APPLICATION NOTE

Atmel AT03289: SAM4L Low Power Design with FreeRTOS

Atmel 32-bit Microcontrollers

Introduction

The Atmel® SAM4L family of microcontrollers is based on the ARM® Cortex®-M4 CPU architecture. The SAM4L series embeds the state-of-the-art Atmel picoPower® technology for ultra-low power consumption. The device allows a wide range of options between functionality and power consumption, giving the user the ability to reach the lowest possible power consumption with the feature set required for the application. SAM4L is designed specifically for use in applications that require extremely low power consumption.

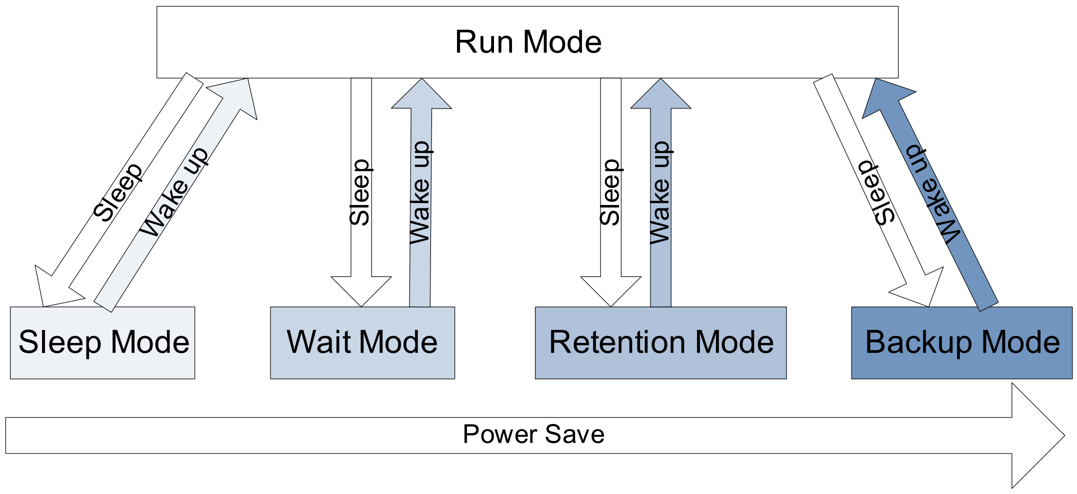
The purpose of this document is to introduce various low power modes of the SAM4L family and demonstrate how to do low power design with FreeRTOS™ tick suppression feature on SAM4L device.

Table of Contents

1. Low Power Modes in SAM4L

The Atmel SAM4L family of microcontrollers has various power modes, including Run mode and different low power modes (Sleep, Wait, Retention, and Backup) [1]. Various power modes are shown in Figure 1-1. Each low power mode can be entered from Run mode and woken up by related enabled wake-up interrupt or event. The various low power modes are described below.

Figure 1-1. SAM4L Low Power Modes.



1.2 Sleep Mode

The Sleep mode allows power optimization with the fastest wake up time. In this mode, the CPU clock is stopped. To further reduce power consumption, modules clocks and synchronous clock sources can be switched off through the BPM.PMCON.SLEEP field. Table 1-1 shows a summary configuration of four sub-modes of Sleep mode.

Table 1-1. Sleep Mode Configuration.

BPM.PSAVE.SLEEP	CPU clock	AHB clocks	APB clocks GCLK	Clock sources: OSC, RCFAST, RC80M, PLL, DFLL	RCSYS	OSC32K RC32K	Wake up Sources
0	Stop	Run	Run	Run	Run	Run	Any interrupt
1	Stop	Stop	Run	Run	Run	Run	Any interrupt
2	Stop	Stop	Stop	Run	Run	Run	Any interrupt
3	Stop	Stop	Stop	Stop	Run	Run	Any interrupt

1.3 Wait Mode

The Wait mode allows achieving very low power consumption while maintaining the Core domain powered-on. All clock sources are stopped, and the core and all the peripherals are stopped except the modules running with the 32kHz clock if enabled. This is the lowest power configuration where SleepWalking is supported.

