

Xilinx Standalone Library Documentation

XilPM Library v2.4

UG1225 (2018.3) November 14, 2018

Table of Contents

Chapter 1: XilPM APIs

Overview	5
Data Structure Documentation	8
struct XPm_Notifier	8
struct XPm_NodeStatus	9
struct XPmClockSel2ClkIn	9
struct XPmClockMux	10
struct XPmClockModel	10
Enumeration Type Documentation	11
XPmApild	11
XPmApiCbld	11
XPmNodeId	11
XPmRequestAck	11
XPmAbortReason	11
XPmSuspendReason	11
XPmRamState	11
XPmOpCharType	11
XPmBootStatus	12
XPmResetAction	12
XPmReset	12
XPmNotifyEvent	12
XPmClock	12
Function Documentation	12
XPm_InitXilpm	12
XPm_SuspendFinalize	12
XPm_GetBootStatus	13
XPm_RequestSuspend	13
XPm_SelfSuspend	13
XPm_ForcePowerDown	14
XPm_AbortSuspend	14
XPm_RequestWakeUp	15
XPm_SetWakeUpSource	16
XPm_SystemShutdown	16

XPm_SetConfiguration	17
XPm_InitFinalize	17
XPm_InitSuspendCb	17
XPm_AcknowledgeCb	18
XPm_NotifyCb	19
XPm_RequestNode	19
XPm_ReleaseNode	20
XPm_SetRequirement	20
XPm_SetMaxLatency	21
XPm_GetApiVersion	21
XPm_GetNodeStatus	22
XPm_RegisterNotifier	23
XPm_UnregisterNotifier	23
XPm_GetOpCharacteristic	24
XPm_ResetAssert	24
XPm_ResetGetStatus	25
XPm_MmioWrite	25
XPm_MmioRead	26
XPm_ClockEnable	26
XPm_ClockDisable	27
XPm_ClockGetStatus	27
XPm_ClockGetDivider	27
XPm_ClockSetParent	28
XPm_ClockGetParent	28
XPm_ClockSetRate	28
XPm_ClockGetRate	29
XPm_PllSetParameter	29
XPm_PllGetParameter	30
XPm_PllSetMode	30
XPm_PllGetMode	30
XPm_PinCtrlRequest	31
XPm_PinCtrlRelease	31
XPm_PinCtrlSetFunction	31
XPm_PinCtrlGetFunction	32
XPm_PinCtrlSetParameter	32
XPm_PinCtrlGetParameter	32
Error Status	33
Overview	33

Appendix A: Additional Resources and Legal Notices

XilPM APIs

Overview

Xilinx Power Management(XilPM) provides Embedded Energy Management Interface (EEMI) APIs for power management on Zynq® UltraScale+™ MPSoC. For more details about power management on Zynq Ultrascale+ MPSoC, see the Zynq UltraScale+ MPSoC Power Management User Guide (UG1199). For more details about EEMI, see the Embedded Energy Management Interface (EEMI) API User Guide(UG1200).

Modules

- [Error Status](#)

Data Structures

- struct [XPm_Notifier](#)
- struct [XPm_NodeStatus](#)
- struct [XPmClockSel2CikIn](#)
- struct [XPmClockMux](#)
- struct [XPmClockModel](#)

Enumerations

- enum [XPmApild](#)
- enum [XPmApiCbld](#)
- enum [XPmNodeId](#)
- enum [XPmRequestAck](#)
- enum [XPmAbortReason](#)
- enum [XPmSuspendReason](#)
- enum [XPmRamState](#)
- enum [XPmOpCharType](#)
- enum [XPmBootStatus](#)
- enum [XPmResetAction](#)
- enum [XPmReset](#)
- enum [XPmNotifyEvent](#)
- enum [XPmClock](#)

