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

Version:	1.0
Release date:	19 October 2017

Document Revision History

Revision	Date	Description
1.0	19 Oct 2017	Initial release

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.

2. Power Modes and Low Power Configuration for MT2625

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 modes".
- FreeRTOS low power feature is in section 2.2, "Enabling the FreeRTOS low power mode".

2.1. Power modes

The HAL low power driver provides a CPU sleep interface to configure the power mode based on the application requirements. MT2625 provides different power modes — Active, Idle Sleep, Light Sleep, Deep Sleep, Power Off and Ship Mode. For more details, refer to `HAL/SLEEP_MANAGER in "<sdk_root>/doc/LinkIt SDK for 2625 API Reference Manual.html"`.

The MT2625 power state diagram is shown in Figure 1.

- Active
  - At this state, the frequency of Cortex-M4 can be configured at 104MHz or 26MHz. Instructions can be executed and the peripheral access is active.
- LIGHTSLEEP
  - System enters this state when a specific idle time is available. PMU can be changed to the low power mode to lower current consumption with 32K clock only.
- DEEPSLEEP
  - When Modem enters sleep state and the system is in idle state, MT2625 can enter DEEPSLEEP mode. The **DEEPSLEEP** mode provides a lower current consumption than **LIGHTSLEEP** mode with the ability for 8K SRAM retention. It is suitable for the applications that could be idle in a long period but also need to keep data in SRAM. It is controlled by software to enter the **DEEPSLEEP** mode and exited by RTC or PWRKEY event.
- POWEROFF
  - The Power **OFF** mode is similar to **DEEPSLEEP** mode, but no data is retained in SRAM.
- SHIP
  - **SHIP** mode, provides the lowest power consumption and only some logics to detect the power on conditions are available.

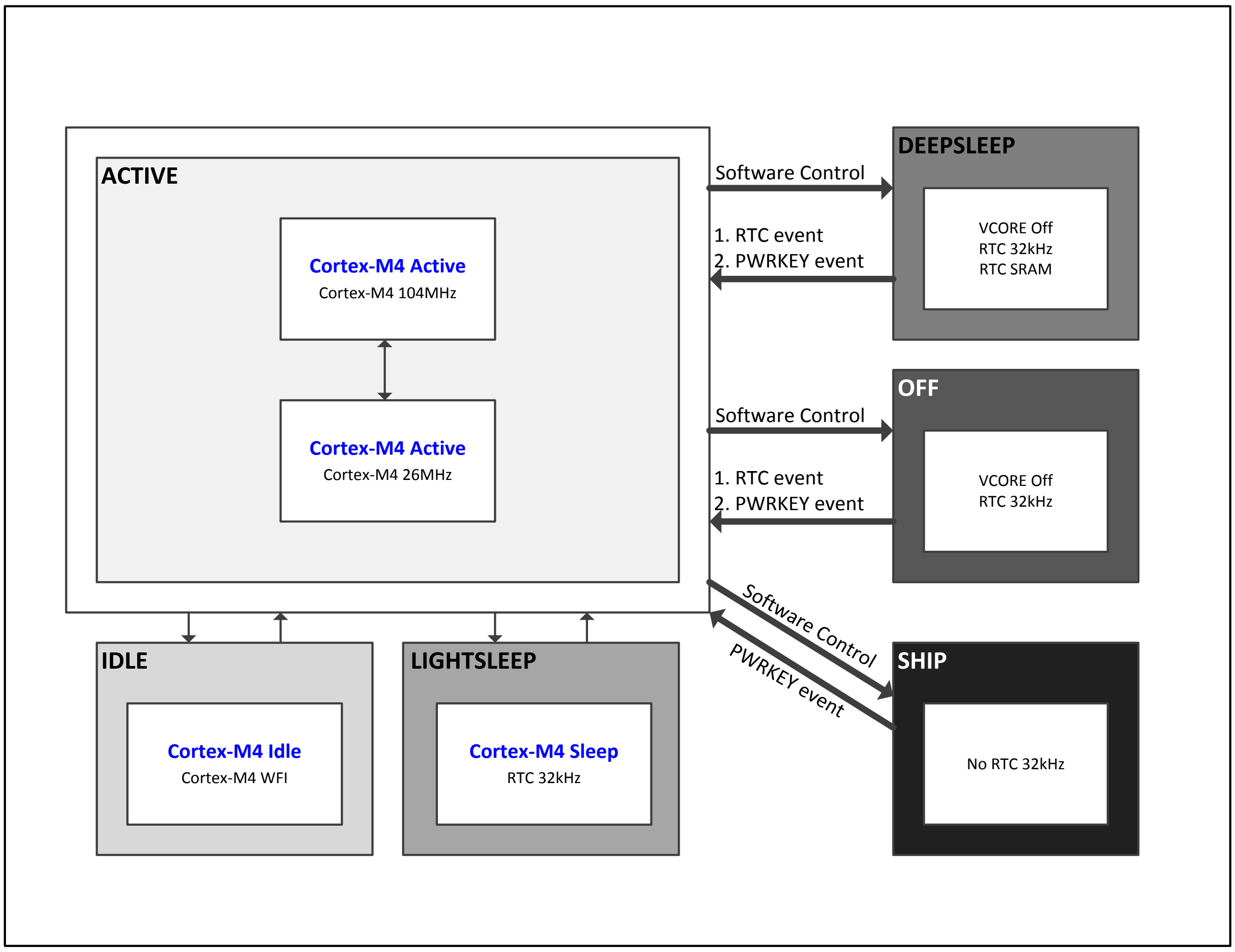

Figure 1 MT2625 Power States

2.2. Enabling the FreeRTOS low power mode

FreeRTOS provides tick suppression option to enable lower power consumption while an 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`”.

3. 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.

3.1. 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 2) in your Makefile or preferred IDE (GCC, IAR embedded workbench IDE, Keil µVision IDE).

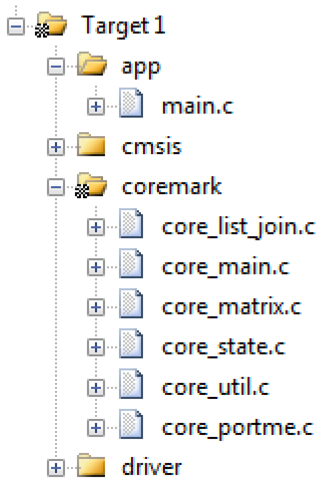

Figure 2 An example project with CoreMark source files on µVision IDE

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.

3.2. 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

3.3. 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.