


# MT793X IoT SDK for Low Power User Guide

Version: 1.0

Release date: 2021-08-05

# Version History

| Version | Date | Author | Description |
|---|---|---|---|
| 0.1 | 2020-09-04 | JJ Chang | Initial draft |
| 1.0 | 2021-08-05 | JJ Chang | Official release |

# Table of Contents

## List of Figures



## List of Tables

# 1 Overview

Power consumption of the whole system is a key performance index to IoT product user experience. This document addresses the definition and design of MCU system's power mode and provides a guide for configuring the system into each low power modes.

# 2 Power Modes for the MT793X

This chapter introduces the power modes defined for the MT793X family MCU system.

## 2.1 Summary of Power Modes

There are 3 low power modes defined in MT793X family products. Every power mode can be triggered programmatically using API command call. Idle and deep sleep modes may be triggered from FreeRTOS unintendedly when the system enters idle mode to reduce overall power consumption.

***Table 1. MT793X Platform Power Modes***

| Power Modes | MCU clock | MCU bus | Wi-Fi | Clock | SRAM / PSRAM | Wake Up Source | Wake Up Latency | Power Consumption[1] |
|---|---|---|---|---|---|---|---|---|
| Active | Full speed | Active | On/Off | PLL / XTAL | Active | --- | --- | |
| Low Power - **CPU Idle** | Gated | Active | On/Off | PLL / XTAL | Active | All IRQ | < 1ms | < 25mW |
| Low Power - **Deep Sleep** | Power off | Power off | On/Off | RTC 32K | Sleep | All IRQ | < 10ms | < 3mW |
| Low Power - **RTC Mode** | Power off | Power off | Off | RTC 32K | Power off | RTC_EINT, RTC_TIMER | < 50ms | < 10uW |

## 2.2 Definition of Power Mode

### 2.2.1 Active Mode

While MCU is executing code, the processor operates in full speed, as shown in the table below, with all mandatory peripheral power and clocks on. The MCU bus has DCM feature enabled, which allows the bus to slow down clock rate on idle in order to reduce power consumption.

***Table 2. MT793x MCU Default Active Clock Rate***

| CHIP ID | MCU default full speed clock rate |
|---|---|
| MT7931AN | 200 MHz |
| MT7931AT | 300 MHz |
| MT7933CT | 300 MHz |

[1] Power consumption values are estimated and may be changed on different feature setting. Power consumption of connectivity sub-system is not included.

### 2.2.2 Low Power CPU Idle Mode and Deep Sleep Mode

The CPU idle mode and deep sleep mode can be triggered either manually or by OS tickless sleep feature itself. When FreeRTOS tickless sleep feature is enabled, OS idle task triggers system idle or deep sleep unintendedly depending on the next available duration of sleep slot. See section 3 for details.

In CPU idle mode, the processor enters WFI state. Clock of the processor is gated. All peripherals remain in operating state. The CPU idle mode is used when the sleep time slot is too short for deep sleep or when some hardware resources are sleep-locked and therefore does not allow the system to enter deep sleep.

In deep sleep mode, the goal is to minimize system power consumption while code can be resumed seamlessly after the system returns from sleep. Data in all memory is retained after resume. In this deep sleep mode, most peripherals power down or are configured to low power retention mode before sleep. Sleep lock mechanism in sleep manager can be used to control whether the system can enter deep sleep and which peripheral shall remain on during sleep.

Any interrupt registered to the sleep manager wakeup source may wake the system from deep sleep to active. In FreeRTOS tickless sleep feature, a RTC 32-kHZ clock generated GPT timer is also registered to wake the system for the next software task execution.