1.4 Retention Mode

The Retention mode is similar to Wait mode in terms of clock activity. All clock sources are stopped, and the core and all the peripherals are stopped except the modules running with the 32kHz clock if enabled. This is the lowest power configuration where the logic is retained.

1.5 Backup Mode

The Backup mode allows achieving the lowest power consumption possible in a system, which is performing periodic wake-ups to perform tasks but not requiring fast startup time.

The Core domain is powered-off. The internal SRAM and register contents of the Core domain are lost. The Backup domain is kept powered on. The 32kHz clock (RC32K or OSC32K) is kept running if enabled to feed modules that require clocking.

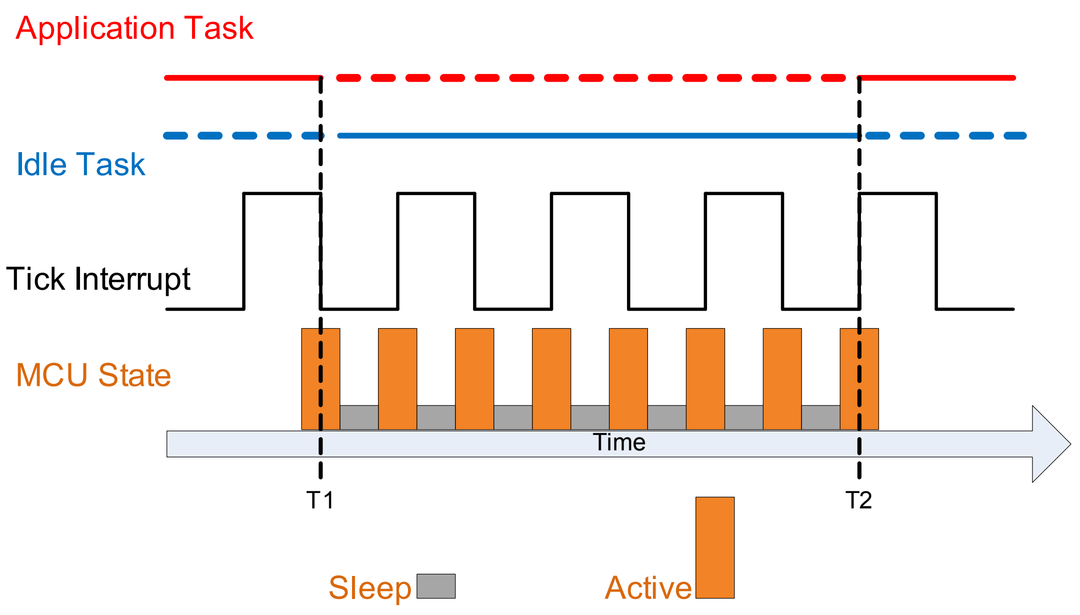
For more details about SAM4L low power modes, refer to Section 6 "Low Power Techniques" in SAM4L Datasheet [1].

2. FreeRTOS Tickless Feature Introduction

FreeRTOS is a light weight, easy-to-use and free Real Time Operating System. Users who are not familiar with FreeRTOS and would like to know more can take a look at FreeRTOS detailed introduction at the FreeRTOS website.

It is common to reduce the power consumed by the microcontroller on which RTOS is running by using the idle task hook to place the microcontroller into a low power state. The power saving that can be achieved by this simple method is limited by the necessity to periodically exit and then re-enter the low power state to process tick interrupts. Further, if the frequency of the tick interrupt is too high, the energy and time consumed entering and then exiting a low power state for every tick will outweigh any potential power saving gains for all but the lightest power saving modes. Figure 2-1 illustrates the low power design for common RTOS. There are no application tasks that are able to run between time T1 and T2, so idle task will run during this time. In order to get low power consumption, the microcontroller will be placed into a low power state, but the microcontroller is woken up periodically by the tick interrupt.