Functions

- XStatus [XPm_InitXilpm](#) (XlpiPsu *lpInst)
- void [XPm_SuspendFinalize](#) (void)
- enum [XPmBootStatus XPm_GetBootStatus](#) (void)
- XStatus [XPm_RequestSuspend](#) (const enum [XPmNodeId](#) target, const enum [XPmRequestAck](#) ack, const u32 latency, const u8 state)
- XStatus [XPm_SelfSuspend](#) (const enum [XPmNodeId](#) nid, const u32 latency, const u8 state, const u64 address)
- XStatus [XPm_ForcePowerDown](#) (const enum [XPmNodeId](#) target, const enum [XPmRequestAck](#) ack)
- XStatus [XPm_AbortSuspend](#) (const enum [XPmAbortReason](#) reason)
- XStatus [XPm_RequestWakeUp](#) (const enum [XPmNodeId](#) target, const bool setAddress, const u64 address, const enum [XPmRequestAck](#) ack)
- XStatus [XPm_SetWakeUpSource](#) (const enum [XPmNodeId](#) target, const enum [XPmNodeId](#) wkup_node, const u8 enable)
- XStatus [XPm_SystemShutdown](#) (u32 type, u32 subtype)
- XStatus [XPm_SetConfiguration](#) (const u32 address)
- XStatus [XPm_InitFinalize](#) (void)
- void [XPm_InitSuspendCb](#) (const enum [XPmSuspendReason](#) reason, const u32 latency, const u32 state, const u32 timeout)
- void [XPm_AcknowledgeCb](#) (const enum [XPmNodeId](#) node, const XStatus status, const u32 oppoint)
- void [XPm_NotifyCb](#) (const enum [XPmNodeId](#) node, const u32 event, const u32 oppoint)
- XStatus [XPm_RequestNode](#) (const enum [XPmNodeId](#) node, const u32 capabilities, const u32 qos, const enum [XPmRequestAck](#) ack)
- XStatus [XPm_ReleaseNode](#) (const enum [XPmNodeId](#) node)
- XStatus [XPm_SetRequirement](#) (const enum [XPmNodeId](#) nid, const u32 capabilities, const u32 qos, const enum [XPmRequestAck](#) ack)
- XStatus [XPm_SetMaxLatency](#) (const enum [XPmNodeId](#) node, const u32 latency)
- XStatus [XPm_GetApiVersion](#) (u32 *version)
- XStatus [XPm_GetNodeStatus](#) (const enum [XPmNodeId](#) node, [XPm_NodeStatus](#) *const nodestatus)
- XStatus [XPm_RegisterNotifier](#) ([XPm_Notifier](#) *const notifier)
- XStatus [XPm_UnregisterNotifier](#) ([XPm_Notifier](#) *const notifier)
- XStatus [XPm_GetOpCharacteristic](#) (const enum [XPmNodeId](#) node, const enum [XPmOpCharType](#) type, u32 *const result)
- XStatus [XPm_ResetAssert](#) (const enum [XPmReset](#) reset, const enum [XPmResetAction](#) assert)
- XStatus [XPm_ResetGetStatus](#) (const enum [XPmReset](#) reset, u32 *status)
- XStatus [XPm_MmioWrite](#) (const u32 address, const u32 mask, const u32 value)
- XStatus [XPm_MmioRead](#) (const u32 address, u32 *const value)
- XStatus [XPm_ClockEnable](#) (const enum [XPmClock](#) clock)
- XStatus [XPm_ClockDisable](#) (const enum [XPmClock](#) clock)
- XStatus [XPm_ClockGetStatus](#) (const enum [XPmClock](#) clock, u32 *const status)
- XStatus [XPm_ClockGetDivider](#) (const enum [XPmClock](#) clock, u32 *const divider)
- XStatus [XPm_ClockSetParent](#) (const enum [XPmClock](#) clock, const enum [XPmClock](#) parent)
- XStatus [XPm_ClockGetParent](#) (const enum [XPmClock](#) clock, enum [XPmClock](#) *const parent)

- XStatus [XPm_ClockSetRate](#) (const enum [XPmClock](#) clock, const u32 rate)
- XStatus [XPm_ClockGetRate](#) (const enum [XPmClock](#) clock, u32 *const rate)
- XStatus [XPm_PllSetParameter](#) (const enum [XPmNodeId](#) node, const enum [XPmPllParam](#) parameter, const u32 value)
- XStatus [XPm_PllGetParameter](#) (const enum [XPmNodeId](#) node, const enum [XPmPllParam](#) parameter, u32 *const value)
- XStatus [XPm_PllSetMode](#) (const enum [XPmNodeId](#) node, const enum [XPmPllMode](#) mode)
- XStatus [XPm_PllGetMode](#) (const enum [XPmNodeId](#) node, enum [XPmPllMode](#) *const mode)
- XStatus [XPm_PinCtrlRequest](#) (const u32 pin)
- XStatus [XPm_PinCtrlRelease](#) (const u32 pin)
- XStatus [XPm_PinCtrlSetFunction](#) (const u32 pin, const enum [XPmPinFn](#) fn)
- XStatus [XPm_PinCtrlGetFunction](#) (const u32 pin, enum [XPmPinFn](#) *const fn)
- XStatus [XPm_PinCtrlSetParameter](#) (const u32 pin, const enum [XPmPinParam](#) param, const u32 value)
- XStatus [XPm_PinCtrlGetParameter](#) (const u32 pin, const enum [XPmPinParam](#) param, u32 *const value)

