HighIntegritySystems

WITTENSTEIN

Saving Power Using Your RTOS

How To Save Power Using an RTOS

With increasing customer demand for green, energy efficient products, and the amazing growth in mobile devices where long battery life is highly desirable, engineers now have to consider how best to reduce the power consumed by their designs. This can be partly achieved by selecting components that consume less power, but can the software architecture also contribute to power saving?

This article discusses some easy, but often overlooked, approaches to saving power by using features commonly found in embedded Real Time Operating Systems.

RTOS Scheduling Efficiency

It is a commonly held belief that adding an RTOS to an embedded design will create additional processing overhead, resulting in greater power usage. For simple designs, where a polling 'super loop' architecture may be more appropriate, this could be true. However, for more advanced, complex designs, deploying an RTOS using an event based scheduling algorithm typically reduces the amount of processing time required to run your application, allowing the selection of a smaller processor or adding functionality.

In a software project containing a priority based, pre-emptive RTOS, Tasks are executed in order of priority, i.e the highest priority Task ready to run is always allocated processor time. When there are no Tasks currently available, for example, they are all blocked waiting for events, the RTOS executes a background Task called the Idle Task. The Idle Task may implement some background processing, but typically the system will remain in the Idle Task waiting for an event to bring a Task back to life. The more effectively the software design exploits the RTOS, the greater the time spent in the Idle Task waiting.

Idle Task Sleep Mode

The first, easy, step to save power is to allow the RTOS to place the processor into a low power mode when it enters the Idle Task. To remain responsive, the processor will wake up on the next interrupt. If no external interrupt is triggered, this will be the next RTOS Tick interrupt.

For example, with OPEN**RTOS**® this simple power saving feature is implemented by placing the processor into a low power mode from within the Idle Task hook function, as shown in Figure 1.

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 mode to process RTOS Tick interrupts.

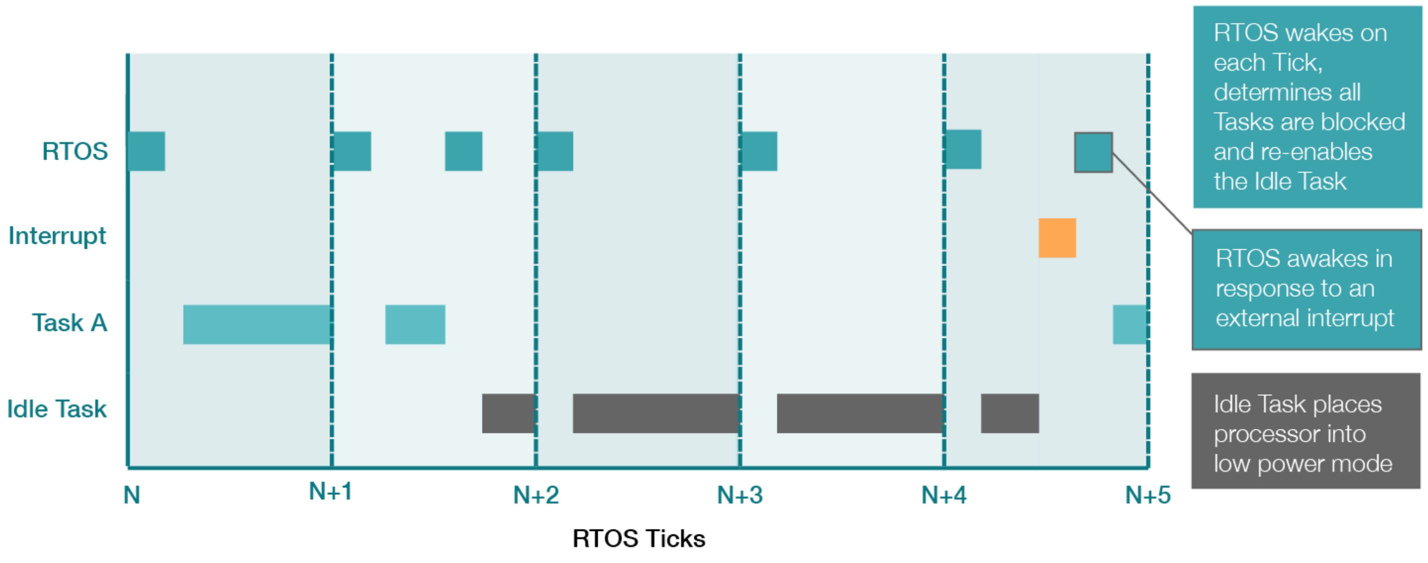
RTOS
Interrupt
Task A
Idle Task
N
N+1
N+2
N+3
N+4
N+5
RTOS Ticks
RTOS wakes on each Tick, determines all Tasks are blocked and re-enables the Idle Task
RTOS awakes in response to an external interrupt
Idle Task places processor into low power mode

