Sleep Enable/Disable

# Introduction

This application note describes the basics of sleep management in InnoOS™, providing a brief on using os\_suspend\_enable() and os\_suspend\_disable() functions.

# System Sleep Enable & Disable

## System Sleep APIs

### os\_suspend\_enable()

Suspends the system when idle.

|  |
| --- |
| void os\_suspend\_enable(void) |

Calling os\_suspend\_enable() will suspend the system or enable deep sleep when the processor is idle. Enabling and disabling suspend mode takes additional time, which will affect the real-time response of the system. When an interrupt occurs, the system will wake up even if it is in a suspended state. However, the latency will be more as compared to when the system operates in a non-suspended mode.

### os\_suspend\_disable()

Disables system suspend.

|  |
| --- |
| void os\_suspend\_disable(void) |

When the system is idle, the kernel will place the CPU in low-power mode, ready to swiftly resume execution if an interrupt occurs.

### os\_avail\_heap()

Returns size of the remaining heap.

|  |
| --- |
| size\_t os\_avail\_heap(void) |

This is the total size including internal overhead. However, this may not represent the amount that can be allocated by an application.

# Code Walkthrough

## Sleep\_enable\_disable.c

### Overview

The sample code is the path: examples/sleep\_enable\_disable/src/sleep\_enable\_disable.c is a simple application which demonstrates sleep mode.

### Sample Code Walkthrough

|  |
| --- |
| int suspend\_count =0; |

To connect to a Wi-Fi network, the following APIs from the Wi-Fi Connection Manager are used:

1. wcm\_create()

This function creates the Wi-Fi network interface using the wcm\_handle pointer.

1. wcm\_notify\_enable()

Enables callbacks of the link and IP address changes.

1. wcm\_add\_network\_profile ()

Asynchronously adds a Wi-Fi network to connect. Currently only one network can be added.

1. wcm\_auto\_connect ()

Enables start or stop auto connection of the device with Wi-Fi.

|  |
| --- |
| my\_wcm\_handle = wcm\_create(NULL);  if(my\_wcm\_handle != NULL)  {  wcm\_notify\_enable(my\_wcm\_handle, my\_wcm\_notify\_cb, NULL);  if (np\_conf\_path != NULL) {  /\* Create a Network Profile from a configuration file in  \*the file system\*/  rval = network\_profile\_new\_from\_file\_system(&profile, np\_conf\_path);  } else {  /\* Create a Network Profile using BOOT ARGS\*/  rval = network\_profile\_new\_from\_boot\_args(&profile);  }  if (rval < 0) {  pr\_err("could not create network profile %d\n", rval);\  return 0;  }  rval = wcm\_add\_network\_profile(my\_wcm\_handle, profile);  if (rval < 0) {  pr\_err("could not associate network profile to wcm %d\n", rval);  return 0;  }  wcm\_cnt = wcm\_auto\_connect(my\_wcm\_handle, 1);  os\_sem\_wait(&connect\_lock);  os\_sleep\_us(1000000, OS\_TIMEOUT\_NO\_WAKEUP);  if (wcm\_cnt == WCM\_SUCCESS) {  auot\_sucss = 1; } |

|  |
| --- |
| static void my\_wcm\_notify\_cb(void \*ctx, struct os\_msg \*msg)  {  switch(msg->msg\_type)  {  case(WCM\_NOTIFY\_MSG\_LINK\_UP):  os\_printf("wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP\n");  break;  case(WCM\_NOTIFY\_MSG\_LINK\_DOWN):  os\_printf("wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_DOWN\n");  break;  case(WCM\_NOTIFY\_MSG\_ADDRESS):  os\_printf("wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS\n");  if (os\_sem\_waiting(&connect\_lock)){  os\_sem\_post(&connect\_lock);  }  break;  case(WCM\_NOTIFY\_MSG\_DISCONNECT\_DONE):  os\_printf("wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_DISCONNECT\_DONE\n");  if (os\_sem\_waiting(&connect\_lock)){  os\_sem\_post(&connect\_lock);  }  break;  default:  break;  }  os\_msg\_release(msg);  } |

suspend\_count is used to count the number of times the sleep enable/disable functionality is called for in the program. The values are printed in the console.