PM Version Number macros

- #define **PM_VERSION_MAJOR** 1
- #define **PM_VERSION_MINOR** 1
- #define **PM_VERSION** ((PM_VERSION_MAJOR << 16) | PM_VERSION_MINOR)

Capabilities for RAM

- #define **PM_CAP_ACCESS** 0x1U
- #define **PM_CAP_CONTEXT** 0x2U
- #define **PM_CAP_WAKEUP** 0x4U

Node default states macros

- #define **NODE_STATE_OFF** 0
- #define **NODE_STATE_ON** 1

Processor's states macros

- #define **PROC_STATE_FORCEDOFF** 0
- #define **PROC_STATE_ACTIVE** 1
- #define **PROC_STATE_SLEEP** 2
- #define **PROC_STATE_SUSPENDING** 3

Maximum Latency/QOS macros

- #define **MAX_LATENCY** (~0U)
- #define **MAX_QOS** 100U

System shutdown/Restart macros

- #define **PMF_SHUTDOWN_TYPE_SHUTDOWN** 0U
- #define **PMF_SHUTDOWN_TYPE_RESET** 1U
- #define **PMF_SHUTDOWN_SUBTYPE_SUBSYSTEM** 0U
- #define **PMF_SHUTDOWN_SUBTYPE_PS_ONLY** 1U
- #define **PMF_SHUTDOWN_SUBTYPE_SYSTEM** 2U

PM API Min and Max macros

- #define **PM_API_MIN** PM_GET_API_VERSION

Payload Packets

Assigning of argument values into array elements. pause and pm_dbg are used for debugging and should be removed in final version.

- #define **PACK_PAYLOAD**(pl, arg0, arg1, arg2, arg3, arg4, arg5)
- #define **PACK_PAYLOAD0**(pl, api_id) PACK_PAYLOAD(pl, api_id, 0, 0, 0, 0, 0)
- #define **PACK_PAYLOAD1**(pl, api_id, arg1) PACK_PAYLOAD(pl, api_id, arg1, 0, 0, 0, 0)
- #define **PACK_PAYLOAD2**(pl, api_id, arg1, arg2) PACK_PAYLOAD(pl, api_id, arg1, arg2, 0, 0, 0)
- #define **PACK_PAYLOAD3**(pl, api_id, arg1, arg2, arg3) PACK_PAYLOAD(pl, api_id, arg1, arg2, arg3, 0, 0)
- #define **PACK_PAYLOAD4**(pl, api_id, arg1, arg2, arg3, arg4) PACK_PAYLOAD(pl, api_id, arg1, arg2, arg3, arg4, 0)
- #define **PACK_PAYLOAD5**(pl, api_id, arg1, arg2, arg3, arg4, arg5) PACK_PAYLOAD(pl, api_id, arg1, arg2, arg3, arg4, arg5)

Data Structure Documentation

struct XPm_Notifier

[XPm_Notifier](#) - Notifier structure registered with a callback by app

Data Fields

- void(*const [callback](#))([XPm_Notifier](#) *const notifier)
- enum [XPmNodeId](#) node
- enum [XPmNotifyEvent](#) event
- u32 [flags](#)
- volatile u32 [oppoint](#)
- volatile u32 [received](#)
- [XPm_Notifier](#) * next

Field Documentation

void(*const callback) (XPm_Notifier *const notifier) Custom callback handler to be called when the notification is received. The custom handler would execute from interrupt context, it shall return quickly and must not block! (enables event-driven notifications)

enum XPmNodeId node Node argument (the node to receive notifications about)

enum XPmNotifyEvent event Event argument (the event type to receive notifications about)

u32 flags Flags

volatile u32 oppoint Operating point of node in question. Contains the value updated when the last event notification is received. User shall not modify this value while the notifier is registered.

volatile u32 received How many times the notification has been received - to be used by application (enables polling). User shall not modify this value while the notifier is registered.

