USER'S GUIDE

FreeRTOS9 on AmbiqSuite SDK

Ultra-Low Power Apollo MCU Family

A-MCUAP3-UGGA05EN v1.0

Legal Information and Disclaimers

AMBIQ MICRO INTENDS FOR THE CONTENT CONTAINED IN THE DOCUMENT TO BE ACCURATE AND RELIABLE. THIS CONTENT MAY, HOWEVER, CONTAIN TECHNICAL INACCURACIES, TYPOGRAPHICAL ERRORS OR OTHER MISTAKES. AMBIQ MICRO MAY MAKE CORRECTIONS OR OTHER CHANGES TO THIS CONTENT AT ANY TIME. AMBIQ MICRO AND ITS SUPPLIERS RESERVE THE RIGHT TO MAKE CORRECTIONS, MODIFICATIONS, ENHANCEMENTS, IMPROVEMENTS AND OTHER CHANGES TO ITS PRODUCTS, PROGRAMS AND SERVICES AT ANY TIME OR TO DISCONTINUE ANY PRODUCTS, PROGRAMS, OR SERVICES WITHOUT NOTICE.

THE CONTENT IN THIS DOCUMENT IS PROVIDED "AS IS". AMBIQ MICRO AND ITS RESPECTIVE SUPPLIERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THIS CONTENT FOR ANY PURPOSE AND DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THIS CONTENT, INCLUDING BUT NOT LIMITED TO, ALL IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHT.

AMBIQ MICRO DOES NOT WARRANT OR REPRESENT THAT ANY LICENSE, EITHER EXPRESS OR IMPLIED, IS GRANTED UNDER ANY PATENT RIGHT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT OF AMBIQ MICRO COVERING OR RELATING TO THIS CONTENT OR ANY COMBINATION, MACHINE, OR PROCESS TO WHICH THIS CONTENT RELATE OR WITH WHICH THIS CONTENT MAY BE USED.

USE OF THE INFORMATION IN THIS DOCUMENT MAY REQUIRE A LICENSE FROM A THIRD PARTY UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF THAT THIRD PARTY, OR A LICENSE FROM AMBIQ MICRO UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF AMBIQ MICRO.

INFORMATION IN THIS DOCUMENT IS PROVIDED SOLELY TO ENABLE SYSTEM AND SOFTWARE IMPLEMENTERS TO USE AMBIQ MICRO PRODUCTS. THERE ARE NO EXPRESS OR IMPLIED COPYRIGHT LICENSES GRANTED HEREUNDER TO DESIGN OR FABRICATE ANY INTEGRATED CIRCUITS OR INTEGRATED CIRCUITS BASED ON THE INFORMATION IN THIS DOCUMENT. AMBIQ MICRO RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN. AMBIQ MICRO MAKES NO WARRANTY, REPRESENTATION OR GUARANTEE REGARDING THE SUITABILITY OF ITS PRODUCTS FOR ANY PARTICULAR PURPOSE, NOR DOES AMBIQ MICRO ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT, AND SPECIFICALLY DISCLAIMS ANY AND ALL LIABILITY, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL OR INCIDENTAL DAMAGES. "TYPICAL" PARAMETERS WHICH MAY BE PROVIDED IN AMBIQ MICRO DATA SHEETS AND/OR SPECIFICATIONS CAN AND DO VARY IN DIFFERENT APPLICATIONS AND ACTUAL PERFORMANCE MAY VARY OVER TIME. ALL OPERATING PARAMETERS, INCLUDING "TYPICALS" MUST BE VALIDATED FOR EACH CUSTOMER APPLICATION BY CUSTOMER'S TECHNICAL EXPERTS. AMBIQ MICRO DOES NOT CONVEY ANY LICENSE UNDER NEITHER ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS. AMBIQ MICRO PRODUCTS ARE NOT DESIGNED, INTENDED, OR AUTHORIZED FOR USE AS COMPONENTS IN SYSTEMS INTENDED FOR SURGICAL IMPLANT INTO THE BODY, OR OTHER APPLICATIONS INTENDED TO SUPPORT OR SUSTAIN LIFE, OR FOR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE AMBIQ MICRO PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR. SHOULD BUYER PURCHASE OR USE AMBIQ MICRO PRODUCTS FOR ANY SUCH UNINTENDED OR UNAUTHORIZED APPLICATION, BUYER SHALL INDEMNIFY AND HOLD AMBIQ MICRO AND ITS OFFICERS, EMPLOYEES, SUBSIDIARIES, AFFILIATES, AND DISTRIBUTORS HARMLESS AGAINST ALL CLAIMS, COSTS, DAMAGES, AND EXPENSES, AND REASONABLE ATTORNEY FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PERSONAL INJURY OR DEATH ASSOCIATED WITH SUCH UNINTENDED OR UNAUTHORIZED USE, EVEN IF SUCH CLAIM ALLEGES THAT AMBIQ MICRO WAS NEGLIGENT REGARDING THE DESIGN OR MANUFACTURE OF THE PART.

