MediaTek LinkIt™ Development Platform for RTOS Power Mode Developer's guide

Version: 1.2

Release date: 13 January 2017

Document Revision History

Revision	Date	Description
1.0	30 June 2016	Initial version
1.1	4 November 2016	• Added MT7687 RTC mode description • Added power mode description for MT2533
1.2	13 January 2017	• Added debug limitation for MT2523/MT7687

Table of contents

Lists of tables and figures

1. Introduction

The system power consumption is a key performance index to user experience and the measurement includes power consumption of data for MCU, connectivity and peripheral systems.

- MCU system includes a processor, memory and related clock source.
- Connectivity system includes the baseband, memory, RF and related clock source.
- Peripheral system usually contains UART, I2C, SPI, GPIO, and more.

This document addresses the MCU system's power mode configuration and power consumption measurement focused on power modes provided by MediaTek LinkIt™ development platform for RTOS.

Detailed description on how to measure the power on chipsets can be found at:

- `<sdk_root>/doc/HDK/LinkIt_2523_HDK_Power_Measurement_Guide.pdf`
- `<sdk_root>/doc/HDK/LinkIt_7687_HDK_Power_Measurement_Guide.pdf`

Note: There is no power measurement guide for MediaTek MT2533.

2. Power Modes and Low Power Configuration for MT2523 and MT2533

In this section, the MCU system's power mode and low power configuration is introduced.

- Power mode for the MCU is described in section 2.1, "Power mode."
- MCU clock configurations are described in section 2.2, "Configuring the MCU clock."
- FreeRTOS low power feature is in section 2.3, "Enabling the FreeRTOS low power mode."

2.1. Power modes

The power consumption value of each power mode can be found in the chipset's datasheet under `<sdk_root>/doc/MCU` and the detailed description on how to switch the power mode can be found in the DVFS and Sleep Manager module of the API Reference Manual under `<sdk_root>/doc`. The power modes for MT2523 and MT2533 are summarized below and in Table 1.

- High Speed/Full Speed/Low Speed
    - CPU and RAMs are in active state, instructions can be executed and the peripheral access is active.
- Idle
    - CPU enters clock-gated state to save core power and internal and external interrupts can wake up the processor.
    - RAMs (PSRAM and RAM) are in idle state.
- Sleep or Deep Sleep
    - CPU power is turned off to save more power.
    - RAM (PSRAM and RAM) enters the lowest power state, while the data is preserved.
    - All peripherals such as I2C, UART and SPI are powered off.

The differences between Sleep and Deep Sleep are that Sleep state keeps the core power (V_{CORE}) unchanged, on the other hand, Deep Sleep state changes the core power (V_{CORE}) to a lower value to save more power.

- Off
    - Only RTC power is retained.
    - RAM (PSRAM and RAM) enters power-down state and data is lost.

Table 1. The power modes for MT2523/MT2533

Mode	Description	MCU clock source	V_{CORE} voltage minimum value	RAM	PSRAM	FLASH	Peripherals
High Speed	The maximum frequency of	PLL	1.3V	Active	Active	Active	Active

Mode	Description	MCU clock source	V_{CORE} voltage minimum value	RAM	PSRAM	FLASH	Peripherals
	MCU is 208MHz.						
Full Speed	The maximum frequency of MCU is 104MHz.	HFOSC	1.1V				
Low Speed	The maximum frequency of MCU is 26MHz.	LFOSC/DCXO (*)	0.9V				
Idle	The MCU clock is gated to save core power.	Not available	Same as active state	Full Retention (Data kept)	Idle	Idle	Active
Sleep	The MCU power is turned off to achieve lower current consumption.	Not available	0.9V	Full Retention (Data kept)	Self-refresh (Data kept)	Deep Power Down	Power Off
Deep Sleep(**)			0.7V				
Off	Only RTC power is retained.	Not available	Not available	Power Off (Data Lost)	Power Off (Data Lost)	Power Off	Power Off

Note:

(*) The selection of MCU clock source in Low Speed state is determined during compilation (see section 2.2, “Configuring the MCU clock”).