XPm_Notifier* next Pointer to next notifier in linked list. Must not be modified while the notifier is registered. User shall not ever modify this value.

struct XPm_NodeStatus

[XPm_NodeStatus](#) - struct containing node status information

Data Fields

- u32 [status](#)
- u32 [requirements](#)
- u32 [usage](#)

Field Documentation

u32 status Node power state

u32 requirements Current requirements asserted on the node (slaves only)

u32 usage Usage information (which master is currently using the slave)

struct XPmClockSel2ClkIn

Pair of multiplexer select value and selected clock input

Data Fields

- enum [XPmClock clkIn](#)
- const u8 [select](#)

Field Documentation

enum XPmClock clkIn ID of the clock that is selected with the 'select' value

const u8 select Select value of the clock multiplexer

struct XPmClockMux

MUX select values to clock input mapping

Data Fields

- const [XPmClockSel2ClkIn](#) *const [inputs](#)
- const u8 [size](#)
- const u8 [bits](#)
- const u8 [shift](#)

Field Documentation

const XPmClockSel2ClkIn* const inputs Mux select to pll mapping at the input of the multiplexer

const u8 size Size of the inputs array

const u8 bits Number of bits of mux select

const u8 shift Number of bits to shift 'bits' in order to get mux select mask

struct XPmClockModel

Clock model

Data Fields

- enum [XPmClock id](#)
- const [XPmClockMux](#) *const [mux](#)
- const u8 [type](#)
- const [XPmClockModel](#) *const [next](#)

Field Documentation

enum XPmClock id Clock ID

const XPmClockMux* const mux Pointer to the mux model

const u8 type Type specifying the available divisors

const XPmClockModel* const next Next clock in the list

Enumeration Type Documentation

enum XPmApild

APIs for Miscellaneous functions, suspending of PUs, managing PM slaves and Direct control.

enum XPmApiCbld

PM API Callback Id Enum

enum XPmNodeId

PM Node ID Enum

enum XPmRequestAck

PM Acknowledge Request Types

enum XPmAbortReason

PM Abort Reasons Enum

enum XPmSuspendReason

PM Suspend Reasons Enum

enum XPmRamState

PM RAM States Enum

enum XPmOpCharType

PM Operating Characteristic types Enum

enum XPmBootTestStatus

Boot Status Enum

enum XPmResetAction

PM Reset Action types

enum XPmReset

PM Reset Line IDs

enum XPmNotifyEvent

PM Notify Events Enum

enum XPmClock

PM Clock IDs

Function Documentation

XStatus XPm_InitXilpm (XlpiPsu * *IpInst*)

Initialize xilpm library.

Parameters

<i>IpInst</i>	Pointer to IPI driver instance
---------------	--------------------------------

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

None

void XPm_SuspendFinalize (void)

This Function waits for PMU to finish all previous API requests sent by the PU and performs client specific actions to finish suspend procedure (e.g. execution of wfi instruction on A53 and R5 processors).

Note

This function should not return if the suspend procedure is successful.

enum XPmBootTestStatus XPm_GetBootTestStatus (void)

This Function returns information about the boot reason. If the boot is not a system startup but a resume, power down request bitfield for this processor will be cleared.

Returns

Returns processor boot status

- PM_RESUME : If the boot reason is because of system resume.
- PM_INITIAL_BOOT : If this boot is the initial system startup.

Note

None

XStatus XPm_RequestSuspend (const enum XPmNodeid *target*, const enum XPmRequestAck *ack*, const u32 *latency*, const u8 *state*)

This function is used by a PU to request suspend of another PU. This call triggers the power management controller to notify the PU identified by 'nodeID' that a suspend has been requested. This will allow said PU to gracefully suspend itself by calling XPm_SelfSuspend for each of its CPU nodes, or else call XPm_AbortSuspend with its PU node as argument and specify the reason.

