Airoha IoT SDK for BT Audio Power Mode Developer's guide

Version: 1.0

Release date: 7 July 2020

Document Revision History

Revision	Date	Description
1.0	7 July 2020	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 Airoha IoT development platform.

2. Power Modes and Current Measurement for AB155x

This section introduces the MCU system's power mode and low-power configuration.

- Power mode for the MCU is described in Section 2.1."Power modes".
- The FreeRTOS low-power feature is described in Section 2.2. "Enabling the FreeRTOS low-power mode".

2.1. Power modes

A detailed description of how to switch the power mode can be found in the Sleep Manager module of the API Reference Manual under `<sdk_root>/mcu/doc`. The power modes for AB155x are summarized below and in Table 1.

- Active
  - The CPU and RAM are in an active state. Instructions can be executed and the peripheral access is active.
- Idle
  - The CPU goes into a clock-gated state to save core power. Internal and external interrupts can wake up the processor.
  - RAM (PSRAM and RAM) is in an idle state.
- Sleep
  - CPU power is turned off to save more power.
  - RAM and PSRAM (if any) go into the lowest power state and the data is preserved.
  - All peripherals including I2C, UART, and SPI are powered off.
- RTC
  - Only RTC power is retained.
  - RAM (PSRAM and RAM) goes into a power-down state and data is lost.

Table 1. The power modes for AB155x

Mode	Description	MCU clock source	V_{CORE} voltage minimum value	RAM	PSRAM(*)	FLASH	Peripherals
High Speed	The maximum frequency of CM4 is 156MHz. DSP is 312MHz.	PLL	1.3V	Active	Active	Active	Active
Full Speed	The maximum frequency of CM4 is 78MHz. DSP is 156MHz.	LPOSC	1.1V				
Low Speed	The maximum frequency of CM4 is 26MHz. DSP is 52MHz.	LPOSC	0.9V				

Mode	Description	MCU clock source	V_{CORE} voltage minimum value	RAM	PSRAM(*)	FLASH	Peripherals
Idle	The CM4/DSP clock is gated to save core power.	Not available	Same as active state	Data kept	Idle	Idle	Active
Sleep	The CM4/DSP power is turned off to achieve lower current consumption.	Not available	0.7V	Data kept	Data kept	Deep Power Down	Power Off
RTC	Only RTC power is retained.	Not available	Not available	Power Off (Data Lost)	Power Off (Data Lost)	Power Off	Power Off
Off	-	Not available	Not available	Power Off (Data Lost)	Power Off (Data Lost)	Power Off	Power Off

Note: The information in this column is only for AB1558 which has PSRAM.

2.2. Enabling the FreeRTOS low-power mode

FreeRTOS provides a tick suppression option to enable lower power consumption when 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 the tickless feature by using the setting "`#define configUSE_TICKLESS_IDLE 2`".

There are two conditions in the tickless porting files (*) that are verified before going into the low-power state. The first condition is that no sleep lock is held by users (**). The second condition is to check if the sleep time is more than 10ms.

- The system goes into the **Sleep** state if the sleep time is more than 10ms.
  - To prevent the system from going into the **Sleep** state, the Sleep Manager module provides functions to lock or unlock the sleep mode. Please refer to the API Reference Manual under `<sdk_root>/doc` for more information.
- The system goes into the **Idle** state if the sleep time is less than 10ms.

Figure 1. AB155x FreeRTOS tickless feature flow shows the tickless process for AB155x.