(**) Optional for MT2533 — it requires a companion PMIC to adjust the supply voltage of V_{CORE} by hardware control signal.

2.2. Configuring the MCU clock

MT2523 clock settings are configured using Clock Management driver that controls clock dividers and MUXs. Two things are addressed:

1) Selecting the default clock frequency for ARM Cortex-M4 MCU clock.

Cortex-M4 clock supports 26MHz (Low Speed), 104MHz and 208MHz (divided from PLL). The default MCU clock setting is configured during compilation.

Based on minimum requirements of V_{CORE} voltage, you can provide either one of the options in your project’s configuration file.

- `MTK_SYSTEM_CLOCK_208M`
- `MTK_SYSTEM_CLOCK_104M`
- `MTK_SYSTEM_CLOCK_26M`

2) Specifying the clock source in low speed power mode for the MCU clock.

- Digitally Controlled Oscillator (DCXO) is selected as default clock source in low speed power mode. The Low Frequency Oscillator (LFOSC) can be used by adding `MTK_CLK_USE_LFOSC` in project's `hal_feature_config.h`.
- The selection guideline could be based on the following:
  - The DCXO has better accuracy than LFOSC.
  - The LFOSC consumes less power than DCXO.

The configuration of MCU clock is defined in each project's configuration header file, such as `<SDK_ROOT>/project/mt2523_hdk/apps/iot_sdk_demo/inc/hal_feature_config.h`.

2.3. Enabling the FreeRTOS low power mode

FreeRTOS provides tick suppression option to enable lower power consumption while idle task is scheduled.

To suspend the ticks, enable the tickless feature in each project's header file, `FreeRTOSConfig.h`, as shown below.

- Enable tickless feature using the setting "`#define configUSE_TICKLESS_IDLE 2`".

In the tickless porting files (*), there are two conditions to be verified before entering the low power state. The first condition is that no sleep lock is held by users (**) and the second is to check if the sleep time exceeds 20ms.

- If the sleep time exceeds 20ms, **Deep Sleep** state is chosen.
  - To prevent the system from entering the **Deep Sleep** state, Sleep Manager module provides functions to lock or unlock the sleep mode. More details can be found in the API Reference Manual under `<sdk_root>/doc`.
- If the sleep time is less than 20ms, **Idle** state is chosen.

The tickless flow is shown in Figure 1.

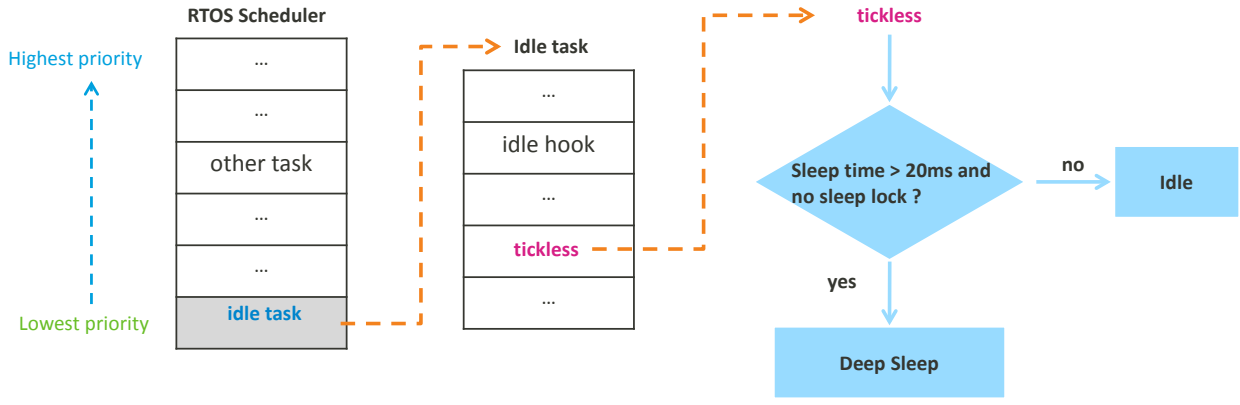

Figure 1. FreeRTOS tickless feature flow in MT2523/MT2533

Note:

(*) The porting files can be found under `<sdk_root>/kernel/rtos/FreeRTOS/Source/portable`

(**) By default, the Sleep Manager will hold a sleep lock to prevent the system from entering the low power state. Users need to use the Sleep Manager API, `hal_sleep_manager_unlock_sleep(uint8_t handle_index)`, to release the sleep lock before checking the tickless function. See the API Reference Manual under `<sdk_root>/doc`, for more details.

