AN13956

Power Manager Framework Usage for MCU Class of Devices

Rev. 1 — 10 August 2023 Application note

Document Information

Information	Content
Keywords	AN13956, MCUXpresso SDK, power optimizations
Abstract	This document describes how to leverage the power manager framework to optimize power consumption in an application.

1 Introduction

MCUs feature different low-power states for static power optimizations. Multiple states such as ON, retention (for memories), or OFF are defined for each MCU low-power state, peripheral, memory, or clock. Each of these states can be specific to a device.

The NXP MCUXpresso SDK includes the power manager component. This component is a software framework for BareMetal code and RTOS applications. The power manager component aims to speed up the development of these applications. By abstracting the SoC architecture, the developer can easily integrate the management of low-power states in the application and speed up the time to market. The SDK power manager uses low-level drivers to offload the entire comprehension of the device by providing the resources and operating modes constraints mechanism. The SDK power manager also optimizes the power consumption by shutting down the resources not required by the application.

2 Acronyms

Table 1 defines the acronyms used in this document.

Table 1. Acronyms

Acronym	Description
BareMetal	Application/driver code without an operating system
EVK	Evaluation kit/ evaluation board
MCU	Microcontroller unit
SDK	Software development kit
SoC	System on chip

3 SDK power manager

This section explains the features and architecture of the SDK power manager.

3.1 Features

SDK power manager consists of the following features:

- Manages the transition for different operating modes by seamlessly modifying the registers based on resource constraints:
  - SDK power manager turns OFF all the resources by default, except the ones required by the application.
- Eases the management of wake-up sources.
- Notifies the upper layer. For example, the SDK power manager notifies the application about power transitions or wake-up events.
- Gathers constraints and/or finds the lowest-power state achievable depending on application constraints or timing (if declared):
  - The application can specify the low-power state to enter and the resource constraints. If the resource constraints or timings do not match the constraints of the low-power state, the SDK power manager identifies a lighter low-power state that satisfies these requirements.
  - The application prompts the user to enter a low-power state. The SDK power manager turns OFF all the possible resources satisfying the low-power states.
  - The application can set resource constraints. The SDK power manager identifies the deepest low-power state to enter that satisfies the resource constraints.

3.2 Architecture

The SDK power manager is composed of two parts as follows:

- "Core" part: This part is generic across devices and provides APIs to be called in the application. This part is composed of different submodules:
  - Policy module: Gathers all the constraints and identifies the deepest power state allowed.
  - Wake-up-source manager module: Configures the wake-up sources and processes registered wake-up-source handler callbacks.
  - Notification module: Notifies the upper layer of specific power transitions and events.
- "Device" part: This part is specific for each device and describes the entry/exit sequences of the power modes, called the *sequencer*. There is also a description of all the resource constraints available: the pre-defined constraints for the low-power states. This translation, extracted from the device reference manual, shows whether clocks and peripherals are available or not for each low-power state. The user cannot modify this part. Instead, the application defines the resource constraints to be kept enabled for a given low-power state. Each device has a constraint for each power mode in terms of resources. For example, for RT500 in Deep Power-down mode, SYS PLL must be OFF while RTC must be powered ON. This is the translation of what is available or not in terms of clock/peripherals for each low-power state, that is, taken from the reference manual.