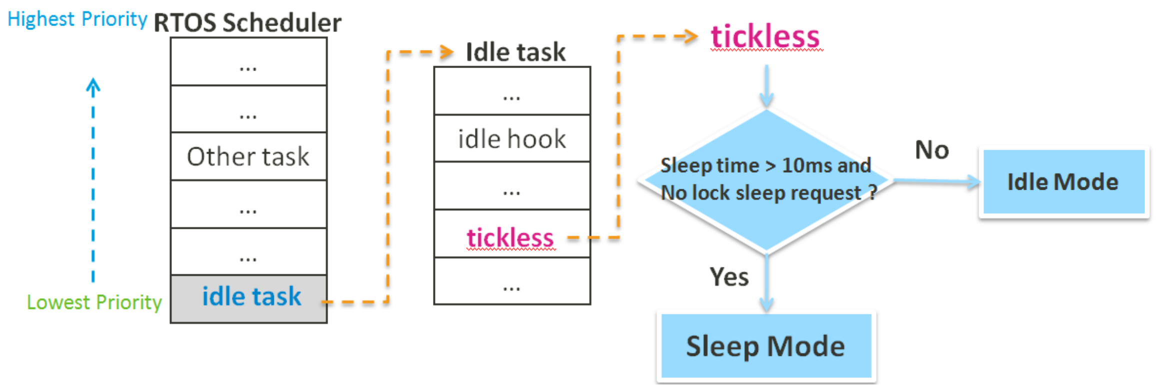

Figure 1. AB155x FreeRTOS tickless feature flow

2.2.1. Debugging limitations in FreeRTOS low-power mode

The MCU (Cortex-M4) is gated when the system goes into Sleep mode from the FreeRTOS idle task. The debugging identification information is stored in the Cortex-M4 ROM table and cannot be accessed when a debugging tool is attached to the AB155x EVK through a workbench IDE. Therefore, all resources for CPU debugging are busy. Because of this limitation, the Sleep mode must be disabled to allow debugging. To disable the Sleep mode:

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

3. Power Modes and Current Measurement for AB1565

This section introduces the MCU system's power mode and low-power configuration.

Power mode for the MCU is described in Section 3.1."Power mode".

The FreeRTOS low-power feature is described in Section 3.2. "Enabling the FreeRTOS low-power mode"

3.1. Power mode

A detailed description of how to switch the power mode can be found in the Sleep Manager module of the API Reference Manual under `<sdk_root>/mcu/doc`. The power modes for AB1565 are summarized below and in Table 2.

- Active
  - The CPU and RAM are in an active state. Instructions can be executed and the peripheral access is active.
- Idle
  - The CPU goes into a clock-gated state to save core power. Internal and external interrupts can wake up the processor.
  - RAM is in an idle state.
- Sleep
  - CPU power is turned off to save more power.
  - RAM go into the lowest power state and the data is preserved.
  - All peripherals including I2C, UART, and SPI are powered off.
- RTC
  - Only RTC power is retained.
  - RAM goes into a power-down state and data is lost.

Table 2. The power modes for AB1565

Mode	Description	MCU clock source	V_{CORE} voltage minimum value	RAM	FLASH	Peripherals
High Speed	The maximum frequency of CM4 is 208MHz. DSP is 416MHz.	LPOSC	0.9V	Active	Active	Active
Full Speed	The maximum frequency of CM4 is 104MHz. DSP is 208MHz.	LPOSC	0.8V			

Mode	Description	MCU clock source	V_{CORE} voltage minimum value	RAM	FLASH	Peripherals
Idle	The CM4/DSP clock is gated to save core power.	Not available	Same as active state	Data kept	Idle	Active
Sleep	The CM4/DSP power is turned off to achieve lower current consumption.	Not available	0.8V	Data kept	Deep Power Down	Power Off
RTC	Only RTC power is retained.	Not available	Not available	Power Off (Data Lost)	Power Off	Power Off
Off	-	Not available	Not available	Power Off (Data Lost)	Power Off	Power Off

3.2. Enabling the FreeRTOS low-power mode

FreeRTOS provides a tick suppression option to enable lower power consumption when 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 the tickless feature by using the setting "`#define configUSE_TICKLESS_IDLE 2`".

There are two conditions in the tickless porting files (*) that are verified before going into the low-power state. The first condition is that no sleep lock is held by users (**). The second condition is to check if the sleep time is more than 10ms.

- The system goes into the **Sleep** state if the sleep time is more than 10ms.
  - To prevent the system from going into the **Sleep** state, the Sleep Manager module provides functions to lock or unlock the sleep mode. Please refer to the API Reference Manual under `<sdk_root>/doc` for more information.
- The system goes into the **Idle** state if the sleep time is less than 10ms.