Revision History

Revision	Date	Description
1.0	April 11, 2022	Initial release.

Reference Documents

Document ID	Description

Table of Contents

SECTION

1

Overview

This Application Guide explains the implementation specific aspects of FreeRTOS9 delivered with the AmbiqSuite SDK. The guide covers low power operation with FreeRTOS9 when using Ambiq specific port, and also outlines optional application framework used by the example applications. An overview of the example applications provided with the SDK is included, to enable quick start with FreeRTOS9.

This application guide is intended as a supplement to the extensive documentation already provided with FreeRTOS9. Detailed information about FreeRTOS internals and usage guide can be found at: http://www.freertos.org/Documentation/RTOS_book.html

Low power operation with FreeRTOS is detailed at:
http://www.freertos.org/low-power-tickless-rtos.html

SECTION

2

Low Power Operation with FreeRTOS9

During periods of prolonged inactivity, it is optimal to place the microcontroller into a low power state for power saving. By default, the FreeRTOS9 port on Arm M4 microcontrollers relies on SYSTICK interrupt to track time. Hence, even during idle times, it is necessary to periodically exit and then re-enter the low power state to process tick interrupts. This could severely limit the efficiency, depending on the frequency of the tick interrupt.

FreeRTOS provides mechanisms to override this default behavior for better power efficiency. This section lists the FreeRTOS options geared for power saving.

2.1 Tick Management

Tick Management defines how the internal ticks are driven/managed. These are controlled by two flags in **FreeRTOSConfig.h**.

- configOVERRIDE_DEFAULT_TICK_CONFIGURATION
  - 0: Rely on ARM SYSTICK (Default implementation) as described above
  - 1: Rely on custom implementation.
- configUSE_TICKLESS_IDLE
  - 0: Use a periodic tick interrupt regardless of the other activities, and hence suffers from the need to periodically wake up the code to service the tick interrupts
  - 1: FreeRTOS tickless idle mode stops the periodic tick interrupt during idle periods (when there are no application tasks that are able to execute), then makes a correcting adjustment to the RTOS tick count value when the tick interrupt is restarted. Stopping the tick interrupt allows the microcontroller to remain in a power saving state until either an interrupt occurs, or it is time for the RTOS kernel to transition a task into the Ready state.
  - 2: Rely on custom implementation.

The **freertos_lowpower** example sets these configuration parameters as follows:

```
#define configOVERRIDE_DEFAULT_TICK_CONFIGURATION 1 // Enable non-SysTick
based Tick
#define configUSE_TICKLESS_IDLE 2 // Ambiq specific implementation for Tick-
less
```

2.2 Idle Implementation

The default implementation of the Idle Task in FreeRTOS invokes a WFI after doing the required maintenance operations. FreeRTOS provides following hooks to implement low power specific actions when idling.

- configPRE_SLEEP_PROCESSING
  - Should be used to implement any actions prior to, and optionally including getting into WFI (e.g., saving necessary state information and powering down the peripherals).
  - When using this function implementing call to WFI inside, the function should always return 0.
  - If WFI is not invoked from this function, it should return the same value as passed in as idleTime. In this case, the FreeRTOS9 implementation would invoke WFI
- configPOST_SLEEP_PROCESSING
  - Should be used to implement actions needed to recover after getting back to active state (e.g., powering the peripherals back up, and restoring internal state).