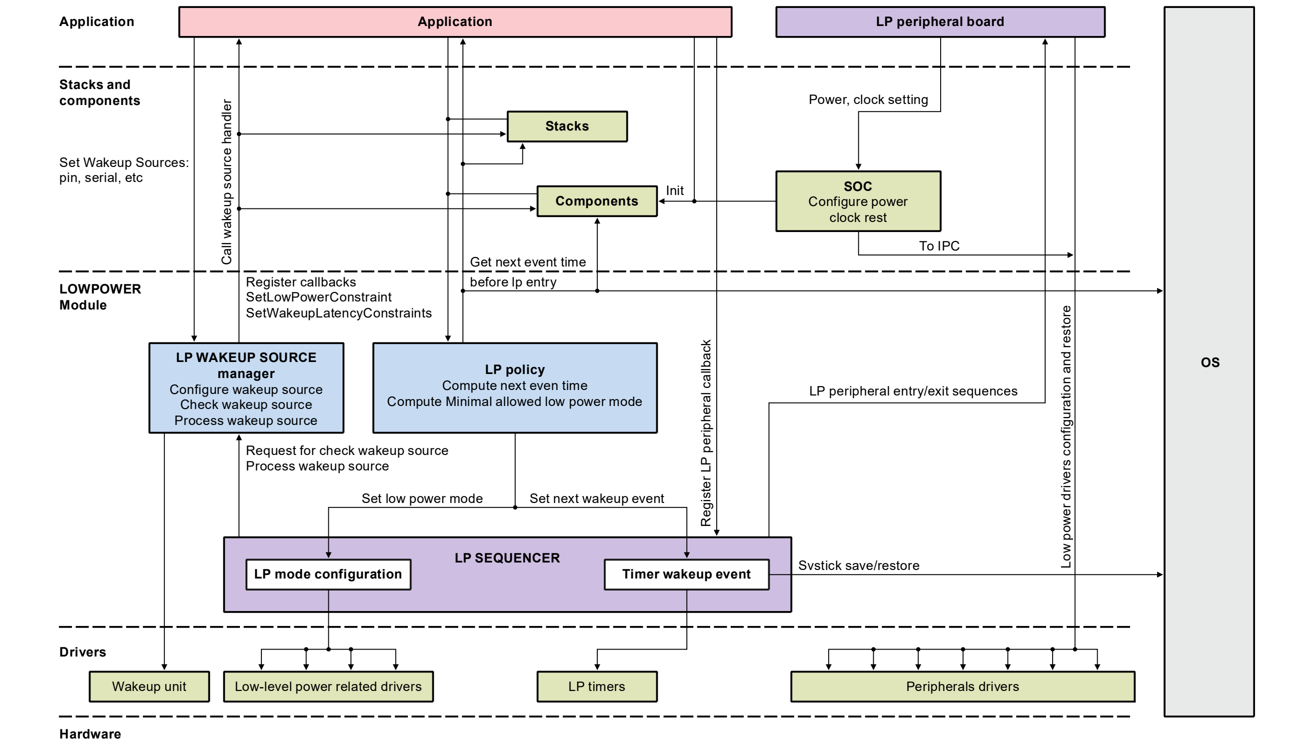

Figure 1. SDK power manager architecture diagram

NXP defines and develops the framework, exposing the pre-defined constraints and easy-to-use APIs to the user for application development.

To use the SDK power manager, consider the examples of APIs, as shown in Table 2 that are to be called in the application:

Table 2. Example of APIs

Name	Mandatory/Optional in the application	Description
`PM_CreateHandle`	Mandatory	Initializes the power manager handler, to be called before using other power manager APIs
`PM_RegisterNotify`	Optional	Registers a notify element into the selected group. The callback of the group is called before the entry to the low-power state and after the exit from the low-power state.
`PM_InitWakeupSource`	Optional	Initialize the wake-up source
`PM_RegisterTimerController`	Optional	Register a timer as a wake-up source, to be called with `PM_InitWakeupSource`
`PM_SetConstraints`	Mandatory	Set constraints to the power manager defined by the user, and/or for a low-power state. To define constraints easily, the user can define a macro.
`PM_EnablePowerManager`	Mandatory	Enable/disable power manager functions
`PM_EnterLowPower`	Mandatory	Finds the ideal low-power state available based on registered constraints, then notifies groups, and enters/exits the low-power state.

The user can modify the macros given in Table 3 depending on the requirement, available in `fsl_pm_device_config.h`:

Table 3. Macros

Name	Description
`FSL_PM_SUPPORT_NOTIFICATION`	Allows the power manager to notify created notification groups of power transitions, that is, the entry/exit of a state. It can be useful to re-enable a peripheral just after exiting the low-power state.
`FSL_PM_SUPPORT_WAKEUP_SOURCE_MANAGER`	Allows the power manager to manage wake-up sources entirely: create, disable, handle, trigger
`FSL_PM_SUPPORT_LP_TIMER_CONTROLLER`	Allows the power manager to control timers
`FSL_PM_SUPPORT_ALAWAYS_ON_SECTION`	Allows the power manager to store variables in an always-on RAM

For more details on APIs available and description, see fsl_pm_core files.

4 Application example

The example used in this document is a BareMetal application based on the i.MX RT500 EVK. This example demonstrates low-power transition by using the SDK power manager. The code is running in SRAM partition 16 at address 0x2010 0000 with a size of 256 kB (0x40000). The code is stored in external Octal flash using FlexSPI0.