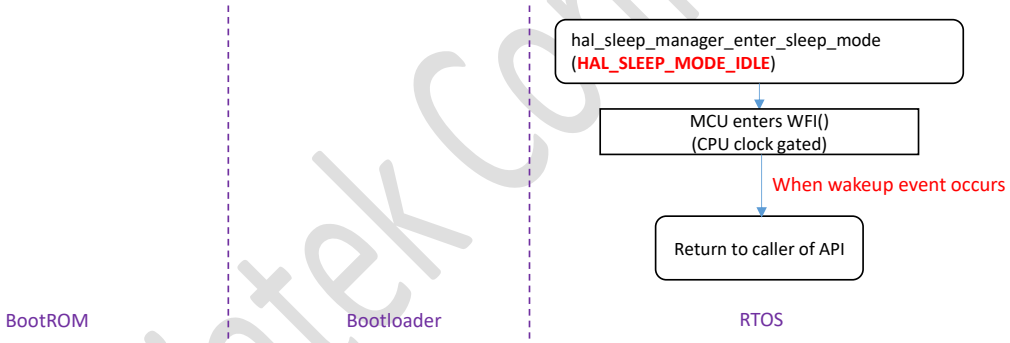


***Figure 1 Software Sleep/Wakeup Flow for CPU Idle Power Mode***

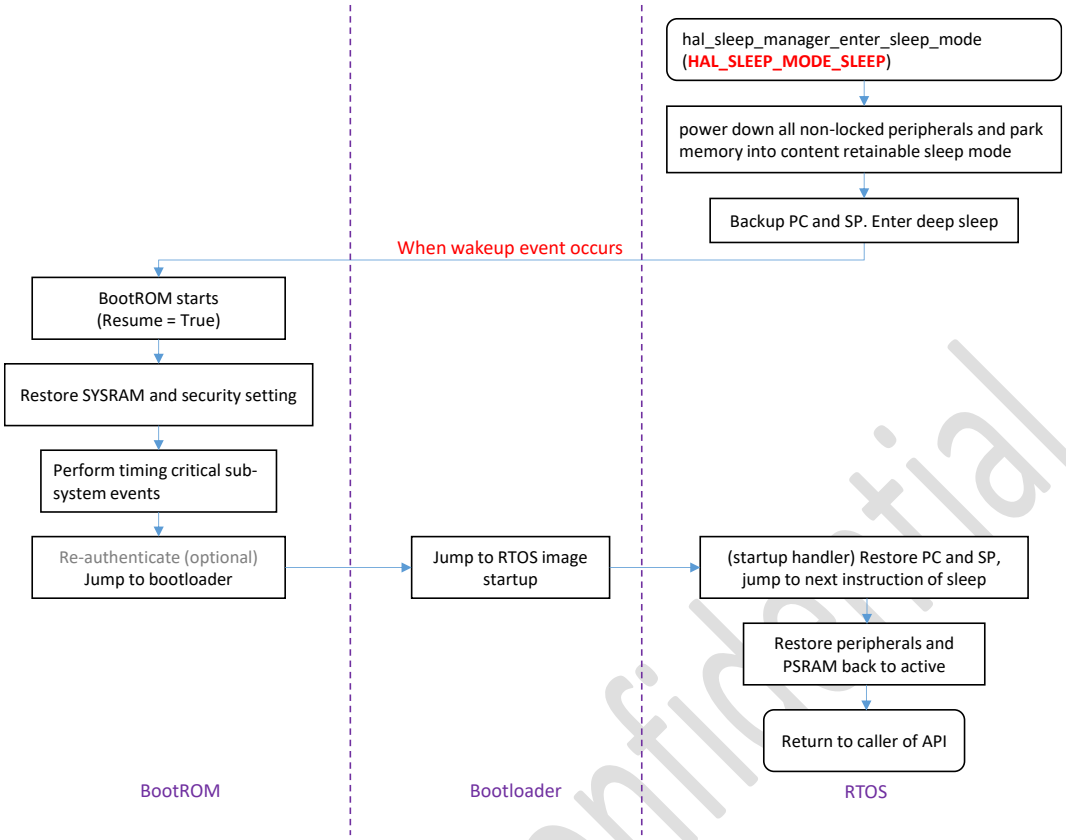


***Figure 2 Software Sleep/Wakeup Flow for Deep Sleep Power Mode***

### 2.2.3 Low Power RTC Mode

RTC low power mode is provided as an alternative power off mechanism if the power source cannot be fully cut out. In RTC mode, only the RTC hardware and the internal RTC circuit with 512 bytes of RTC memory are retained. All other peripherals and memory blocks are fully powered off.

The RTC timer and RTC_EINT signal can wake the system back from RTC mode to system cold boot.