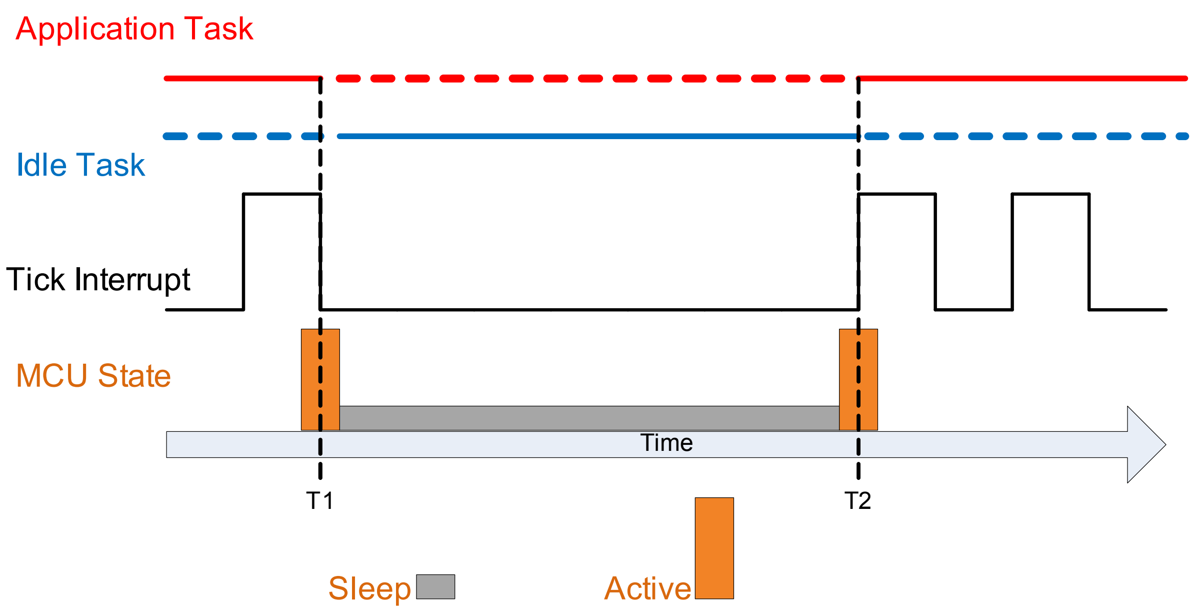
Figure 2-1. Common Way for RTOS to do Low Power Design



The FreeRTOS tickless idle mode stops the periodic tick interrupt during idle periods (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 deep power saving state until either an interrupt occurs, or it is time for the RTOS kernel to transition a task into the Ready state. Figure 2-2 illustrates the FreeRTOS tickles mode to do low power design. The tick interrupt is stopped between time T1 and T2, so the microcontroller can remain in a deep power saving state for a long time and save more power.

Figure 2-2. FreeRTOS Tickless Feature to do Low Power Design



3. FreeRTOS Tickless Feature Implementation

FreeRTOS built in tickless idle functionality is enabled by defining configUSE_TICKLESS_IDLE as 1 in FreeRTOSConfig.h. Note that only V7.3.0 and later version can support tickless feature.

When the tickless idle functionality is enabled the kernel will call the `portSUPPRESS_TICKS_AND_SLEEP`() macro when the following two conditions are both true:

1. The Idle task is the only task able to run because all the application tasks are either in the Blocked state or in the Suspended state.
2. At least n further complete tick periods will pass before the kernel is due to transition an application task out of the Blocked state, where n is set by the configEXPECTED_IDLE_TIME_BEFORE_SLEEP definition in FreeRTOSConfig.h.

The parameter `xExpectedIdleTime` for the function `portSUPPRESS_TICKS_AND_SLEEP(xExpectedIdleTime)` represents the total number of tick periods before a task is due to be moved into the Ready state. The parameter value is therefore the time the microcontroller can safely remain in a deep sleep state, with the tick interrupt stopped (suppressed).