Parameters

<i>target</i>	Node ID of the PU node to be suspended
<i>ack</i>	Requested acknowledge type. REQUEST_ACK_BLOCKING is not supported.
<i>latency</i>	Maximum wake-up latency requirement in us(micro sec)
<i>state</i>	Instead of specifying a maximum latency, a PU can also explicitly request a certain power state.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

If 'ack' is set to PM_ACK_NON_BLOCKING, the requesting PU will be notified upon completion of suspend or if an error occurred, such as an abort. REQUEST_ACK_BLOCKING is not supported for this command.

XStatus XPm_SelfSuspend (const enum XPmNodeid *nid*, const u32 *latency*, const u8 *state*, const u64 *address*)

This function is used by a CPU to declare that it is about to suspend itself. After the PMU processes this call it will wait for the requesting CPU to complete the suspend procedure and become ready to be put into a sleep state.

Parameters

<i>nid</i>	Node ID of the CPU node to be suspended.
<i>latency</i>	Maximum wake-up latency requirement in us(microsecs)
<i>state</i>	Instead of specifying a maximum latency, a CPU can also explicitly request a certain power state.
<i>address</i>	Address from which to resume when woken up.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

This is a blocking call, it will return only once PMU has responded

XStatus XPm_ForcePowerDown (const enum XPmNodeId *target*, const enum XPmRequestAck *ack*)

One PU can request a forced poweroff of another PU or its power island or power domain. This can be used for killing an unresponsive PU, in which case all resources of that PU will be automatically released.

Parameters

<i>target</i>	Node ID of the PU node or power island/domain to be powered down.
<i>ack</i>	Requested acknowledge type

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

Force power down may not be requested by a PU for itself.

XStatus XPm_AbortSuspend (const enum XPmAbortReason *reason*)

This function is called by a CPU after a XPm_SelfSuspend call to notify the power management controller that CPU has aborted suspend or in response to an init suspend request when the PU refuses to suspend.

Parameters

<i>reason</i>	Reason code why the suspend can not be performed or completed <ul style="list-style-type: none"> • ABORT_REASON_WKUP_EVENT : local wakeup-event received • ABORT_REASON_PU_BUSY : PU is busy • ABORT_REASON_NO_PWRDN : no external powerdown supported • ABORT_REASON_UNKNOWN : unknown error during suspend procedure
---------------	--

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

Calling PU expects the PMU to abort the initiated suspend procedure. This is a non-blocking call without any acknowledge.

XStatus XPm_RequestWakeUp (const enum XPmNodeId *target*, const bool *setAddress*, const u64 *address*, const enum XPmRequestAck *ack*)

This function can be used to request power up of a CPU node within the same PU, or to power up another PU.

Parameters

<i>target</i>	Node ID of the CPU or PU to be powered/woken up.
<i>setAddress</i>	Specifies whether the start address argument is being passed. <ul style="list-style-type: none"> • 0 : do not set start address • 1 : set start address
<i>address</i>	Address from which to resume when woken up. Will only be used if <i>set_address</i> is 1.
<i>ack</i>	Requested acknowledge type

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

If acknowledge is requested, the calling PU will be notified by the power management controller once the wake-up is completed.

XStatus XPm_SetWakeUpSource (const enum XPmNodeId *target*, const enum XPmNodeId *wkup_node*, const u8 *enable*)

This function is called by a PU to add or remove a wake-up source prior to going to suspend. The list of wake sources for a PU is automatically cleared whenever the PU is woken up or when one of its CPUs aborts the suspend procedure.

Parameters

<i>target</i>	Node ID of the target to be woken up.
<i>wkup_node</i>	Node ID of the wakeup device.
<i>enable</i>	Enable flag: <ul style="list-style-type: none"> • 1 : the wakeup source is added to the list • 0 : the wakeup source is removed from the list

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

Declaring a node as a wakeup source will ensure that the node will not be powered off. It also will cause the PMU to configure the GIC Proxy accordingly if the FPD is powered off.

XStatus XPm_SystemShutdown (u32 *type*, u32 *subtype*)

This function can be used by a privileged PU to shut down or restart the complete device.

Parameters

<i>restart</i>	Should the system be restarted automatically? <ul style="list-style-type: none"> • PM_SHUTDOWN : no restart requested, system will be powered off permanently • PM_RESTART : restart is requested, system will go through a full reset
----------------	--

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

In either case the PMU will call `XPm_InitSuspendCb` for each of the other PUs, allowing them to gracefully shut down. If a PU is asleep it will be woken up by the PMU. The PU making the `XPm_SystemShutdown` should perform its own suspend procedure after calling this API. It will not receive an init suspend callback.

