AN1358: Using Power Manager Instead of Sleep Driver when Migrating Projects from GSDK 2.x

This application note provides information about Power Manager and compares Power Manager with its GSDK 2.x predecessor Sleep Driver. Users developing applications with GSDK 3.0 and later must migrate to Power Manager, as Sleep Driver is deprecated. For users migrating from Sleep Driver to Power Manager, this note compares their respective APIs so that the user can use the best Power Manager API for their purpose. Some points to be considered when working with Power Manager are also covered.

KEY POINTS

- Introduction to Power Manager
- Power Manager APIs
- Comparison with Sleep Driver
- Considerations when working with Power Manager

1 Introduction to Power Manager

Power Manager is a platform-level software module that manages the system's energy modes. Its main purpose is to transition the system to a lower energy mode when the processor has nothing to execute. Each time the system goes to sleep, Power Manager uses requirements to determine the target energy mode. These requirements are set by the different software modules (drivers, stack, application code, and so on). Power Manager also ensures strict control of some power-hungry resources such as the high frequency external oscillator (normally called HFXO). In other words, Power Manager can be defined as a central module that all stacks could use by communicating their power and resource requirements, and that would then turn OFF everything that is not needed at any given time in order to save as much power as possible.

1.1 Features of Power Manager

1. Power Manager acts as the middleman between different software modules and the device.
2. Energy Mode EM0 (where everything is powered ON) is the highest energy mode supported by Power Manager. EM3 (where only the ultra-low frequency oscillator is enabled) is the lowest energy mode supported.
3. Power Manager's purpose is to put the system into sleep mode when the CPU has nothing to process. The sleep mode (or) energy mode the system will enter is determined by the requirements. Using the requirement APIs, requirements for a specific energy mode can be added or removed. When a requirement is added on a particular energy mode, Power Manager does not sleep in a mode lower than that energy mode. For example, when transmitting data on a USART, the system can go to sleep. However, it can only sleep in EM1 mode, as otherwise the USART would stop working. The correct way to handle this is to add a requirement on EM1 when the transfer is set up and remove the requirement on EM1 once the transfer complete ISR is triggered. Add and remove requirement functions must always be called in pairs. If the remove function is not called after adding, then Power Manager will not be able to sleep in a mode lower than the added energy mode.
4. Power Manager offers a notification mechanism. This mechanism allows any piece of software module to be notified of any energy mode transition. When transitioning from a high energy mode to lower one (for example EM0 to EM2), the listeners are notified before the transition. When transitioning from a lower energy mode to higher energy mode (for example EM2 to EM0), the listeners are notified after the transition is completed. The main purpose of these notifications is for the different software modules to adapt to the new energy mode. Requirements cannot be added or removed from a notification.
5. Power Manager contains a sleep loop, which checks for the lowest possible energy mode that a system can transit into and makes the system enter it. It uses logic where, by default, the system goes back to sleep after processing an interrupt unless a wakeup is requested.
6. Power Manager offers a contribution mechanism through which any software module can contribute to the decision to go to sleep. Before entering the sleep loop, Power Manager checks if it is ok to sleep and then enters the sleep loop.
7. Power Manager implements a full restore when waking up from EM1 energy mode. When waking up from EM2/EM3 energy modes, Power Manager offers partial or full restore depending on the events that trigger the wake up. If the events are synchronous, then Power Manager wakes up in advance and restores the system. If the events are asynchronous, then Power Manager partially restores the system based on the requirements added.
8. Power Manager uses Sleep Timer for precision timing, to wake up in advance and prepare the system at right time.

2 Power Manager APIs

This section briefly describes some of the APIs offered by Power Manager. For more information on APIs and API examples, see the Power Manager Service online documentation.

2.1 sl_power_manager_init()

This API is used to initialize Power Manager. It also does all the required hardware initialization. Before this API is called, all the clock tree setup must be complete and cannot be changed later.

Example:

```
sl_power_manager_init();
```

This call is not required if `sl_system_init()` is being used.