2.3.1. Debugging limitations in FreeRTOS low power mode

The MCU (Cortex-M4) is gated when the system enters into Deep Sleep mode from the FreeRTOS idle task. The debug identification information is stored in Cortex-M4 ROM table and cannot be accessed when a debugger is attached to the LinkIt 2523 HDK through Keil IDE or IAR workbench IDE. Thus, the resources to apply CPU debugging are all occupied.
Because of this limitation, the Deep Sleep mode needs to be disabled in order to debug. To disable the Deep Sleep mode:

- Call the Sleep Manager API, `hal_sleep_manager_lock_sleep(uint8_t handle_index)` to disable the Deep Sleep mode.

3. Power Modes and Current Measurement for MT7687

This section provides details on the supported power modes and current measurement.

- Supported power modes are described in section 3.1, “Power modes”.
- FreeRTOS low power feature in MT7687 is described in section 3.2, “Enabling the FreeRTOS low power mode.”

3.1. Power modes

The HAL low power driver provides a CPU sleep interface to configure the power mode based on the application requirements. MT7687 provides different power modes — Idle, Wait for Interrupt (WFI), Legacy Sleep and RTC Mode, as shown in Table 2.

Table 2. MT7687 power modes

Power Mode	CPU clock	Cortex-M4 MCUSYS bus clock	PLL	RAM
Idle	Gated	---	Active	Active
WFI	Gated	---	Off	Active
Legacy Sleep	Gated	Gated	Off	Sleep
RTC Mode	Powered Off	Powered Off	Off	Powered Off

To reduce the power consumption, the system can determine the proper timing to enter into the desired power mode. The CPU and its related modules will also enter the proper power mode to save power and extend the battery life.

In Idle mode, MCU remains in the sleep state until woken up by an interrupt. When MCU is in Idle mode:

- CPU clock is gated to keep power consumption at a lower level.
- RAM is still active.

In WFI mode, MCU remains in the sleep state until woken up by an interrupt. When MCU is in WFI mode:

- CPU clock is gated to keep power consumption at a lower level.
- PLL is turned off.
- RAM is still active.

This mode is ideal for a short period time of sleep, as the required time to wake up from WFI mode is much shorter than from the Legacy Sleep mode. If it’s required to keep the peripherals active during the sleep, the WFI mode is the only option, because the bus clock is not gated. The measured floor current in WFI mode is about 9.97mA, as shown in Figure 2.