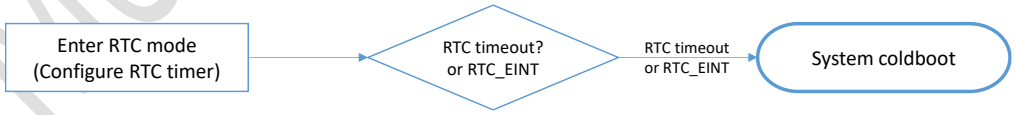


***Figure 3 Software Flow for RTC Power Mode***

# 3 FreeRTOS Tickless Sleep

## 3.1 Tickless Sleep Design

FreeRTOS provides tick suppression option to enable lower power consumption when no software task is running. For detailed concept of FreeRTOS tickless sleep, please refer to the FreeRTOS website:
https://www.freertos.org/low-power-tickless-rtos.html

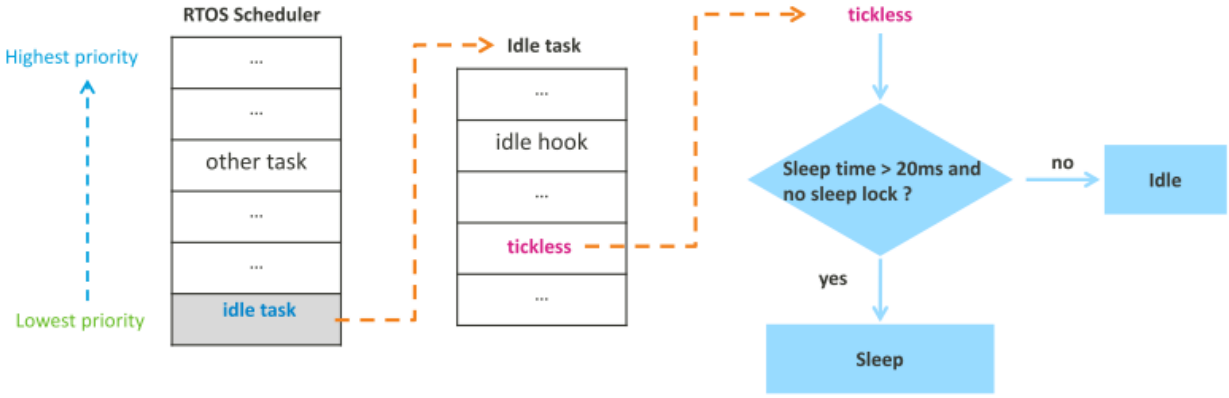


***Figure 4 Software Flow for Tickless Sleep***

If no sleep lock is acquired at the moment and the expected sleep time is larger than pre-estimated deep sleep overhead, the system places itself into deep sleep low power modes; otherwise, the system enters CPU idle low power mode to reduce energy.

## 3.2 Enabling FreeRTOS Low Power Tickless Sleep

To suspend the ticks, enable the tickless feature in each project's header file FreeRTOSConfig.h by setting "#define configUSE_TICKLESS_IDLE 2".
Before the system enters the tickless low power state, it first checks whether no sleep lock is held by users or modules and second, check whether the available idle time slot is long enough for idle or deep sleep mode. See Table 3 for how the criteria are defined.

***Table 3. Modes Selected by FreeRTOS Tickless Sleep***

| Power Modes | Available Idle Time Slot |
|---|---|
| Idle | < 20 ms |
| Deep Sleep | ≥ 20 ms |

# 4 Low Power Module and API

## 4.1 System Power Management Module

The System Power Management (SPM) module is responsible for controlling power switches of each chip internal power domain. It is fully controlled by the MT793X internal driver for optimized low power control sequence. You do not need to call any of the SPM interfaces in the MT793X SDK.

## 4.2 Clock Management Module (hal_clock.h)

The MT793X family offers several types of clock domains and clock gating/clock sources for you to choose from. The clock management module provides API to control clock gating switch and source selection of each clock. In the MT793X family, the clock gating switches are fully controlled by the driver of each module. You do not need to manually control clock gates as it may result in uncertainty. Most clock source selections are also controlled by system drivers. You may overwrite some clock sources for performance tuning only when necessary.