XStatus XPm_SetConfiguration (const u32 *address*)

This function is called to configure the power management framework. The call triggers power management controller to load the configuration object and configure itself according to the content of the object.

Parameters

<i>address</i>	Start address of the configuration object
----------------	---

Returns

`XST_SUCCESS` if successful, otherwise an error code

Note

The provided address must be in 32-bit address space which is accessible by the PMU.

XStatus XPm_InitFinalize (void)

This function is called to notify the power management controller about the completed power management initialization.

Returns

`XST_SUCCESS` if successful, otherwise an error code

Note

It is assumed that all used nodes are requested when this call is made. The power management controller may power down the nodes which are not requested after this call is processed.

void XPm_InitSuspendCb (const enum XPmSuspendReason *reason*, const u32 *latency*, const u32 *state*, const u32 *timeout*)

Callback function to be implemented in each PU, allowing the power management controller to request that the PU suspend itself.

Parameters

<i>reason</i>	Suspend reason: <ul style="list-style-type: none"> • SUSPEND_REASON_PU_REQ : Request by another PU • SUSPEND_REASON_ALERT : Unrecoverable SysMon alert • SUSPEND_REASON_SHUTDOWN : System shutdown • SUSPEND_REASON_RESTART : System restart
<i>latency</i>	Maximum wake-up latency in us(micro secs). This information can be used by the PU to decide what level of context saving may be required.
<i>state</i>	Targeted sleep/suspend state.
<i>timeout</i>	Timeout in ms, specifying how much time a PU has to initiate its suspend procedure before it's being considered unresponsive.

Returns

None

Note

If the PU fails to act on this request the power management controller or the requesting PU may choose to employ the forceful power down option.

void XPm_AcknowledgeCb (const enum XPmNodeId *node*, const XStatus *status*, const u32 *oppoint*)

This function is called by the power management controller in response to any request where an acknowledge callback was requested, i.e. where the 'ack' argument passed by the PU was REQUEST_ACK_NON_BLOCKING.

Parameters

<i>node</i>	ID of the component or sub-system in question.
<i>status</i>	Status of the operation: <ul style="list-style-type: none"> • OK: the operation completed successfully • ERR: the requested operation failed
<i>oppoint</i>	Operating point of the node in question

Returns

None

Note

None

void XPm_NotifyCb (const enum XPmNodeId *node*, const u32 *event*, const u32 *oppoint*)

This function is called by the power management controller if an event the PU was registered for has occurred. It will populate the notifier data structure passed when calling XPm_RegisterNotifier.

Parameters

<i>node</i>	ID of the node the event notification is related to.
<i>event</i>	ID of the event
<i>oppoint</i>	Current operating state of the node.

Returns

None

Note

None

XStatus XPm_RequestNode (const enum XPmNodeId *node*, const u32 *capabilities*, const u32 *qos*, const enum XPmRequestAck *ack*)

Used to request the usage of a PM-slave. Using this API call a PU requests access to a slave device and asserts its requirements on that device. Provided the PU is sufficiently privileged, the PMU will enable access to the memory mapped region containing the control registers of that device. For devices that can only be serving a single PU, any other privileged PU will now be blocked from accessing this device until the node is released.

Parameters

<i>node</i>	Node ID of the PM slave requested
<i>capabilities</i>	Slave-specific capabilities required, can be combined <ul style="list-style-type: none"> PM_CAP_ACCESS : full access / functionality PM_CAP_CONTEXT : preserve context PM_CAP_WAKEUP : emit wake interrupts
<i>qos</i>	Quality of Service (0-100) required
<i>ack</i>	Requested acknowledge type

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

None

XStatus XPm_ReleaseNode (const enum XPmNodeId *node*)

This function is used by a PU to release the usage of a PM slave. This will tell the power management controller that the node is no longer needed by that PU, potentially allowing the node to be placed into an inactive state.