3.2 Tickless Feature Implemented with SysTick

Normally, in FreeRTOS environment, SAM4L uses system timer (SysTick) as system tick. The SysTick is a 24-bit timer, which counts down from the reload value to zero. Using this timer, periodic interrupt can be easily generated for FreeRTOS, and timer management and context switch in FreeRTOS will be done based on it. Figure 3-2 shows the SysTick handler and if configUSE_TICKLESS_IDLE is set to 1, the SysTick load register will be reset.

Figure 3-2. SysTick Handler in FreeRTOS

```
void SysTick_Handler( void )
{
        /* If using preemption, also force a context switch. */
        #if configUSE_PREEMPTION == 1
                portNVIC_INT_CTRL_REG = portNVIC_PENDSVSET_BIT;
        #endif

        /* If using tickless mode, reset the systick load register. */
        #if configUSE_TICKLESS_IDLE == 1
                portNVIC_SYSTICK_LOAD_REG = ulTimerReloadValueForOneTick;
        #endif

        ( void ) portSET_INTERRUPT_MASK_FROM_ISR();
        {
                vTaskIncrementTick();
        }
        portCLEAR_INTERRUPT_MASK_FROM_ISR( 0 );
}
```

The other one function `vPortSuppressTicksAndSleep()` should be implemented if tickless feature is used. This function has a single parameter that equals to the total number of ticks period before a task is due to be moved into Ready state. There is much work to do in this function, such as, recalculating the allowed sleep time, re-configuring the SysTick timer according to sleep time, entering low power mode, calculating how much time the MCU has slept after woken up and stepping forward the ticks. Fortunately, this function has been implemented by FreeRTOS. The only thing needed to do is to enable tickless mode by setting `configUSE_TICKLESS_IDLE` to 1.

The MCU will enter low power mode automatically if there are no application tasks running and the least block time of the application tasks is not less than `configEXPECTED_IDLE_TIME_BEFORE_SLEEP` which's definition is in FreeRTOSConfig.h.

However, there is a potential risk that perhaps the MCU can't be woken up if it enters some deeper low power mode e.g. Wait mode, Retention mode, etc. SysTick timer should generate a wake-up signal to the MCU when the sleep time is over, but the SysTick may not work at all in deeper low power mode. Taking a look at the datasheet, SysTick timer runs on the processor clock, and if the clock is stopped, the SysTick timer stops. In this case, the MCU will remain in low power state all the time and will never be woken up by the SysTick timer.

FreeRTOS tickless feature implemented with SysTick timer can't save much power since MCU can't enter deeper low power mode.

3.3 Tickless Feature Implemented with AST

Asynchronous timer (AST) can run in any low power mode, even in the backup mode as long as its selected clock source is running. Using AST as FreeRTOS tick timer, MCU can enter any deeper low power mode and doesn't need to worry about that it can't be woken up.

AST has a 32-bit counter, which increments every 0-to-1 transition of the selected prescaler tapping, and it will generate an interrupt request when the counter value matches the selected alarm value if the alarm interrupt is enabled. Using AST alarm interrupt, periodic interrupt with different interval can be easily generated for FreeRTOS.

3.3.1 Setup AST Timer

Enable clock source, enable interrupt with proper priority, and set alarm value to generate a tick with regular period should be done before using AST for FreeRTOS.

- Enable AST clock source

  A number of clocks can be selected as AST source, but only the 32kHz clock can keep running in any low power mode. The 32kHz oscillator must be enabled if external oscillator is used.

```
/* Ensure the 32KHz oscillator is enabled. */
if( osc_is_ready( OSC_ID_OSC32 ) == pdFALSE )
{
        osc_enable( OSC_ID_OSC32 );
        osc_wait_ready( OSC_ID_OSC32 );
}
```