Table 4. Application example

Partition number	Size	M33code/DSP code address	Fusion DSP data address	All other AHB controllers and GPU/LCD address
16	256 kB	0x0010 0000	0x0090 0000	0x2010 0000 (AHB P7)

The first step defines the resources that the user wants to keep ON or retain for a specific low-power state. Each resource-constraint definition is already defined in the file `fsl_pm_device.h`.

```
#define PM_RESC_ACMP_ACTIVE PM_ENCODE_RESC(PM_RESOURCE_FULL_ON, kResc_ACMP)
#define PM_RESC_PQ_SRAM_ACTIVE PM_ENCODE_RESC(PM_RESOURCE_FULL_ON, kResc_SRAM_PQ)
#define PM_RESC_FLEXSPI0_SRAM_ACTIVE PM_ENCODE_RESC(PM_RESOURCE_FULL_ON, kResc_SRAM_FLEXSPI0)
#define PM_RESC_FLEXSPI0_SRAM_RETENTION PM_ENCODE_RESC(PM_RESOURCE_PARTABLE_ON1, kResc_SRAM_FLEXSPI0)
```

For example, using `PM_RESC_ACMP_ACTIVE` ensures that the resource ACMP remains active during a low-power state if it complies with the pre-defined constraints.

For Deep Sleep mode, which is specified later as a power mode constraint, only the mandatory resources are ON. Therefore, the example application is declared as follows:

```
#define APP_DEEP_SLEEP_CONSTRAINTS        \
2U, PM_RESC_SRAM16_256KB_RETENTION, PM_RESC_FLEXSPI0_SRAM_RETENTION
```

In Deep Sleep mode, only the defined SRAM partition and the FlexSPI0 SRAM are retained, that is, the memory is retained but not accessible. All the other resources are turned OFF.

For the Sleep low-power state, the following resources are kept ON:

```
#define APP_SLEEP_CONSTRAINTS
7U, PM_RESC_MAIN_CLK_ON, PM_RESC_SYSXTAL_ON, PM_RESC_LPOSC_ON, PM_RESC_SYSPLLLDO_ON, PM_RESC_SYSPLLANA_ON,
 PM_RESC_FLEXSPI0_SRAM_ACTIVE, \ PM_RESC_SRAM16_256KB_ACTIVE
```

The application must first create the PM handle as follows:

```
PM_CreateHandle(&g_pmHndle);
```

To declare wake-up sources, the application must first call the PM API, and then declare the wake-up-source parameter. This example uses the SW2 button on the EVK as a wake-up source.

The example application calls the PM API with the corresponding parameter:

```
PM_InitWakeupSource(&g_UserkeyWakeupSource, (uint32_t)PIN_INT0_IRQn, NULL, true);
```

Then the user defines the GPIO parameter using the MCUXpresso SDK driver APIs:

```
gpio_pin_config_t gpioPinConfigStruct;
/* Set SW pin as GPIO input. */
gpioPinConfigStruct.pinDirection = kGPIO_DigitalInput;
GPIO_PinInit(APP_USER_WAKEUP_KEY_GPIO, APP_USER_WAKEUP_KEY_PORT, APP_USER_WAKEUP_KEY_PIN, &gpioPinConfigStruct);
/* Configure the Input Mux block and connect the trigger source to PinInt channel. */
INPUTMUX_Init(INPUTMUX);
INPUTMUX_AttachSignal(INPUTMUX, kPINT_PinInt0, APP_USER_WAKEUP_KEY_INPUTMUX_SEL); /* Using channel 0. */
INPUTMUX_Deinit(INPUTMUX); /* Turnoff clock to inputmux to save power. Clock is only needed to make changes */
/* Configure the interrupt for SW pin. */
PINT_Init(PINT);
PINT_PinInterruptConfig(PINT, kPINT_PinInt0, kPINT_PinIntEnableFallEdge, pint_intr_callback);
PINT_EnableCallback(PINT); /* Enable callbacks for PINT */
```

Next, the application sets the defined constraints, with the following functions:

```
PM_SetConstraints(PM_LP_STATE_DEEP_SLEEP, APP_DEEP_SLEEP_CONSTRAINTS);
```

There are two types of constraints that can be set:

- Constraints on the low-power mode.
- Constraints on the resources.

Consider the following example:

```
PM_SetConstraints(PM_LP_STATE_NO_CONSTRAINT, APP_DEEP_SLEEP_CONSTRAINTS);
```

The command line above specifies no low-power mode. The power manager identifies the deep power state that satisfies the resource constraints set in `APP_DEEP_SLEEP_CONSTRAINTS`.