Below is a list of available clock sources that you can control.

***Table 4. List of MT793x User Controllable Clock Selection***

| Clock Name | Value in hal_clock_sel_id enumeration | Available Mux Sources (Default values are bold) | Controllable by user |
|---|---|---|---|
| MCU | HAL_CLOCK_SEL_CM33_HCLK | **300MHz, 200MHz**, XTAL | Yes |
| MCU bus | HAL_CLOCK_SEL_INFRA_BUS | **133MHz**, 120MHz, XTAL | Yes |

### 4.2.1 API Reference

#### 4.2.1.1 hal_clock_init

```
hal_clock_status_t hal_clock_init(void)
```

This function initializes the clock driver and CG before any clock API is used.

Returns:

**HAL_CLOCK_STATUS_OK**, if the operation completes successfully.
**HAL_CLOCK_STATUS_UNINITIALIZED**, if the clock driver is not initialized.
**HAL_CLOCK_STATUS_INVALID_PARAMETER**, if the input parameter is invalid.
**HAL_CLOCK_STATUS_ERROR**, if the clock function detects a common error.

#### 4.2.1.2 hal_clock_get_mcu_clock_frequency

`uint32_t` **`hal_clock_get_mcu_clock_frequency`**`(void)`

This function gets the MCU clock frequency.

Returns:

The MCU clock frequency in Hz.

#### 4.2.1.3 hal_clock_mux_select

`hal_clock_status_t` **`hal_clock_mux_select`**`(hal_clock_sel_id clock_id, uint32_t clk_sel)`

This function changes the source of clock; it also enables or disables the corresponding divider.

Parameters:

| | | |
|---|---|---|
| in | *clock_id* | Unique clock identifier in hal_clock_sel_id enum. |
| in | *clk_sel* | Enum value of clk_XXX_clksel_t based on clock_ID e.g. if clock_id=HAL_CLOCK_SEL_FLASH, it shall be the value in clk_flash_clksel_t enum |

Returns:

**HAL_CLOCK_STATUS_OK**, if the operation completes successfully.
**HAL_CLOCK_STATUS_UNINITIALIZED**, if the clock driver is not initialized.
**HAL_CLOCK_STATUS_INVALID_PARAMETER**, if the input parameter is invalid.
**HAL_CLOCK_STATUS_ERROR**, if the clock function detects a common error.

#### 4.2.1.4 hal_clock_get_selected_mux

`hal_clock_status_t` **`hal_clock_get_selected_mux`**`(hal_clock_sel_id clock_id, uint32_t *clk_sel)`

This function gets the current clock source in use.

Parameters:

| | | |
|---|---|---|
| in | *clock_id* | Unique clock identifier in hal_clock_sel_id enum. |
| out | *clk_sel* | Return enum value of clk_XXX_clksel_t based on clock_id |

Returns:

**HAL_CLOCK_STATUS_OK**, if the operation completes successfully.
**HAL_CLOCK_STATUS_UNINITIALIZED**, if the clock driver is not initialized.
**HAL_CLOCK_STATUS_INVALID_PARAMETER**, if the input parameter is invalid.
**HAL_CLOCK_STATUS_ERROR**, if the clock function detects a common error.

## 4.3 Sleep Management Module (hal_sleep_manager.h)

The sleep management module API offers four main features:

- **Enter power saving mode.**
  Check the sleep lock status by calling *hal_sleep_manager_is_sleep_locked* and then call *hal_sleep_manager_enter_sleep_mode* to enter dedicated sleep mode to save power.
- **Sleep lock control**
  Call *hal_sleep_manager_set_sleep_handle*, *hal_sleep_manager_release_sleep_handle*, *hal_sleep_manager_lock_sleep* and *hal_sleep_manager_unlock_sleep* to register and lock or unlock sleep user lock to prevent the system from entering low power mode when the condition is not met.
- **Deep sleep suspend and resume callback**
  Call hal_sleep_manager_register_suspend_callback() and hal_sleep_manager_register_resume_callback() to set callback function prior to and after the system returning from deep sleep to perform necessary backup/restore operation or low power setup.
- **Set wakeup time**
  Call *hal_sleep_manager_set_sleep_time()* to wake up the system after the system enters low power mode for a specific time in milliseconds.