- Enable AST interrupt with proper priority

  Since AST is used as tick interrupt, AST interrupt must execute at the lowest interrupt priority configLIBRARY_LOWEST_INTERRUPT_PRIORITY which's definition is in FreeRTOSConfig.h.

```
/* Tick interrupt MUST execute at the lowest interrupt priority. */
NVIC_SetPriority( AST_ALARM_IRQn, configLIBRARY_LOWEST_INTERRUPT_PRIORITY);
ast_enable_interrupt( AST, AST_INTERRUPT_ALARM );
NVIC_ClearPendingIRQ( AST_ALARM_IRQn );
NVIC_EnableIRQ( AST_ALARM_IRQn );
```

- Set alarm value to generate a tick with regular period

  FreeRTOS tick rate is defined by macro `configTICK_RATE_HZ` which definition is in FreeRTOSConfig.h. Set alarm register with a proper value according to `configTICK_RATE_HZ` to generate a tick with regular period.

```
/* Start with the tick active and generating a tick with regular period. */
ast_write_alarm0_value( AST, ulAlarmValueForOneTick );
ast_write_counter_value( AST, 0 );
```

3.3.2 Implement AST Interrupt Handler

The AST interrupt handler is similar to SysTick's except that one flag should be set for the AST interrupt handler. This flag will be used to identify which interrupt brings out of the MCU from low power state in function `vPortSuppressTicksAndSleep()`.

Figure 3-3. AST interrupt handler in FreeRTOS

```
void AST_ALARM_Handler(void)
{
        /* If using preemption, also force a context switch. */
        #if configUSE_PREEMPTION == 1
        {
                portNVIC_INT_CTRL_REG = portNVIC_PENDSVSET_BIT;
        }
        #endif
        ( void ) portSET_INTERRUPT_MASK_FROM_ISR();
        {
                vTaskIncrementTick();
        }
        portCLEAR_INTERRUPT_MASK_FROM_ISR( 0 );

        /* The CPU woke because of a tick. */
        ulTickFlag = pdTRUE;

        ast_write_alarm0_value( AST, ulAlarmValueForOneTick );
        /* Ensure the interrupt is clear before exiting */
        ast_clear_interrupt_flag( AST, AST_INTERRUPT_ALARM );
}
```

3.3.3 Implement Tick Suppression Function

The tick suppress function `vPortSuppressTicksAndSleep(xExpectedIdleTime)` has a single parameter that equals to number of ticks that is allowed to sleep. AST will use this parameter to set its alarm value and bring the MCU out of low power state after the allowed sleep time.

- Calculate the AST alarm value

```
/* Calculate the alarm value required to wait xExpectedIdleTime tick periods. */
ulAlarmValue = ( ulAlarmValueForOneTick * ( xExpectedIdleTime - 1UL ) );
```

  The variable `xExpectedIdleTime` is the total ticks that is allowed to sleep. -1 is used for time compensation.

```
/* Write alarm value to AST register. */
ast_write_alarm0_value( AST, ulAlarmValue );
```

- Enter low power mode

```
/* Sleep until something happens. */
sleepmgr_enter_sleep();
```

  When this function is called, the MCU enters low power state. All the functions after `sleepmgr_enter_sleep()` will never be executed until the MCU is woken up by AST alarm interrupt or other interrupts that can wake up the MCU.

- Calculate how much time the MCU slept after woken up

  If the MCU is woken up by AST alarm interrupt, the time the MCU has slept just equals to the number of ticks that is allowed to sleep.

```
ulCompleteTickPeriods = xExpectedIdleTime - 1UL;
```

  If the MCU is woken up by other interrupts, the time the MCU has slept will be less than the number of ticks that is allowed to sleep.

```
/*Calculate the actual time the mcu has slept.*/
ulCompleteTickPeriods = ast_read_counter_value( AST ) / ulAlarmValueForOneTick;
```

- Step the tick forward by the number of ticks that the MCU has slept

```
vTaskStepTick( ulCompleteTickPeriods );
```