Consider another example as follows:

```
PM_SetConstraints(PM_LP_STATE_DEEP_SLEEP, APP_DEEP_SLEEP_CONSTRAINTS);
```

The command line above sets a constraint on the low-power mode and the resources. The power manager compares the resource constraints to the pre-defined ones for the low-power mode. If an incompatibility occurs, the power manager identifies a lighter low-power mode that satisfies the resource constraints set by the user. For example, with the i.MX RT500, if the power manager cannot meet the resource constraints using Deep Sleep mode, it next tries to meet these constraints using Sleep mode.

For the other cases in the above example, Deep Sleep mode is reached with the resources specified in `APP_DEEP_SLEEP_CONSTRAINTS` kept ON.

Another example is as follows:

```
PM_SetConstraints(PM_LP_STATE_DEEP_SLEEP, 0);
```

The command line above sets only a low-power mode constraint without resource constraints. Therefore, the power manager turns everything OFF in this state, except the pre-defined resource constraints for this low-power mode. This case is rarely used, as resources are always required for RAM retention to wake the device in sleep and deep sleep properly. For lower low-power states where a reset is required, use this type of constraint.

Constraints on the low-power mode are a priority. In other words, if there are two constraints on the low-power mode, the power manager selects the lighter one.

Next, the application enables the SDK power manager framework and enters the low-power mode:

```
PM_EnablePowerManager(true);
```

To enter in a low-power mode, the following power manager function must be used:

```
PM_EnterLowPower(durationTicks);
```

When an exit latency is declared for a low-power state, the `durationTicks` parameter can be used. If the specified duration is less than the exit latency of the low-power state, it influences the low-power state entered.

The power manager component defines the exit latency of each device for a low-power state. For example, in the i.MX RT500, Deep Sleep mode has an exit latency of 250 µs and is declared as follows in `fsl_pm_device.c`:

```
/* Deep Sleep */
            {
                .exitHwLatency = 250U, /* 250 us */
```

When the power manager tries to identify the deep state reachable, it compares the exit latency of the low-power state `exitHwLatency` with the `durationTicks` specified by the application. Even if the resource constraints are satisfied, the Deep Sleep state is unreachable if the `durationTicks` variable is less than or equal to 250 µs. A lighter low-power state is reached, satisfying the resource constraints and `durationTicks`.

The application can also set constraints on resources and low-power mode following a similar mechanism. Consider the following example for Sleep mode:

```
PM_SetConstraints(PM_LP_STATE_SLEEP, APP_SLEEP_CONSTRAINTS);
```

The power manager enters the lightest low-power mode by having two low-power mode constraints: Sleep and Deep Sleep. In this example, the Sleep state is entered, as no exit latency exists in this state. In other words, the framework compares the time passed as a parameter to `PM_EnterLowPower()`, that is, the amount of time to spend in the low-power state, with the minimum exit time of the low-power mode. This time is the minimum time spent in this low-power state. Therefore, if `durationTicks` > `exitHwLatency`, then the low-power state can be reached. If not, the framework tries a lighter low-power mode. The resource constraints that must be maintained are the sum of the previous ones with the newly defined ones, that is, `APP_DEEP_SLEEP_CONSTRAINTS` + `APP_SLEEP_CONSTRAINTS`.

To unset resource constraint and/or low-power mode constraint, the following function must be used:

```
PM_ReleaseConstraints(PM_LP_STATE _SLEEP, APP_DEEP_SLEEP_CONSTRAINTS);
```

Here, the Sleep state is no longer registered in the power manager. If the `durationTicks` exceeds the exit latency of the Deep Sleep state, the Deep Sleep state is entered with the corresponding constraints. The `APP_DEEP_SLEEP_CONSTRAINTS` resource constraints are also unregistered, therefore the resource constraints to maintain are `APP_SLEEP_CONSTRAINTS`. In other words, the user can define resources that must be ON for a given low-power state with the help of macros. In this example, there are two constraints in the low-power mode: Sleep and Deep Sleep, each with respective constraints. If the user unregisters the Deep Sleep state, then the constraints for the Deep-Sleep mode are also unregistered. Therefore, the resource constraints to be maintained only apply to the Sleep state.