The freertos_lowpower example implements a call to:

```
am_hal_sysctrl_sleep(AM_HAL_SYSCTRL_SLEEP_DEEP);
```

For the configPRE_SLEEP_PROCESSING function.

SECTION

3

Tick Management with FreeRTOS9 Port for Apollo/ Apollo2

Default Ambiq port of FreeRTOS9 builds upon the hooks provided for extending the low power operation modes, and provides following options for FreeRTOS9 tick management. Three possible modes possible as described below. Other combinations are invalid.

3.1 Default FreeRTOS Mode

This mode is elected by setting the following:

- **configOVERRIDE_DEFAULT_TICK_CONFIGURATION = 0**
- **configUSE_TICKLESS_IDLE = 0**

This is the native mode provided by FreeRTOS9. This implementation uses Arm SysTick interrupts to keep track of time. When in this mode, core will need to keep servicing the high frequency SysTick interrupts even when idling. Core should not be powered down when using this setting.

3.2 FreeRTOS Tickless Idle Mode

This mode is elected by setting the following:

- **configOVERRIDE_DEFAULT_TICK_CONFIGURATION = 0**
- **configUSE_TICKLESS_IDLE = 1**

This is the Tickless Idle mode provided by FreeRTOS. This implementation also uses ARM SysTick interrupts to keep track of time. However, it reprograms the SysTick interrupts for less interruptions when idling to reduce the core wake-ups for power efficiency. Core should not be put in Deep Sleep mode when using this setting as the SysTick timer needs to keep running.

3.3 Ambiq Tickless Idle Mode

This mode is elected by setting the following:

- **configOVERRIDE_DEFAULT_TICK_CONFIGURATION = 1**
- **configUSE_TICKLESS_IDLE = 2**

This is Ambiq specific implementation geared towards lowest power usage with FreeRTOS9. This implementation relies on an external timer instead of Arm SysTick to keep track of time - allowing the core to be completely powered down when idling.

> **NOTE:** The Core can be put in Deep Sleep mode when using this setting.

This mode provides further configurability through flag **AM_FREERTOS_USE_STIMER_FOR_TICK** in **FreeRTOSConfig.h**.

- **AM_FREERTOS_USE_STIMER_FOR_TICK** defined
  - Uses STimer for FreeRTOS9 Tick
  - This mode is only available in Apollo2, which supports the Stimer
  - Stimer is started from, and managed by the porting code, and is not available for reuse for other purpose
    - Limited reuse of the Stimer is still possible by applications, as long as they do not reconfigure the Stimer. An example of the same is provided under **USE_STIMER_FOR_WSF in freertos_fit** example.
- **AM_FREERTOS_USE_STIMER_FOR_TICK** not defined
  - Uses CTimer3 for FreeRTOS9 Tick
  - This is the only mode available in Apollo
  - CTimer3 is started from and managed by the porting code, and is not available for reuse for other purpose

    > **NOTE:** CTIMER3 is tied into the ADC sampling in Apollo2. If this feature is desired, then **AM_FREERTOS_CTIMER_NUM** should be modified to use another Ctimer.

  - Application needs to implement generic Ctimer ISR handler (**am_ctimer_isr)** to service the registered interrupts by calling **am_hal_ctimer_int_service()**
    - Sample code can be found in example applications rtos.c
- Other Ctimers can be used by application

SECTION

4

Idle Implementation with FreeRTOS9 for Apollo and Apollo2

Default Ambiq port of FreeRTOS9 does not override the definition of **configPRE_SLEEP_PROCESSING** / **configPOST_SLEEP_PROCESSING** hooks.

The optional Application Framework used by all the sample examples does provide default implementation for these as described in following section.

SECTION

5

Sample Applications Framework

Ambiq provided sample applications follow a uniform framework, which could also be re-used by application writers if they chose so. This section details organization of this framework.