**Case 1**:

On successfully connecting to the Wi-Fi network, enable sleep for a duration of 5 seconds and then disable it:

|  |
| --- |
| /\*\*Enable sleep for 5 sec then disable\*\*/  os\_printf("\nCase 1: Enable then disable\n");  os\_printf("os\_suspend enable 1x\n");  enable\_sleep();  os\_printf("suspend\_count: %d\n", suspend\_count);  os\_sleep\_us(5000000, OS\_TIMEOUT\_NO\_WAKEUP);  os\_printf("os\_suspend disable 1x\n");  disable\_sleep();  os\_printf("suspend\_count: %d\n", suspend\_count);  os\_sleep\_us(5000000, OS\_TIMEOUT\_NO\_WAKEUP); |

**Case 2**:

Sleep enable function is called twice to put the device to sleep and disabled after 5 seconds. Though the sleep mode is disabled, os\_suspend\_disable() function is called once the device continues to stay in sleep mode for the next five seconds. On calling the disable sleep function again, the system resets to default state and sleep is disabled.

|  |
| --- |
| /\*\*  Calls os\_suspend\_enable twice then os\_suspend\_disable.  T2 will not get out of low-power mode on first os\_suspend\_disable call  \*\*/  os\_printf("\nCase 2: Enable 2x then disable 1x\n");  os\_printf("os\_suspend enable 2x\n");  enable\_sleep();  enable\_sleep();  os\_printf("suspend\_count: %d\n", suspend\_count);  os\_sleep\_us(5000000, OS\_TIMEOUT\_NO\_WAKEUP);  /\*low-power mode will not be disabled though we call  os\_suspened\_disable()\*/  os\_printf("os\_suspend disable 1x\n");  disable\_sleep();  os\_printf("suspend\_count: %d\n", suspend\_count);  os\_sleep\_us(5000000, OS\_TIMEOUT\_NO\_WAKEUP);  /\* reset to default state. Sleep will be disabled \*/  os\_printf("os\_suspend disable 1x\n");  disable\_sleep();  os\_printf("suspend\_count: %d\n", suspend\_count);  os\_sleep\_us(5000000, OS\_TIMEOUT\_NO\_WAKEUP); |

**Case 3**:

Call the os\_suspend\_disable() first. In this case, the low power mode will not be enabled as the disable function is called before os\_suspend\_enable().

|  |
| --- |
| /\*\*  Calls os\_suspend\_disable dirst os\_suspend\_enable.  low-power mode not be enabled because os\_suspend\_disable  was called before a os\_suspend\_enable  \*\*/  os\_printf("\nCase 3: Disable then enable\n");  os\_printf("os\_suspend disable 1x\n");  disable\_sleep();  os\_printf("suspend\_count: %d\n", suspend\_count);  os\_sleep\_us(5000000, OS\_TIMEOUT\_NO\_WAKEUP);  /\*T2 will not go into low-power mode because we called  os\_suspend\_disable before os\_suspend\_enable  \*/  os\_printf("os\_suspend enable 1x\n");  enable\_sleep();  os\_printf("suspend\_count: %d\n", suspend\_count);  os\_sleep\_us(5000000, OS\_TIMEOUT\_NO\_WAKEUP); |

### Running the Application

Program sleep\_enable\_disable.elf (sdk\_x.y\examples\sleep\_enable\_disable\bin) using the Download tool:

Launch the Download tool provided with InnoPhase Talaria TWO SDK.

In the GUI window:

1. Boot Target: Select the appropriate EVK from the drop-down
2. ELF Input: Load the sleep\_enable\_disable.elf by clicking on Select ELF File.
3. AP Options: Provide the SSID and Passphrase under AP Options to connect to an Access Point.
4. Programming: Prog RAM or Prog Flash as per requirement.

For more details on using the Download tool, refer to the document: UG\_Download\_Tool.pdf (path: *sdk\_x.y/pc\_tools/Download\_Tool/doc*).

**Note**: x and y refer to the SDK release version. For example: sdk\_2.4/doc.

### Expected Output

sleep\_enable\_disable.elf is created when compiling the code which provides the following console output when programmed to Talaria TWO.

|  |
| --- |
| UART:SNWWWWWAEBuild $Id: git-ba65998b7 $  np\_conf\_path=/data/nprofile.json ssid=InnoPhase passphrase=43083191  $App:git-73e7f910  SDK Ver: sdk\_2.5  Sleep Enable Disable App  addr e0:69:3a:00:13:90  [2.099,560] CONNECT:00:5f:67:cd:c5:a6 Channel:11 rssi:-46 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  [3.009,283] MYIP 192.168.0.104  [3.009,330] IPv6 [fe80::e269:3aff:fe00:1390]-link  [3.009,394] IPv6 [2406:7400:63:5c3:e269:3aff:fe00:1390]  Case 1: Enable then disable  os\_suspend enable 1x  suspend\_count: 1  os\_suspend disable 1x  suspend\_count: 0  Case 2: Enable 2x then disable 1x  os\_suspend enable 2x  suspend\_count: 2  os\_suspend disable 1x  suspend\_count: 1  os\_suspend disable 1x  suspend\_count: 0  Case 3: Disable then enable  os\_suspend disable 1x  suspend\_count: -1  os\_suspend enable 1x  suspend\_count: 0 |