Note: *If a low-power mode constraint or resource constraint is set multiple times through `PM_SetConstraints`, call the `PM_ReleaseConstraints` function as many times as required to remove the constraint completely from the SDK power manager. This behavior is useful where multiple peripherals set a constraint in a low-power mode.*

Finally, disable the power manager with the following function when it is not required anymore in the application:

```
PM_EnablePowerManager(false);
```

5 APIs references

This section describes the API references used in this application note as follows:

1. PM_CreateHandle
2. PM_EnablePowerManager
3. PM_EnterLowPower
4. PM_RegisterTimerController
5. PM_GetLastLowPowerDuration
6. PM_RegisterCriticalRegionController
7. PM_RegisterNotify
8. PM_UpdateNotify
9. PM_UnregisterNotify
10. PM_InitWakeupSource
11. PM_EnableWakeupSource
12. PM_DisableWakeupSource
13. PM_HandleWakeUpEvent

5.1 PM_CreateHandle

```
void PM_CreateHandle (pm_handle_t * handle);
```

Description

This function initializes the power manager handle. This function must be invoked before using other power manager APIs.

Note: *By default, the power manager is disabled.*

Parameters

`handle`: Pointer to the `pm_handle_t structure`, upper-layer software must pre-allocate the handle-global variable.

Returns

None

5.2 PM_EnablePowerManager

```
void PM_EnablePowerManager (bool enable);
```

Description

This function enable/disables the power manager functions.

Parameters

`enable`: Used to enable/disable the power manager functions.

Returns

None

5.3 PM_EnterLowPower

```
void PM_EnterLowPower (uint64_t duration);
```

Description

This API is a power manager core API. If using an RTOS, call this API in the Idle task.

This function contains the following steps:

1. Compute the target power state based on the policy module.
2. Notify the upper layer software of the power mode transitions.
3. Enter into the targeted power state.
4. Exit from the low-power state if the wake-up event occurs.
5. Notify the upper layer software of the power mode exiting.

The target power state is determined based on two factors:

- The input parameter must be larger than the `exitHwLatency` attribution of the state.
- resConstraintsMask logical AND lossFeature of the state must be equal to 0.

Parameters

duration: The time (μ) in low-power mode.

Returns

None

5.4 PM_RegisterTimerController

```
void PM_RegisterTimerController
 (pm_handle_t * handle,
 pm_low_power_timer_start_func_t timerStart,
 pm_low_power_timer_stop_func_t timerStop,
 pm_low_power_timer_get_timestamp_func_t getTimestamp,
 pm_low_power_timer_get_duration_func_t getTimerDuration);
```

Description

If a low-power timer is a wake-up source, ensure to register it into the power manager using the `PM_InitWakeupSource` function.

Parameters

handle: Pointer to the `pm_handle_t` structure.

timerStart: Low-power timer start function. This parameter can be NULL. It means that the low-power timer is not set as the wake-up source.

timerStop: Low-power timer stop function. This parameter can also be set as NULL.

getTimestamp: Low-power timestamp function. This parameter can also be set as NULL.

getTimerDuration: Get timer duration function. This parameter can also be set as NULL.

Returns

None

5.5 PM_GetLastLowPowerDuration

```
void PM_GetLastLowPowerDuration (uint64_t duration);
```

Description

This API gets the actual low-power state duration.

Parameters

None

Returns

None

5.6 PM_RegisterCriticalRegionController

```
void PM_RegisterCriticalRegionController
```

```
 (pm_handle_t * handle,
 pm_enter_critical criticalEntry,
 pm_exit_critical criticalExit);
```

Description

This API registers critical region-related functions to the power manager.

Note: *There are multiple methods to implement critical regions. For example, interrupt controller, locker, and semaphore.*

Parameters

`handle`: Pointer to the `pm_handle_t` structure.

`criticalEntry`: Enter critical function to register.

`criticalExit`: Exit critical function to register.

Returns

None

5.7 PM_RegisterNotify

```
status_t PM_RegisterNotify
 (pm_notify_group_t groupId,
 const pm_notify_element_t * notifyElement);
```

Description

This API registers to notify elements into the selected group.

Parameters

`groupId`: The group of the notified list. This parameter affects the execution sequence.

`notifyElement`: Pointer to the `pm_notify_element_t`.

Returns

`status_t`: The status of the register notifies object behavior.

5.8 PM_UpdateNotify

```
void PM_UpdateNotify
 (void * notifyElement,
 pm_notify_callback_func_t callback,
 void * data);
```

Description

Update the notify callback function of the element and application data.

Parameters

`notifyElement`: Pointer to the notify element to update.

`callback`: The callback function to be updated.

`data`: Pointer to the callback function private data.

Returns

None

5.9 PM_UnregisterNotify

```
status_t PM_UnregisterNotify (void * notifyElement);
```

Description

This API removes the notify element from its notify group.