Parameters

<i>node</i>	Node ID of the PM slave.
-------------	--------------------------

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

None

XStatus XPm_SetRequirement (const enum XPmNodeId *nid*, const u32 *capabilities*, const u32 *qos*, const enum XPmRequestAck *ack*)

This function is used by a PU to announce a change in requirements for a specific slave node which is currently in use.

Parameters

<i>nid</i>	Node ID of the PM slave.
<i>capabilities</i>	Slave-specific capabilities required.
<i>qos</i>	Quality of Service (0-100) required.
<i>ack</i>	Requested acknowledge type

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

If this function is called after the last awake CPU within the PU calls SelfSuspend, the requirement change shall be performed after the CPU signals the end of suspend to the power management controller, (e.g. WFI interrupt).

XStatus XPm_SetMaxLatency (const enum XPmNodeid *node*, const u32 *latency*)

This function is used by a PU to announce a change in the maximum wake-up latency requirements for a specific slave node currently used by that PU.

Parameters

<i>node</i>	Node ID of the PM slave.
<i>latency</i>	Maximum wake-up latency required.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

Setting maximum wake-up latency can constrain the set of possible power states a resource can be put into.

XStatus XPm_GetApiVersion (u32 * *version*)

This function is used to request the version number of the API running on the power management controller.

Parameters

<i>version</i>	Returns the API 32-bit version number. Returns 0 if no PM firmware present.
----------------	---

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

None

XStatus XPm_GetNodeStatus (const enum XPmNodeId *node*, XPm_NodeStatus *const *nodestatus*)

This function is used to obtain information about the current state of a component. The caller must pass a pointer to an [XPm_NodeStatus](#) structure, which must be pre-allocated by the caller.

Parameters

<i>node</i>	ID of the component or sub-system in question.
<i>nodestatus</i>	Used to return the complete status of the node.

- status - The current power state of the requested node.
 - For CPU nodes:
 - 0 : if CPU is powered down,
 - 1 : if CPU is active (powered up),
 - 2 : if CPU is suspending (powered up)
 - For power islands and power domains:
 - 0 : if island is powered down,
 - 1 : if island is powered up
 - For PM slaves:
 - 0 : if slave is powered down,
 - 1 : if slave is powered up,
 - 2 : if slave is in retention
- requirement - Slave nodes only: Returns current requirements the requesting PU has requested of the node.
- usage - Slave nodes only: Returns current usage status of the node:
 - 0 : node is not used by any PU,
 - 1 : node is used by caller exclusively,
 - 2 : node is used by other PU(s) only,
 - 3 : node is used by caller and by other PU(s)

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

None

XStatus XPm_RegisterNotifier (XPm_Notifier *const *notifier*)

A PU can call this function to request that the power management controller call its notify callback whenever a qualifying event occurs. One can request to be notified for a specific or any event related to a specific node.

Parameters

<i>notifier</i>	Pointer to the notifier object to be associated with the requested notification. The notifier object contains the following data related to the notification:
-----------------	---

- nodeID : ID of the node to be notified about,
- eventID : ID of the event in question, '-1' denotes all events (- EVENT_STATE_CHANGE, EVENT_ZERO_USERS),
- wake : true: wake up on event, false: do not wake up (only notify if awake), no buffering/queueing
- callback : Pointer to the custom callback function to be called when the notification is available. The callback executes from interrupt context, so the user must take special care when implementing the callback. Callback is optional, may be set to NULL.
- received : Variable indicating how many times the notification has been received since the notifier is registered.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

The caller shall initialize the notifier object before invoking the XPm_RegisterNotifier function. While notifier is registered, the notifier object shall not be modified by the caller.

XStatus XPm_UnregisterNotifier (XPm_Notifier *const *notifier*)

A PU calls this function to unregister for the previously requested notifications.

Parameters

<i>notifier</i>	Pointer to the notifier object associated with the previously requested notification
-----------------	--

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

None

XStatus XPm_GetOpCharacteristic (const enum XPmNodeId *node*, const enum XPmOpCharType *type*, u32 *const *result*)

Call this function to request the power management controller to return information about an operating characteristic of a component.

Parameters

<i>node</i>	ID of the component or sub-system in question.
<i>type</i>	Type of operating characteristic requested: <ul style="list-style-type: none"> • power (current power consumption), • latency (current latency in us to return to active state), • temperature (current temperature),
<i>result</i>	Used to return the requested operating characteristic.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

None

XStatus XPm_ResetAssert (const enum XPmReset *reset*, const enum XPmResetAction *assert*)

This function is used to assert or release reset for a particular reset line. Alternatively a reset pulse can be requested as well.