- File FreeRTOSConfig.h contains various FreeRTOS9 customization options
  Refer to http://www.freertos.org/a00110.html for more details
- File rtos.c contains:
  - Debug Hooks
    Default implementation of FreeRTOS provided debug hooks **vApplicationMallocFailedHook()** and **vApplicationStackOverflowHook()** are provided in for example applications.
  - Low Power function hooks
    The default **FreeRTOSConfig.h** maps FreeRTOS provided hooks: **configPRE_SLEEP_PROCESSING** and **configPOST_SLEEP_PROCESSING** to the following functions, which could be implemented to applications' choosing
    - **am_freertos_sleep()** - should be used to implement any actions prior to, and optionally including getting into WFI, (e.g., saving necessary state information & powering down the peripherals).
    - Ideally, the last action in the function should be to call the HAL call for getting into appropriate low power mode.
    - **am_hal_sysctrl_sleep(AM_HAL_SYSCTRL_SLEEP_DEEP)** OR **am_hal_sysctrl_sleep(AM_HAL_SYSCTRL_SLEEP_NORMAL)**

> **NOTE: AM_HAL_SYSCTRL_SLEEP_DEEP** mode powers down the core and hence is only possible when using Ambiq Tickless Idle Mode (**configUSE_TICKLESS_IDLE = 2**).

        - When using this HAL function or directly invoking WFI inside this function, the function should always return 0.
        - If WFI is not invoked from this function, it should return the same value as passed in as idleTime. In this case, the FreeRTOS implementation would invoke WFI
        - **am_freertos_wakeup()** - should be used to implement actions needed to recover after getting back to active state, e.g. powering the peripherals back up, and restoring internal state

- Default implementation of **am_ctimer_isr()**
  - This is important when using **AM_FREERTOS_USE_STIMER_FOR_TICK = 0**, along with **configOVERRIDE_DEFAULT_TICK_CONFIGURATION = 1** and **configUSE_TICKLESS_IDLE = 2**

> **NOTE:** The Ctimers share the **am_ctimer_isr** function, so the default implementation includes the call to **am_hal_ctimer_int_service** with the interrupt information so that Ctimer interrupts are processed for the application.

- Task Setup framework
  - **run_tasks()**
    - This function is called from the **main()** of the application after it finishes all the required non-FreeRTOS setup operations.
    - This function is not expected to return
    - This function creates a single task **setup_task** and starts the FreeRTOS Task scheduler.
  - **setup_task()**
    - This function provides a place for any specific initializations that can only happen after the FreeRTOS scheduler has been started.
    - After the required setup operations, it should create all the application specific tasks and suspend itself.

The main application makes a single call to **run_tasks()** to start the FreeRTOS tasks. This function never returns.

SECTION

6 Additional Notes for Application Developers

6.1 Interrupt Priority

Listed below are some important things to consider when setting up the interrupt priorities. Detailed information is available at http://www.freertos.org/RTOS-Cortex-M3-M4.html

- Arm Cortex-M4 ports use numerically lower values to represent logically higher priority levels.
- Cortex-M interrupts default to having a priority value of zero. Zero is the highest possible priority value. Therefore, never leave the priority of an interrupt that uses the interrupt safe RTOS API at its default value.
- Any interrupt that uses the FreeRTOS API (API functions that end in "FromISR") must be set to a priority value numerically at or above the RTOS kernel (as configured by the **configKERNEL_INTERRUPT_PRIORITY** macro), but lowest numerical value cannot be lower than **configMAX_SYSCALL_INTERRUPT_PRIORITY**.
- Arm Cortex-M core stores interrupt priority values in the most significant bits of its eight bit interrupt priority registers. The **configMAX_SYSCALL_INTERRUPT_PRIORITY** and **configKERNEL_INTERRUPT_PRIORITY** settings found in FreeRTOSConfig.h require their priority values to be specified as the Arm Cortex-M core itself wants them - already shifted to the most significant bits of the byte.
- **configMAX_SYSCALL_INTERRUPT_PRIORITY** must not be set to 0

6.2 Ctimer/Stimer Usage

As explained above, the use of Ctimer3/Stimer by the applications may be restricted, depending on the Tickless mode being used.