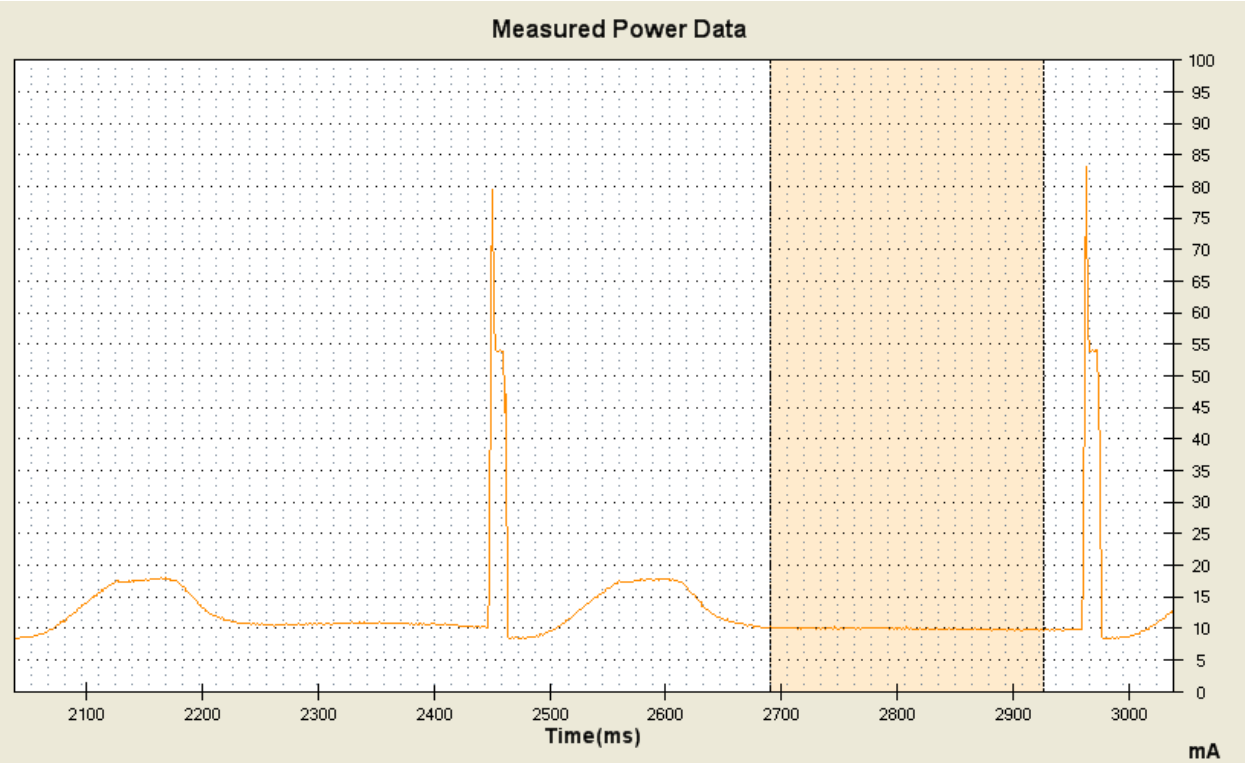

Figure 2. The floor current for MT7687 WFI sleep mode

Legacy Sleep mode provides much lower power consumption during the sleep. The following behavior is observed when MCU is in the Legacy Sleep mode.

- CPU clock is gated.
- PLL is turned off.
- Bus clock is gated.
- RAM is in sleep mode and the data is retained.

This mode is ideal for a long period time of sleep, because the power performance is better than in WFI mode. However, it needs 50µs more to wake up from the sleep to execute instructions. This mode does not support active peripheral configuration during sleeping. The measured floor current in this mode is about 1.12mA, as shown in Figure 3.

Any interrupt can wake up the MCU from the Legacy Sleep mode, to exit the sleep mode.