### 4.3.1 API Reference

In this section, a list of commonly used API references is provided. For a full list of provided API, please refer to the comment in header file hal_sleep_manager.h

#### 4.3.1.1 hal_sleep_manager_init

```
hal_sleep_manager_status_t hal_sleep_manager_init(void)
```

This function initializes the sleep manager internal context.
This function must be called once prior to the use of other sleep manager API.

Returns:
**HAL_SLEEP_MANAGER_OK**, if the operation completes successfully.

#### 4.3.1.2 hal_sleep_manager_set_sleep_handle

```
uint8_t hal_sleep_manager_set_sleep_handle(const char *handle_name)
```

This function sets a sleep handle to control the sleep state of the system.

Parameters:

| | | |
|---|---|---|
| in | *handle_name* | NULL terminated customizable name for the sleep handle |

Returns:
**handle_index,** Index of the sleep handle assigned.
**0xFF**, if no available handle is available

#### 4.3.1.3 hal_sleep_manager_release_sleep_handle

```
hal_sleep_manager_status_t hal_sleep_manager_release_sleep_handle(uint8_t handle_index)
```

This function releases the sleep handle if it is no longer in use.

Parameters:

| | | |
|---|---|---|
| in | *handle_index* | Index of the sleep handle returned from hal_sleep_manager_set_sleep_handle() |

Returns:

**HAL_SLEEP_MANAGER_OK**, if the operation completes successfully.
**HAL_SLEEP_MANAGER_ERROR**, if an invalid handle_index is provided.

#### 4.3.1.4 hal_sleep_manager_lock_sleep

```
hal_sleep_manager_status_t hal_sleep_manager_lock_sleep(uint8_t handle_index)
```

This function prevents the MCU from getting into sleep or deep sleep mode.

Parameters:

| | | |
|---|---|---|
| in | *handle_index* | Index of the sleep handle returned from hal_sleep_manager_set_sleep_handle() |

Returns:

**HAL_SLEEP_MANAGER_OK**, if the operation completes successfully.

#### 4.3.1.5 hal_sleep_manager_unlock_sleep

```
hal_sleep_manager_status_t hal_sleep_manager_unlock_sleep(uint8_t handle_index)
```

This function unlocks the specific sleep lock and permits the MCU to go into sleep mode when needed if all sleep locks are unlocked.

Parameters:

| | | |
|---|---|---|
| in | *handle_index* | Index of the sleep handle returned from hal_sleep_manager_set_sleep_handle() |

Returns:

**HAL_SLEEP_MANAGER_OK**, if the operation completes successfully.

#### 4.3.1.6 hal_sleep_manager_register_suspend_callback

```
void hal_sleep_manager_register_suspend_callback(hal_sleep_manager_callback_t callback, void *data)
```

Register a callback function to be called before the system enters sleep mode. At most two callbacks can be registered in the MT793X SDK. An error message will be shown on the console if you reach the limit.

Parameters:

| | | |
|---|---|---|
| in | *callback* | User defined callback function to be registered. |
| in | *data* | User defined data to be used in the callback function |

#### 4.3.1.7 hal_sleep_manager_register_resume_callback

```
void hal_sleep_manager_register_resume_callback(hal_sleep_manager_callback_t callback, void *data)
```

Register a callback function to be called after the system leaves sleep mode. At most two callbacks can be registered in the MT793X SDK. An error message will be shown on the console if you reach the limit.

Parameters:

| | | |
|---|---|---|
| in | *callback* | User defined callback function to be registered. |
| in | *data* | User defined data to be used in the callback function |

#### 4.3.1.8 hal_sleep_manager_is_sleep_locked

```
bool hal_sleep_manager_is_sleep_locked(hal_sleep_mode_t mode)
```

This function checks if there are any sleep locks with specific level in the system. Apply this function before the system enters the sleep mode.