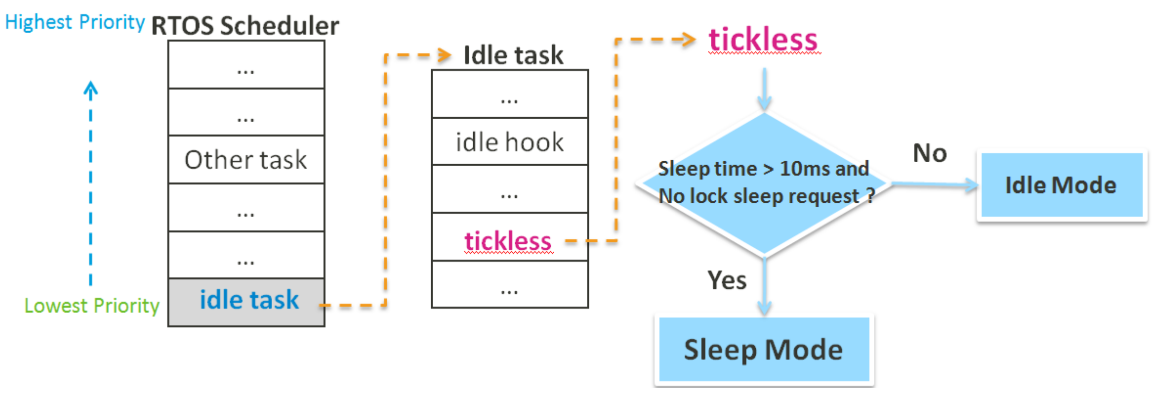

Figure 2. AB1565 FreeRTOS tickless feature flow

3.2.1. Debugging limitations in FreeRTOS low-power mode

The MCU (Cortex-M4) is gated when the system goes into Sleep mode from the FreeRTOS idle task. The debugging identification information is stored in the Cortex-M4 ROM table and cannot be accessed when a debugging tool is attached to the AB1565 HDK through a Keil IDE or an IAR workbench IDE. Therefore, all resources for CPU debugging are busy.
Because of this limitation, the Sleep mode must be disabled to allow debugging. To disable the Sleep mode:

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

4. Power Modes and Current Measurement for AB1568

This section introduces the MCU system's power mode and low-power configuration.

- Power mode for the MCU is described in Section 4.1. "Power mode".
- The FreeRTOS low-power feature is described in Section 4.2 "Enabling the FreeRTOS low-power mode".

4.1. Power mode

A detailed description of how to switch the power mode can be found in the Sleep Manager module of the API Reference Manual under `<sdk_root>/mcu/doc`. The power modes for AB1568 are summarized below and in Table 3.

- Active
  - o The CPU and RAM are in an active state. Instructions can be executed and the peripheral access is active.
- Idle
  - o The CPU goes into a clock-gated state to save core power. Internal and external interrupts can wake up the processor.
  - o RAM is in an idle state.
- Sleep
  - o CPU power is turned off to save more power.
  - o RAM go into the lowest power state and the data is preserved.
  - o All peripherals including I2C, UART, and SPI are powered off.
- RTC
  - o Only RTC power is retained.
  - o RAM goes into a power-down state and data is lost.

Table 3. The power modes for AB1568

Mode	Description	MCU clock source	V_{CORE} voltage minimum value	RAM	FLASH	Peripherals
High Speed	The maximum frequency of CM4 is 208MHz. DSP is 416MHz.	LPOSC	0.9V	Active	Active	Active
Full Speed	The maximum frequency of CM4 is 104MHz. DSP is 208MHz.	LPOSC	0.8V			
Low Speed	The maximum frequency of CM4 is 52MHz. DSP is 104MHz.	LPOSC	0.75V			

Mode	Description	MCU clock source	V_{CORE} voltage minimum value	RAM	FLASH	Peripherals
Idle	The CM4/DSP clock is gated to save core power.	Not available	Same as active state	Data kept	Idle	Active
Sleep	The CM4/DSP power is turned off to achieve lower current consumption.	Not available	0.75V	Data kept	Deep Power Down	Power Off
RTC	Only RTC power is retained.	Not available	Not available	Power Off (Data Lost)	Power Off	Power Off
Off	-	Not available	Not available	Power Off (Data Lost)	Power Off	Power Off

4.2. Enabling the FreeRTOS low-power mode

FreeRTOS provides a tick suppression option to enable lower power consumption when 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 the tickless feature by using the setting "`#define configUSE_TICKLESS_IDLE 2`".