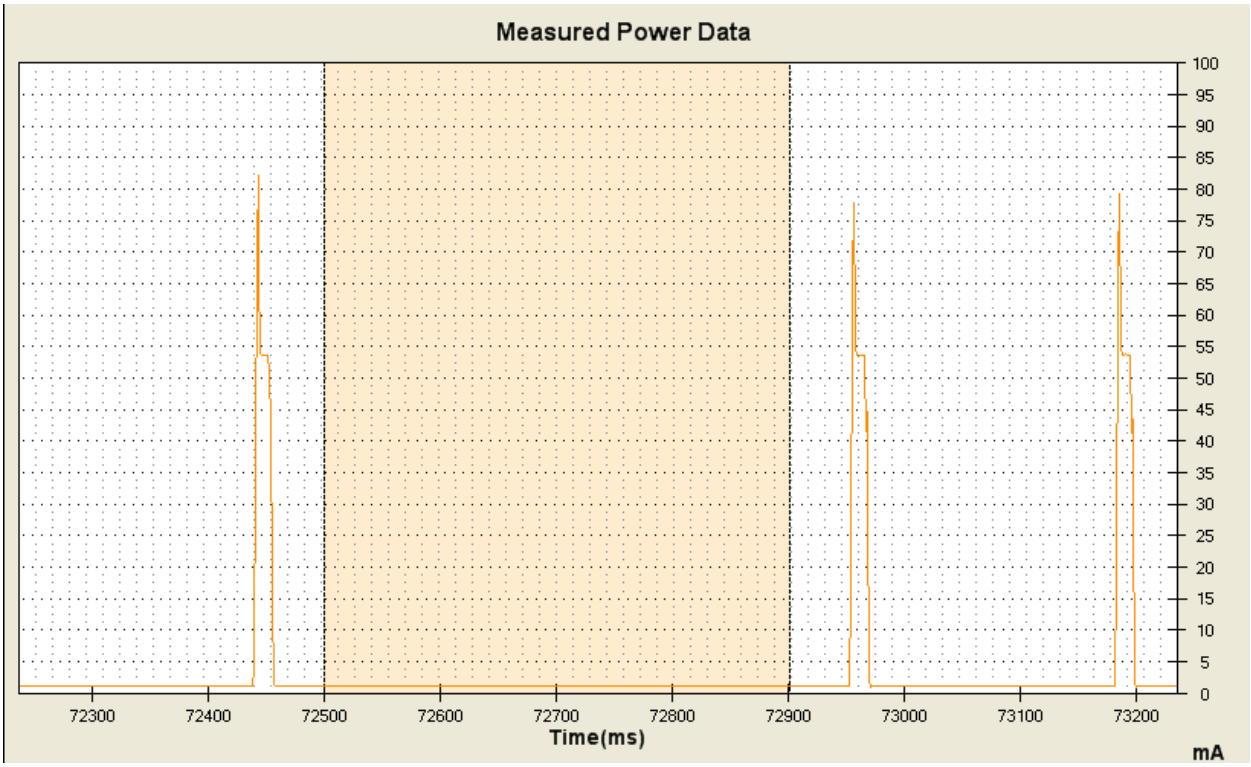

Figure 3. Floor current for MT7687 Legacy Sleep mode

In RTC mode, MCU remains powered off until it's powered up by an RTC alarm interrupt. When MCU is in RTC mode:

- CPU clock is powered off.
- RAM is powered off.

3.2. Enabling the FreeRTOS low power mode

FreeRTOS provides tick suppression option to enable lower power consumption, while idle task is scheduled.

To suspend the ticks, enable the tickless feature in each project's header file, `FreeRTOSConfig.h`, as shown below.

- Enable the tickless feature using the setting **"`#define configUSE_TICKLESS_IDLE 2`"**.

In MT7687's tickless porting files (*), there are two conditions to be verified before determining to enter the low power state. The first condition is that no sleep lock is held by users and the second is to check if the sleep time exceeds 10ms.

- If the sleep time exceeds 10ms, **WFI** state is chosen by default.
    - To prevent the system from entering a sleep state, Sleep Manager module provides functions to lock or unlock the sleep mode.
- If the sleep time is less than 10ms, **Idle** state is chosen.

The tickless flow is shown in Figure 4.

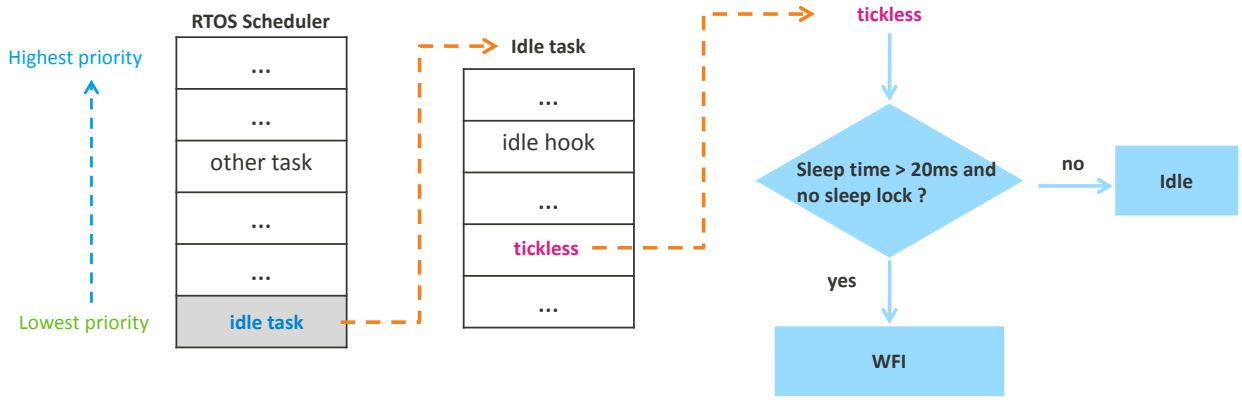

Figure 4. MT7687's FreeRTOS tickless feature flow

Note: (*) The porting files can be found under `<sdk_root>/kernel/rtos/FreeRTOS/Source/portable`.