Figure 1. The interaction of the RTOS and Idle Task during Sleep Mode

WITTENSTEIN high integrity systems
Americas: +1 408 625 4712
ROTW: +44 1275 395 600

email: sales@highintegritysystems.com
web: www.highintegritysystems.com

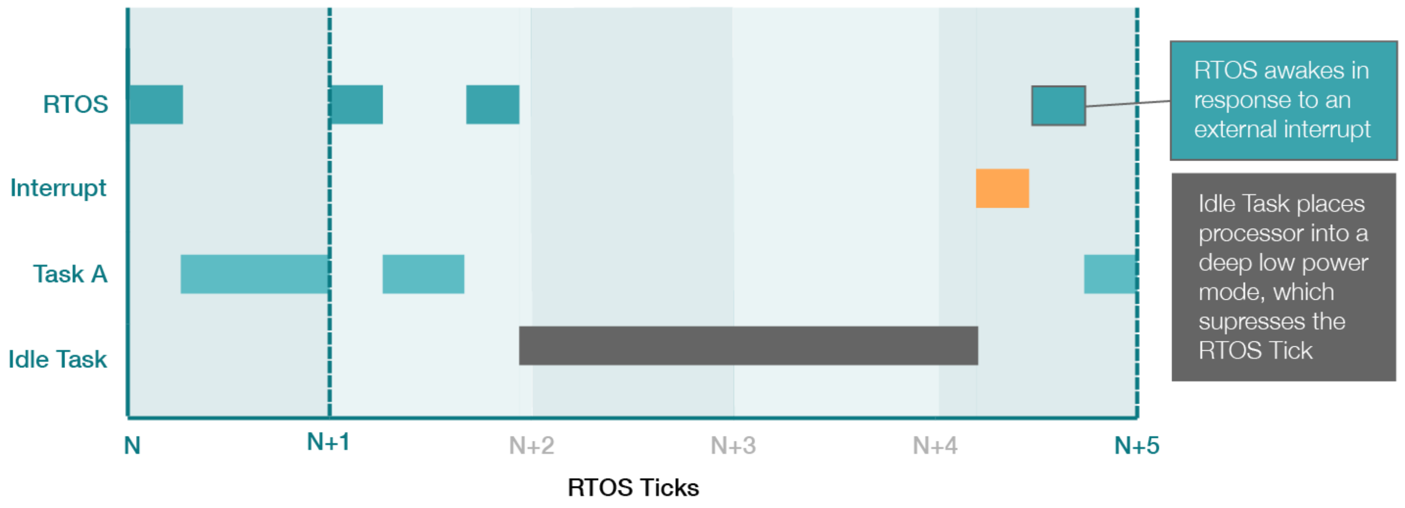
RTOS
Interrupt
Task A
Idle Task
RTOS awakes in response to an external interrupt
Idle Task places processor into a deep low power mode, which supresses the RTOS Tick
N
N+1
N+2
N+3
N+4
N+5
RTOS Ticks

Figure 2. The interaction of the RTOS and Idle Task during Tickless Sleep Mode

Tickless Sleep Mode

Tickless Sleep Mode extends the power save strategy previously presented further, by completely stopping the RTOS Tick interrupt during idle periods. Stopping the Tick interrupt allows the processor to remain in a deep power saving mode longer until either an interrupt occurs, or it is time for the RTOS to re-activate a Task, as shown in Figure 2.

When using OPEN**RTOS**, Tickless Sleep Mode can be entered when:

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 a user-defined value. this is done to avoid having to calculate and reprogram the clock on each tick.

In Tickless Sleep Mode the processor will be put to sleep until the next Task Block Time expires, or the next Software Timer is triggered, whichever is sooner. If the following conditions are true, OPEN**RTOS** will place the processor in deep sleep mode at least until the next external interrupt.

1. Software timers are not being used, or are not due to expire, hence OPEN**RTOS** is not due to execute a Timer Callback function at any time in the future.
2. All the application Tasks are either in the Suspended state, or in the Blocked state with an infinite timeout (a timeout value of portMAX_DELAY), hence OPEN**RTOS** is not due to transition a task out of the Blocked state at any fixed time in the future.

OPEN**RTOS** knows the longest time it can possibly sleep without the risk of missing a Task Block Time expiring, or a Software Timer executing. However it cannot predict when it will “actually” exit the low power sleep mode, because it can't predict asynchronous interrupts being accepted. Therefore, when OPEN**RTOS** exits a low power sleep, it cannot make any assumptions about how long it was asleep for and so calculates the actual sleep time before adjusting the system time accordingly.

Summary

This article presents two very simple power saving strategies that most embedded RTOS’s will support and that may help lower the power consumption of the processor.

WITTENSTEIN high **integrity** systems

Worldwide Sales and Support
Americas: +1 408 625 4712
ROTW: +44 1275 395 600
Email: sales@high**integrity**systems.com

WITTENSTEIN