Parameters:

| | | |
|---|---|---|
| in | *mode* | The sleep mode to be checked<br>HAL_SLEEP_MODE_IDLE: WFI: CPU idle only<br>HAL_SLEEP_MODE_SLEEP: Deep Sleep: all mtcmos off, Xtal off, Wakeup reboot from BROM |

Returns:

**true**, if any locks with the specific level are on hold
**false**, if the system can enter the sleep mode

#### 4.3.1.9 hal_sleep_manager_enter_sleep_mode

```
uint32_t hal_sleep_manager_enter_sleep_mode(hal_sleep_mode_t mode)
```

This function sets the system to any of the modes defined in *hal_sleep_mode_t*.

Parameters:

| | | |
|---|---|---|
| in | *mode* | The desired sleep mode<br>HAL_SLEEP_MODE_IDLE: WFI: CPU idle only<br>HAL_SLEEP_MODE_SLEEP: Deep Sleep: all mtcmos off, Xtal off, Wakeup reboot from BROM |

Returns:

**<0x10000000:** The system enters the sleep mode successfully. Return value represents the IRQ number of the wakeup source.
**≥0x10000000:** The system does not enter the sleep mode successfully. Return value represents error code with the abort reason.

# 5 Programming Reference

## 5.1 Low Power Optimization

Lastly, this section provides several tips for the MT793X product to achieve better low power performance.

**Tip 1: Avoid initializing unnecessary module.**

In theMT793X, power and clocks are optimized for minimum energy usage. The power and clock required by each function are turned on during the driver initialization progress. To reduce the power consumed by unused modules, avoid calling initialization function (or calling de-initialization function, if provided, after module is initialized) may reduce unnecessary power consumption.

**Tip 2: Set unnecessary GPIO pins to high impendence mode**

Power leakage from IO pin is a common cause of power waste in low power devices. Setting unused IO pins to high impendence mode may avoid these types of current leaks. Refer to GPIO document for details.

# Exhibit 1 Terms and Conditions

Your access to and use of this document and the information contained herein (collectively this "Document") is subject to your (including the corporation or other legal entity you represent, collectively "You") acceptance of the terms and conditions set forth below ("T&C"). By using, accessing or downloading this Document, You are accepting the T&C and agree to be bound by the T&C. If You don't agree to the T&C, You may not use this Document and shall immediately destroy any copy thereof.

This Document contains information that is confidential and proprietary to MediaTek Inc. and/or its affiliates (collectively "MediaTek") or its licensors and is provided solely for Your internal use with MediaTek's chipset(s) described in this Document and shall not be used for any other purposes (including but not limited to identifying or providing evidence to support any potential patent infringement claim against MediaTek or any of MediaTek's suppliers and/or direct or indirect customers). Unauthorized use or disclosure of the information contained herein is prohibited. You agree to indemnify MediaTek for any loss or damages suffered by MediaTek for Your unauthorized use or disclosure of this Document, in whole or in part.

MediaTek and its licensors retain titles and all ownership rights in and to this Document and no license (express or implied, by estoppels or otherwise) to any intellectual propriety rights is granted hereunder. This Document is subject to change without further notification. MediaTek does not assume any responsibility arising out of or in connection with any use of, or reliance on, this Document, and specifically disclaims any and all liability, including, without limitation, consequential or incidental damages.

THIS DOCUMENT AND ANY OTHER MATERIALS OR TECHNICAL SUPPORT PROVIDED BY MEDIATEK IN CONNECTION WITH THIS DOCUMENT, IF ANY, ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE. MEDIATEK SPECIFICALLY DISCLAIMS ALL WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, COMPLETENESS OR ACCURACY AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MEDIATEK SHALL NOT BE RESPONSIBLE FOR ANY MEDIATEK DELIVERABLES MADE TO MEET YOUR SPECIFICATIONS OR TO CONFORM TO A PARTICULAR STANDARD OR OPEN FORUM.

Without limiting the generality of the foregoing, MediaTek makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does MediaTek assume any liability arising out of the application or use of any product, circuit or software. You agree that You are solely responsible for the designing, validating and testing Your product incorporating MediaTek's product and ensure such product meets applicable standards and any safety, security or other requirements.

The above T&C and all acts in connection with the T&C or this Document shall be governed, construed and interpreted in accordance with the laws of Taiwan, without giving effect to the principles of conflicts of law.