3.2.1. Debugging limitations in FreeRTOS low power mode

The MCU (Cortex-M4) is gated when the system enters into low power mode from the FreeRTOS idle task. The debug identification information is stored in Cortex-M4 ROM table and cannot be accessed when a debugger is attached to the LinkIt 7687 HDK through Keil IDE or IAR workbench IDE. Thus, the resources to apply CPU debugging are all occupied.
Because of this limitation, the low power mode needs to be disabled in order to debug. To disable the low power mode:

- Disable the tickless feature using the setting "`#define configUSE_TICKLESS_IDLE 0`".

4. Appendix: CPU Benchmarking

CoreMark® is an industry-standard benchmark for embedded systems that aim to measure the performance of central processing units (CPU). The code is in C and can be found here.

Besides performance measurement, you can use CoreMark as a typical active workload on MCU to measure the power consumption during the benchmark's execution.

4.1. Integrating CoreMark in your project

To integrate benchmark in your project:

1) Download CoreMark software from EEMBC website.
2) Add CoreMark source files (see Figure 5) in your Makefile or preferred IDE (GCC, IAR embedded workbench IDE, Keil µVision IDE).

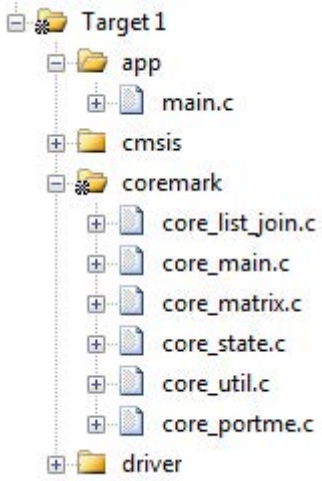

Figure 5. An example project with CoreMark source files on µVision IDE

3) Port `core_portme.h` and `core_portme.c` files to adapt to your platform.
   a) Follow the comments and instructions in the source and header files to configure the porting files based on your development platform and application requirements.
4) Change the name of `main()` in the source file `core_main.c` to `coremark_main()`, to avoid build failure due to two main functions in the project.
5) Call `coremark_main()` after logging and system clock initialization.

4.2. Compiler flags to build the benchmark

- CoreMark score varies due to different compilation optimization levels.
- For performance measurement, use the following option.
- `-O3` for GCC
- `-O3 -Otime` for Keil
- `-Ohs` for IAR

4.3. CoreMark report

After building the CoreMark project successfully, download the image to target board and power it on. Once the CoreMark execution is complete, a standard CoreMark report is generated, similar to this:

```
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 6482640
Total time (secs): 197.834473
Iterations/Sec   : 465.035233
Iterations       : 92000
Compiler version : KEIL v5.15 armcc V5.05 update2 (build 169)
Compiler flags   : --c99 -c --cpu Cortex-M4.fp -D__MICROLIB -g -O3 -Otime
--apcs=interwork --split_sections
Memory location  : STATIC
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x65c5
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 465.035233 / KEIL v5.15 armcc V5.05 update2 (build 169) --
c99 -c --cpu Cortex-M4.fp -D__MICROLIB -g -O3 -Otime --apcs=interwork --
split_sections / STATIC
```

Note:

- It's highly recommended to read the `readme.txt` in CoreMark package.
- Adjust `seed4_volatile` in `core_portme.c` to set different iterations. Make sure the run duration is over 10s.
- If `HAS_PRINTF` is set in `core_portme.h`, make sure the standard `printf()` works on your platform.