2.2 sl_power_manager_sleep()

When this API is triggered, Power Manager enters in a sleep loop and will not exit until a system wakeup is requested after an ISR. Before entering into the sleep loop, the API first checks if it is ok to sleep by calling the callback `sl_power_manager_is_ok_to_sleep()`. This API also handles the full system restore after exiting from the sleep loop.

Note: This API is only required for bare metal applications. Applications using RTOS must not call this function.

Example:

```
sl_power_manager_sleep();
```

This call makes the system to go to lowest energy mode possible. If Kernel is being used, then this call is not required.

2.3 sl_power_manager_add_em_requirement()

This API is used to add a requirement on a given energy mode. Once a requirement is added for an energy mode, Power Manager does not transit lower than that energy mode. If this function is called from an ISR while in a sleep loop to add a requirement on a higher energy mode, it automatically triggers the required restoration and the function will not return until that restore is completed.

Example:

```
sl_power_manager_add_em_requirement(SL_POWER_MANAGER_EM1);
```

This call adds requirement for EM1, which will restrict the system from entering energy mode EM2 or EM3.

2.4 sl_power_manager_remove_em_requirement()

This API is used to remove a requirement on an energy mode. Add and remove requirement functions must be called in pairs. ,If a requirement previously added on a given energy mode is not removed, the application cannot sleep in a lower energy mode. For example, assume a requirement on energy mode EM1 was added, but was not subsequently removed. Power Manager cannot transit to an energy mode lower than EM1 until the requirement is removed.

Example:

```
sl_power_manager_remove_em_requirement(SL_POWER_MANAGER_EM1);
```

This call removes the requirement on EM1, now Power Manager can transit into either EM2 or EM3 depending on the configuration.

2.5 sl_power_manager_subscribe_em_transition_event()

This API can be used to register a callback when transitioning to a different energy mode. All registered subscribers are notified during transitioning. Requirements cannot be added to or removed from a callback on a transition event.

Example:

```
sl_power_manager_subscribe_em_transition_event(&event_handle, &event_info);
```

This call subscribes to notification, with event handle “event_handle” and power manager will trigger respective callback associated with the handle for every energy mode transition.

2.6 sl_power_manager_unsubscribe_em_transition_event()

This API must be used to unregister from an event callback handle on energy mode transition.

Example:

```
sl_power_manager_subscribe_em_transition_event(&event_handle);
```

This call unsubscribes notification to the event handle being passed.

2.7 sl_power_manager_schedule_wakeup_set_restore_overhead_tick()

This API can be used to set a configurable overhead value in Sleep Timer ticks for an early restore time used for a scheduled wake up. This API must be called after initialization, or the value will be overwritten.

Example:

```
sl_power_manager_schedule_wakeup_restore_overhead_tick(overhead_tick);
```

This call will set the configurable overhead value for early restore time in sleep timer ticks used for schedule wake-up. The overhead_tick value is the value to set for early restore time.

Note: The overhead_tick value can also be negative to remove time from the restore process.

3 Differences between Power Manager and Sleep Driver

Both Power Manager and Sleep Driver are platform-level software modules for sleep management. The main purpose of both these modules is to make it easy for the application to always enter the lowest possible energy mode using a simple API. Power Manager provides additional functionalities as well as the basic functionalities formerly provided by Sleep Driver. This section describes the differences between some of the features present in Power Manager and Sleep Driver.

3.1 "Block" in Sleep Driver is "Requirements" in Power Manager

In Sleep Driver, "Block" functionality is used to restrict the lowest energy mode that a system can transit into. With the "Block" functionality the application can block specific energy modes, to prevent the system from transiting into a lower mode than the one specified. An energy mode can be blocked and unblocked using the `SLEEP_SleepBlockBegin()` and `SLEEP_SleepBlockEnd()` APIs respectively.

In Power Manager, "Requirements" functionality is used to restrict the lowest energy mode that a system can transit into. With the "Requirements" functionality the application can add requirements for a specific energy mode. Power Manager ensures that it does not sleep in a mode lower than that energy mode. Once the requirement is added, it must be removed once the required tasks requiring that energy mode are performed, else Power Manager cannot transit to a lower energy mode. The requirements can be added and removed using `sl_power_manager_add_em_requirement()` and `sl_power_manager_remove_em_requirement()` APIs.