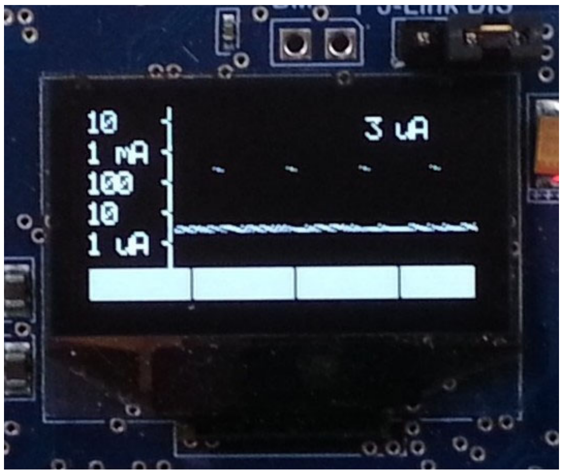
These are the main work that should be done in function `vPortSuppressTicksAndSleep()`. Some details, e.g. time compensation, AST overflow check, etc., have not been referred here but should take note.

3.3.4 Low Power Demo with Tickless Features via AST

The low power demo will demonstrate how much power the MCU will cost when uses FreeRTOS tick suppression features to minimize the power consumption of an application running on SAM4L-EK [2].

The MCU spends most of its time in the Retention low power state. Every 500ms the MCU will come out from the low power state to turn on the LED, then return to low power state for 20ms, before leaving the low power state again to turn off the LED. This will be observed as a rapid blip on the LED every 500ms and the current consumption will be displayed by the board monitor that is built onto the SAM4L-EK.

Figure 3-4. **Current Consumption Displayed on the Board Monitor**



For optimum low power results the SAM4L-EK must be connected and powered using only the JTAG USB connector, but the debugger must not be connected.

3.4 Tickless Feature Implemented with AST and SysTick

The system with FreeRTOS tick suppression feature can run well by using AST, and MCU can enter any low power state and not worry about that it can't be woken up. MCU can remain in low power state for any time as long as it doesn't overflow the AST counter.

However, if the AST calendar mode is used by application, FreeRTOS tickless feature can't be used since it uses AST counter mode. SAM4L only has one AST module which can't run in calendar and counter mode at the same time.

In order to use FreeRTOS tick suppression feature to minimize the power consumption in such an application, both AST and SysTick will be implemented.

FreeRTOS will use SysTick as tick source in MCU active mode, while AST calendar alarm will be used to wake up the MCU in low power state.

The only drawback of this implementation is that the MCU can't sleep for milliseconds period since the minimum interval that calendar alarm can generate is 1s. The MCU can only sleep for 1s, 2s…, but not 10ms, 100ms or 500ms, etc.

3.4.1 Tick Suppression Function Implementation

The timer setup and interrupt handler implementation is most of the same as above section description. Here only the tick suppression function brief description is made.

- Calculate the sleep period

```
ulTimeSeconds = xExpectedIdleTime / configTICK_RATE_HZ;
/* if sleep period less than 1s, do nothing */
if(ulTimeSeconds == 0)
return;
```

- Store the user alarm value and reset alarm according to sleep period

```
/* Store the user alarm value */
ulUserAlarmValue = ast_read_alarm0_value(AST);
/* Set alarm register in time/date format */
ast_set_calendar_alarm(ulTimeSeconds);
```

- Enter low power state

  ```
  /* Sleep until something happens. */
  sleepmgr_enter_sleep();
  ```

- Calculate how much time the MCU slept after woken up

  If the MCU is woken up by AST alarm interrupt, the time the MCU has slept just equals to the number of ticks that is allowed to sleep.

  ```
  ulCompleteTickPeriods = ulTimeSeconds * configTICK_RATE_HZ;
  /* Restore user alarm value */
  ast_write_alarm0_value(AST, ulUserAlarmValue);
  ```

  If the MCU is woken up by other interrupts, the time the MCU has slept will be less than the number of ticks that is allowed to sleep.

  ```
  /*Calculate the actual time the mcu has slept.*/
  ulCompleteTickPeriods = (ulCalValue_after - ulCalValue_before) *
  configTICK_RATE_HZ;
  ```