6.3 Implementation of am_freertos_sleep()

As explained above, applications can implement this function to optionally include calls to **am_hal_sysctrl_sleep()**. **AM_HAL_SYSCTRL_SLEEP_DEEP** is possible only when using Ambiq Tickless Idle Mode (**configOVERRIDE_DEFAULT_TICK_-CONFIGURATION = 1**, **configUSE_TICKLESS_IDLE = 2**).

SECTION

7

Example Application in SDK

Ambiq SDK provides various examples demonstrating FreeRTOS usage.

- **freertos_sampler**
  - This application has been written to demonstrate various FreeRTOS features, and has been described in detail in next section.
- **freertos_lowpower**
  - This example implements LED task within the FreeRTOS framework.
  - It monitors three On-board buttons, and toggles respective on-board LEDs in response.
  - When Idle, it puts the core in deep sleep mode.
- **freertos_fit**
  - This example uses Cordio BLE stack andthen invokes the Fit profile therein, implemented on FreeRTOS, using the Dialog BLE daughter card for Ambiq EVK.
  - By default, it uses FreeRTOS Timer for implementing WSF ticks.
  - As a demonstration, it can be compiled to instead use external timers (either Ctimer1, or in case of Apollo2 reusing Stimer used for implementing FreeRTOS ticks)

7.1 FreeRTOS Sampler Example

This Ambiq Micro Demo program shows several FreeRTOS API structures and how to use them in a simple application. This application demo shows the use of Event Groups, TaskNotify, Queues, Interrupts and Tickless Operation in Low Power Sleep Functions. Placing the Apollo / Apollo_2 processors into a Sleep mode reduces the Average power with a single Timer running. Further power reductions can be achieved by turning off all clocks and power domains. A single Timer must be kept active for FreeRTOS, to be able to keep Real Time Ticks available and current. Three interrupts are setup to test the low power idle modes ability to keep track of System Tick time in a typical Real Time System.

STimer->CMP-0, UART1, GPIO24, GPIO26 and ITM peripherals are left powered on.

7.1.1 ISR_Table

Table 7-1: ISR_Table

Peripheral	Interrupt Function Name	Demo Function
UART1	freertos_sampler.c ->am_uart1_isr()	Captures serial data to a circular buffer Pushes an event message to SERQueue after each byte received
Stimer	port.c ->xPortStimerTickHandler()[1]	Used for FreeRTOS Tick functions Called from am_s**timer_cmpr0_isr()**
Ctimer	port.c ->xPortCTimer0TickHandler()[1]	Used for FreeRTOS Tick functions. Called from **am_ctimer_isr()**
GPIO24 (BTN2)	button_task.c ->am_gpio_isr()	Captures BTN2 presses, Toggles LED 1 Calls **button_task.c->button1_handler()**
GPIO26 (BTN3)	button_task.c ->am_gpio_isr()	Captures BTN3 presses, Toggles LED 2 Calls **button_task.c->button2_handler()**

[1] Only one timer will be used at a time.

BTN2 and BTN3 handlers trigger an **xEventGroupSetBitsFromISR()** to **Button_task.c->ButtonTask()**

7.2 Task Function

Each task uses a different FreeRTOS API function to pass data from its interrupt to the task. Each FreeRTOS function has a different internal delay due to the internal structures involved.