There are two conditions in the tickless porting files (*) that are verified before going into the low-power state. The first condition is that no sleep lock is held by users (**). The second condition is to check if the sleep time is more than 10ms.

- The system goes into the **Sleep** state if the sleep time is more than 10ms.
  - To prevent the system from going into the **Sleep** state, the Sleep Manager module provides functions to lock or unlock the sleep mode. Please refer to the API Reference Manual under `<sdk_root>/doc` for more information.
- The system goes into the **Idle** state if the sleep time is less than 10ms.

Figure 3. AB1568 FreeRTOS tickless feature flow shows the tickless process for AB1568.

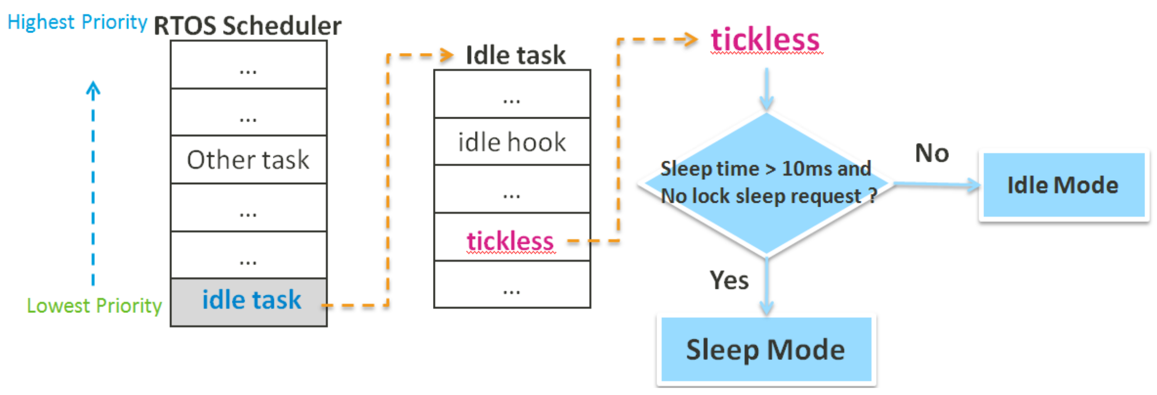

Figure 3. AB1568 FreeRTOS tickless feature flow

4.2.1. Debugging limitations in FreeRTOS low-power mode

The MCU (Cortex-M4) is gated when the system goes into Sleep mode from the FreeRTOS idle task. The debugging identification information is stored in the Cortex-M4 ROM table and cannot be accessed when a debugging tool is attached to the AB1568 HDK through a Keil IDE or an IAR workbench IDE. Therefore, all resources for CPU debugging are busy.
Because of this limitation, the Sleep mode must be disabled to allow debugging. To disable the Sleep mode:

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

5. 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.

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

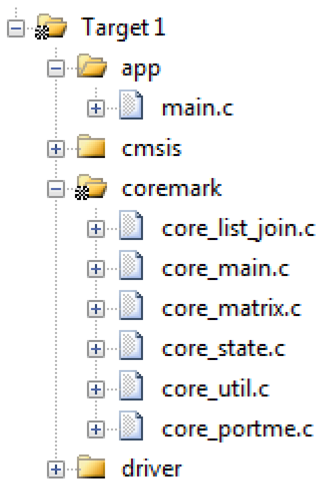

Figure 4. An example project with CoreMark source files on µVision IDE

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.

5.2. 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

5.3. 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.