Table 4.1. Sleep Driver APIs and their Power Manager Equivalents

Sleep Driver	Power Manager	Functionality
SLEEP_SleepBlockBegin(sleepEM2)	sl_power_manager_add_requirement(SL_POWER_MANAGER_EM1)	Restricts the system from entering energy mode EM2 or lower.
SLEEP_SleepBlockEnd(sleepEM2)	sl_power_manager_remove_requirement(SL_POWER_MANAGER_EM1)	Removes the restriction.
SLEEP_SleepBlockBegin(sleepEM3)	sl_power_manager_add_requirement(SL_POWER_MANAGER_EM2)	Restricts the system from entering energy mode EM3 or lower.
SLEEP_SleepBlockEnd(sleepEM3)	sl_power_manager_remove_requirement(SL_POWER_MANAGER_EM2)	Removes the restriction.

3.2 Register Multiple Callbacks for Energy Mode Transition

Sleep Driver has a provision to register only one callback for energy mode transition. It is not possible to register multiple callbacks.

Power Manager supports registering multiple callbacks for energy mode transition. Each software module that is subscribing to the notification from Power Manager can add its own specific callback for energy mode transition.

3.3 New Sleep on ISR Exit Feature

Power Manager has a sleep loop, which will check if it is ok to enter sleep each time an interrupt (or multiple interrupts) wakes the system up. API `sl_power_manager_sleep_on_isr_exit()` is used to validate if it is ok to enter sleep after ISR exits.

3.4 Power Manager uses Sleep Timer

Power Manager is tightly coupled with Sleep Timer for precision timing, so that it can wake up a bit early and get the system ready at the right time for synchronous events. Before entering sleep, Power Manager looks at the next timer to expire and ensures that everything will be restored once the timer expires.

The Sleep Driver does not use Sleep Timer and manual callbacks must be implemented to handle the early wake up and system restore for synchronous events.

Note: For more information on Synchronous and Asynchronous events, see the Power Manager Service online documentation.

3.5 Neither Handle Energy Mode EM4

Neither Sleep Driver nor Power Manager support handling of energy mode EM4, since the wake up from EM4 requires a full system reset. Sleep Driver has an API `SLEEP_ForceSleepIntoEM4()` to force the system to transit into EM4. In GSDK 3.x, the application must handle entry into and exit from EM4..

4 Considerations when Working with Power Manager

This section focuses on considerations when working with Power Manager.

4.1 Power Manager Initialization

Power Manager must be initialized before calling any Power Manager API. If sl_system is used only `sl_system_init()` must be called, otherwise `sl_power_manager_init()` must be called manually. Power Manager must be initialized after the clocks(s) when initializing manually, as Power Manager checks which oscillators are used during the initialization phase.

4.2 Only Energy Modes EM1 and EM2 Can Be Added as a Requirement

Since Power Manager does not support handling of energy mode EM4, only EM1 and EM2 energy modes can be added as requirements.

4.3 Waking Up from Deep Sleep

When waking up from deep sleep (energy modes EM2 and EM3) and executing the interrupt's ISR, the device only restores fast startup RC oscillator to provide a high frequency clock source for the CPU. This is because the process of restoring all the clock sources can be time consuming and requires energy. Full clock restore only happens if the interrupt is triggered by Synchronous events. See Power Manager Service online documentation, for more information on Synchronous, Asynchronous events and the detailed process of waking up from deep sleep.

4.4 Entering Energy Mode EM4

Power Manager does not support the energy mode EM4. However, the application can use EM4 mode. Usually a system enters EM4 when the sleep period is to be fairly long and the wakeup source is quite basic (GPIO interrupt, low energy timer compare match, and so on). As such, in most circumstances, the application is in the best position to decide to use EM4.

Note: For information on how to enter EM4 explicitly, see the Power Manager Service online documentation.

Smart. Connected. Energy-Friendly.

IoT Portfolio
www.silabs.com/products

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com