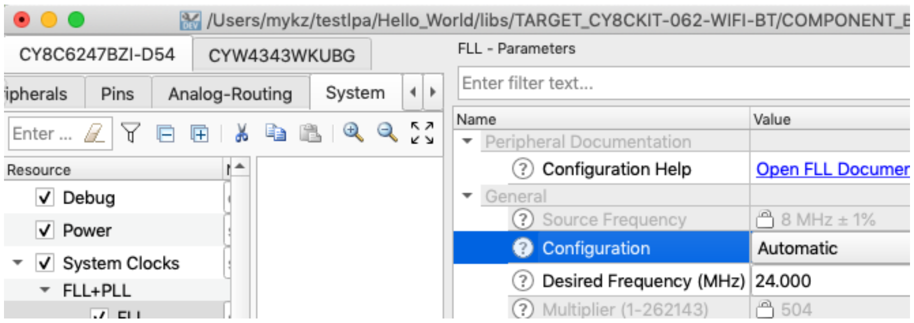
Power - Parameters

Name	Value
Documentation	
SysPm API Reference	Open SysPm Doc
General	
System Active Power Mode	ULP
Core Regulator	Buck
Enable External PMIC Output	☐

☐ 11. Select Resource System Clocks > FLL + PLL > FLL

In the FLL – Parameters pane, change the parameter:

- Desired Frequency (MHz): 100 -> 24



/Users/mykz/testlpa/Hello_World/libs/TARGET_CY8CKIT-062-WIFI-BT/COMPONENT_E

CY8C6247BZI-D54 | CYW4343WKUBG

ipherals | Pins | Analog-Routing | System

Resource: ✓ Debug, ✓ Power, ✓ System Clocks, FLL+PLL

FLL - Parameters

Name	Value
Peripheral Documentation	
Configuration Help	Open FLL Documer
General	
Source Frequency	8 MHz ± 1%
Configuration	Automatic
Desired Frequency (MHz)	24.000
Multiplier (1-262143)	504

☐ 12. Click **File > Save**.

☐ 13. Build and program the application.

☐ 14. Verify that the application works as expected.

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

Now disable Unused Resources

☐ 16. Open the Device Configurator.

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

18. Expand Resource System Clocks, ensure the following listed clocks are selected (uncheck all others):
    - FLL
    - PATH_MUX0
    - CLK_FAST
    - CLK_HF0
    - CLK_PERI
    - 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, so you won't be able to uncheck them.
24. Navigate to **CY8C6247BZI-D54 > Peripheral-Clocks** tab.
25. Uncheck Resource **8 bit Divider 3.**
26. Click **File > Save**.
27. On your disk (or directly from Eclipse project explorer) navigate to the following:

    …\Hello_World\libs\TARGET_CY8CKIT-062-WIFI-BT\COMPONENT_BSP_DESIGN_MODUS\GeneratedSource\
28. Delete cycfg_capsense.c and cycfg_capsense.h files
29. Build and program the application.
30. Verify that the application works as expected.
31. 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.

32. In the ModusToolbox IDE menu, select **Project > Properties**.
33. On the dialog, select **C/C++ Build** and change the build configuration to Release.
34. Click **Apply and Close**
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 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 Slide Example for the CY8CKIT-062-WIFI-BT board using the ModusToolbox IDE.
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 the LED status: ON / OFF / 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 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
    - PATH_MUX0
    - CLK_FAST
    - CLK_HF0
    - CLK_PERI
    - ILO
    - IMO
15. Unselect (Disable) Resource **Debug.**
16. Navigate to **CY8C6247BZI-D54 > Pins** tab and unselect P6[4], P6[6] and P6[7] pins.
17. Click **File > Save**.
18. Build and program the application.
19. Verify that the application works as expected by touching the slider and buttons.
20. Measure power consumption. What is the consumption you see?

Change the application.

21. In the ModusToolbox IDE menu, select **Project > Properties**.
22. On the dialog, select **C/C++ Build** and change the build configuration to Release.
23. Click **Apply and Close**
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?