  Variable `ulCalValue_before` and `ulCalValue_after` are the value of the calendar before MCU enter low power state and after MCU exits low power state respectively.

  ```
  /* Restore user alarm value */
  ast_write_alarm0_value(AST,ulUserAlarmValue);
  ```

- Step the tick forward by the number of ticks that the MCU has slept

  ```
  vTaskStepTick( ulCompleteTickPeriods );
  ```

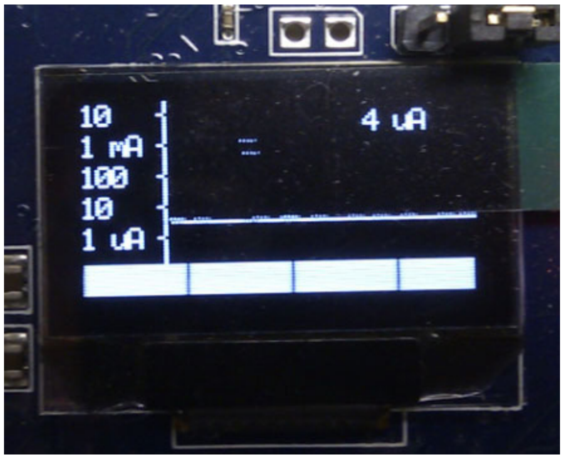
Most of the work in function `vPortSuppressTicksAndSleep()` have been listed here, while a lot of details, such as the time/date format for alarm value, should be taken into account.

3.4.2 Low Power Demo with Tickless Feature via AST and SysTick

The demo is the same as low power demo in Section 3.3.4, except that the MCU will remain in low power state for two seconds.

Every two seconds the MCU will come out from the low power state to turn on the LED, then remain in active for 20ms, after that turn off the LED and enter the low power state again. This will be observed as a rapid blip on the LED every two seconds and the current consumption will be displayed by the board monitor that is built onto the SAM4L-EK.

Figure 3-5. Current Consumption Displayed on the Board Monitor



For optimum low power results the SAM4L-EK must be connected and powered using only the JTAG USB connector, but the debugger must not be connected.

Note that sleep period equal to or long than two seconds is strongly recommended even though the MCU can sleep for one second. The maximum time error is one second, for example, the calculated sleep period is two seconds, but the actual sleep period may be one second due to the minimum interval that calendar alarm can generate. Using `configEXPECTED_IDLE_TIME_BEFORE_SLEEP` which definition is in FreeRTOSConfig.h to block the MCU sleep attempt when the sleep period is less than `configEXPECTED_IDLE_TIME_BEFORE_SLEEP`.

4. Conclusion

This document describes the different low power mode in SAM4L and how to use the AST or the combination of AST and SysTick to implement FreeRTOS tick suppression feature.

In low power design with FreeRTOS tick suppression feature, the key point is that how much time the MCU can remain in low power state. So the best is to consolidate all the application timing functions into a single thread. This thread should make the macro-timing decisions of when to go in and out of low power state (during which time other tasks cannot wake up!) Once this thread is awake, it can dispatch events to other tasks (by means of a semaphore, or message q), to orchestrate the application from timing perspective. Once this thread decides the device should go back into low power state, it should do whatever is necessary to disable tasks from waking up, and then put the device into hibernation.

5. References

[1]. Atmel SAM4L datasheet:

http://www.atmel.com/Images/Atmel-42023-ARM-Microcontroller-ATSAM4L-Low-Power-LCD_Datasheet.pdf

[2]. FreeRTOS SAM4L low power demo:

http://www.freertos.org/Atmel_SAM4L-EK_Low_Power_Tick-less_RTOS_Demo.html

6. Revision History

Doc. Rev.	Date	Comments
42204A	10/2013	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation
1600 Technology Drive
San Jose, CA 95110
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan G.K.
16F Shin-Osaki Kangyo Building
1-6-4 Osaki, Shinagawa-ku
Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370