Table 7-2: Task Function

Task Created	Description	Free RTOS APIs Demonstrated
TaskDelayTask()	Cycles through Task Delay's	TaskDelayUntil
SerialTask()	Handles UART1 Receive interrupts	Queue
ButtonTask()	Handles BTN1 and BTN2 interrupts	Event Groups
AppTask()	Handles Timer interrupts	TaskNotify
ITMTask()	Prints ITM messages to the Debugger	Queue
prvIdleTask()	Used by FreeRTOS for resource cleanup and enable sleep functions.	This is the Idle Task implementation in FreeRTOS

7.2.1 TaskDelayTask

Cycles through four **VTaskDelayUntil()** calls to show off the Sleep functions. As each Delay is called, the IdleTask() will call **port.c->vPortSuppressTicksAnd-Sleep()** to place the processor into sleep mode. Each task delay call uses a delay time in System Ticks. Each tick delay is multiplied by 32 timer counts and programs the selected timer. The four task delay calls use 31, 8, 55 and 24000 ticks to place the processor into sleep mode for different times. The 24000 tick delay is use to verify a sleep delay greater than a 16-bit timer count.

7.2.2 SerialTask

The serial task is linked to the **am_uart1_isr()** with the **SERQueueElement**. Each byte received by uart1 triggers an interrupt service routine to retrieve that byte and places it into a circular buffer. The interrupt service routine sends a Queue message via the **SERQueueElement->RTOS_event**. When the scheduler runs again, the Queue will pass the **RTOS_event** to the **SerialTask()**. The **SerialTask()** will read and decode the **RTOS_event** from the **SERQueueElement** buffer. The decoded **RTOS_event** will call **serial_handleer()** to print a message to the ITM debug port.

7.2.3 ButtonTask

ButtonTaskSetup() registers the GPIO26 and GPIO24 to hardware interrupt handlers **button1_hander()** and **button2_handler** respectively.

When a Button is pressed, the hardware will vector to **am_gpio_isr()** and then pass the **gpio_register** status to **am_hal_gpio_int_service(status)**. This status contains the gpio number index into the **am_hal_gpio_ppfnHandlers[x]** array. The correct handler will be call to service its respective gpio pin.

The registered **button[x]_handler()** will de-bounce the input gpio pin for 20 mSecs before passing to the function **button_handler(x)**.

The **button_handler(x)** will register an event to the **xButtonEventHandle** - EventGroup and yield the scheduler from the ISR.

The **ButtonTask()** will be made active to service the EventGroup and decode the button pressed.

The bitSet in **gpio_register_status** will decode and toggle LED17 and LED18 respectively.

7.2.4 AppTask

This task uses the **TaskNotify()** API to communicate with a timer interrupt. A simple **ulTaskNotifyTake()** function halts the **ApTask()** until a **xTaskNotifyGive(xAppTask)** is executed.

7.2.5 ITMTask

This task is used to print user messages on the ITM port.

A queue is used to pass messages to the **ITMTask()** and those messages are sent to the ITM Port.

At program startup i.e. main() the function **Freertos_sampler.c**->**enable_itm_print(…)** is called to register and enable the ITM/SWO debug output. An API call to **am_util_stdio_printf_init** (**am_bsp_itm_string_print**); will route **stdio printf** calls to the ITM port. Any messages can be printed to a buffer then that buffer can be sent to the ITM port via a call to **print_via_itm_task(pui32Temp)**. **print_via_itm_task(pui32Temp)** will buffer the message to a queue and then to **ITMQueue_send(&itm_msg);** for printing on the ITM/SWO debug port.

7.2.6 IDLE Task

This task is part of core FreeRTOS, and described here just for sake of completeness.

This task is created by **Tasks.c->vTaskStartScheduler()** before the scheduler is started. FreeRTOS will return to the IDLE task when no tasks are in the Ready state.

If **configUSE_TICKLESS_IDLE** is defined and there are no tasks ready to run, IDLE task calls **prvGetExpectedIdleTime()** to check if the **xExpectedIdleTime** is greater than or equal to two (Ticks).

portSUPPRESS_TICKS_AND_SLEEP() performs one last check to **eTaskConfirmSleepModeStatus()** to be sure the sleep function has not be aborted by another task. The processor global interrupts are turned off and the selected timer is programmed to the **xExpectedIdleTime** and the processor is put to sleep.

6500 River Place Boulevard, Building 7, Suite 200, Austin, TX 78730
www.ambiq.com
sales@ambiq.com
+1 (512) 879-2850

A-MCUAP3-UGGA05EN v1.0
April 2022