Parameters

notifyElement: Pointer to the notify element to remove.

Returns

None

5.10 PM_InitWakeupSource

```
void PM_InitWakeupSource
 (pm_wakeup_source_t * ws,
 uint32_t wsId,
 pm_wake_up_source_service_func_t service,
 bool enable);
```

Description

This API initializes the wake-up source object.

Parameters

ws: Pointer to the `pm_wakeup_source_t` variable.

wsId: Used to select the wake-up source, the `wsId` of each wake-up source can be found in `fsl_pm_device.h` or the device description file.

service: The function to be invoked when the wake-up source is asserted.

enable: Used to enable/disable the selected wake-up source.

Returns

None

5.11 PM_EnableWakeupSource

```
status_t PM_EnableWakeupSource (pm_wakeup_source_t * ws);
```

Description

This API enables wake-up source.

Parameters

ws: Pointer to the wake-up source object to be enabled.

Returns

status_t: The status of the enable wake-up source behavior.

5.12 PM_DisableWakeupSource

```
status_t _PM_DisableWakeupSource (pm_wakeup_source_t * ws);
```

Description

This API disables the wake-up source.

Parameters

ws: Pointer to the wake-up source object to be disabled.

Returns

status_t: The status of the disable wake-up source behavior.

5.13 PM_HandleWakeUpEvent

```
status_t_PM_HandleWakeUpEvent (void);
```

Description

This API checks if any enabled wake-up source is responsible for the last wake-up event. If it has been registered, it calls the wake-up source callback. It is likely to be called from the wake-up unit IRQ handler.

Parameters

None

Returns

status_t: The status of handling the wake-up event.

5.14 PM_TriggerWakeSourceService

```
status_t_PM_TriggerWakeSourceService (pm_wakeup_source_t * ws);
```

Description

If the specific wake-up event occurs, invoke this API to execute its service function.

Parameters

ws: Pointer to the wake-up source object

Returns

status_t: The status of the trigger wake-up source behavior.

5.15 PM_SetConstraints

```
status_t_PM_SetConstraints (uint8_t powerModeConstraint, int32_t rescNum, ...);
```

Description

This API is used to set constraints including power mode constraints and resource constraints. For example, if the board supports three resource constraints, such as `PM_RESC_1`, `PM_RESC_2`, and `PM_RESC3`, the function is as follows:

```
PM_SetConstraints(Sleep_Mode, 3, PM_RESC_1, PM_RESC_2, PM_RESC_3);
```

Parameters

powerModeConstraint: The lowest power mode allowed. The power mode constraint macros can be found in `fsl_pm_device.h`.

rescNum: The number of resource constraints to be set.

Returns

status_t: The status of the set constraints behavior.

5.16 PM_ReleaseConstraints

```
status_t_PM_ReleaseConstraints (uint8_t powerModeConstraint, int32_t rescNum, ...);
```

Description

This API is used to release constraints including power mode constraints and resource constraints. For example, if the board supports three resource constraints, such as `PM_RESC_1`, `PM_RESC_2`, and `PM_RESC3`, the function is as follows:

```
PM_ReleaseConstraints(Sleep_Mode, 1, PM_RESC_1);
```

Parameters

powerModeConstraint: The lowest power mode allowed. The power mode constraint macros can be found in `fsl_pm_device.h`.

rescNum: The number of resource constraints to be released.

Returns

status_t: The status of the set constraints behavior.

5.17 PM_GetResourceConstraintsMask

```
pm_resc_mask_t PM_GetResourceConstraintsMask (void);
```

Description

This API gets the current system resource constraints.

Parameters

None

Returns

Current system constraints.

5.18 PM_GetAllowedLowestPowerMode

```
uint8_t_PM_GetAllowedLowestPowerMode (void);
```

Description

This API gets the current system-allowed power mode.

Parameters

None

Returns

Allowed lowest power mode.

6 Conclusion

The power manager is a great option to reduce power consumption in low-power states by managing all the resources seamlessly for the user. By abstracting the overall power architecture and providing easy-to-use APIs/macros, this framework speeds up the time to market and application development.

7 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2023 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

8 Revision history

Table 5 summarizes revisions to this document.

Table 5. Revision history

Revision number	Release date	Description
1	10 August 2023	Initial public release

9 Legal information

9.1 Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

9.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. - NXP B.V. is not an operating company and it does not distribute or sell products.

9.3 Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

i.MX — is a trademark of NXP B.V.

Contents

Date of release: 10 August 2023
Document identifier: AN13956