Parameters

<i>reset</i>	ID of the reset line
<i>assert</i>	Identifies action: <ul style="list-style-type: none"> • PM_RESET_ACTION_RELEASE : release reset, • PM_RESET_ACTION_ASSERT : assert reset, • PM_RESET_ACTION_PULSE : pulse reset,

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

None

XStatus XPm_ResetGetStatus (const enum XPmReset *reset*, u32 * *status*)

Call this function to get the current status of the selected reset line.

Parameters

<i>reset</i>	Reset line
<i>status</i>	Status of specified reset (true - asserted, false - released)

Returns

Returns 1/XST_FAILURE for 'asserted' or 0/XST_SUCCESS for 'released'.

Note

None

XStatus XPm_MmioWrite (const u32 *address*, const u32 *mask*, const u32 *value*)

Call this function to write a value directly into a register that isn't accessible directly, such as registers in the clock control unit. This call is bypassing the power management logic. The permitted addresses are subject to restrictions as defined in the PCW configuration.

Parameters

<i>address</i>	Physical 32-bit address of memory mapped register to write to.
<i>mask</i>	32-bit value used to limit write to specific bits in the register.
<i>value</i>	Value to write to the register bits specified by the mask.

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

If the access isn't permitted this function returns an error code.

XStatus XPm_MmioRead (const u32 *address*, u32 *const *value*)

Call this function to read a value from a register that isn't accessible directly. The permitted addresses are subject to restrictions as defined in the PCW configuration.

Parameters

<i>address</i>	Physical 32-bit address of memory mapped register to read from.
<i>value</i>	Returns the 32-bit value read from the register

Returns

XST_SUCCESS if successful else XST_FAILURE or an error code or a reason code

Note

If the access isn't permitted this function returns an error code.

XStatus XPm_ClockEnable (const enum XPmClock *clock*)

Call this function to enable (activate) a clock.

Parameters

<i>clock</i>	Identifier of the target clock to be enabled
--------------	--

Returns

Status of performing the operation as returned by the PMU-FW

Note

If the access isn't permitted this function returns an error code.

XStatus XPm_ClockDisable (const enum XPmClock *clock*)

Call this function to disable (gate) a clock.

Parameters

<i>clock</i>	Identifier of the target clock to be disabled
--------------	---

Returns

Status of performing the operation as returned by the PMU-FW

Note

If the access isn't permitted this function returns an error code.

XStatus XPm_ClockGetStatus (const enum XPmClock *clock*, u32 *const *status*)

Call this function to get status of a clock gate state.

Parameters

<i>clock</i>	Identifier of the target clock
<i>status</i>	Location to store clock gate state (1=enabled, 0=disabled)

Returns

Status of performing the operation as returned by the PMU-FW

XStatus XPm_ClockGetDivider (const enum XPmClock *clock*, u32 *const *divider*)

Call this function to get divider of a clock.

Parameters

<i>clock</i>	Identifier of the target clock
<i>divider</i>	Location to store the divider value

Returns

XST_INVALID_PARAM or status of performing the operation as returned by the PMU-FW

XStatus XPm_ClockSetParent (const enum XPmClock *clock*, const enum XPmClock *parent*)

Call this function to set parent for a clock.

Parameters

<i>clock</i>	Identifier of the target clock
<i>parent</i>	Identifier of the target parent clock

Returns

XST_INVALID_PARAM or status of performing the operation as returned by the PMU-FW.

Note

If the access isn't permitted this function returns an error code.

XStatus XPm_ClockGetParent (const enum XPmClock *clock*, enum XPmClock *const *parent*)

Call this function to get parent of a clock.

Parameters

<i>clock</i>	Identifier of the target clock
<i>parent</i>	Location to store clock parent ID

Returns

XST_INVALID_PARAM or status of performing the operation as returned by the PMU-FW.

XStatus XPm_ClockSetRate (const enum XPmClock *clock*, const u32 *rate*)

Call this function to set rate of a clock.

Parameters

<i>clock</i>	Identifier of the target clock
<i>rate</i>	Clock frequency (rate) to be set

Returns

Status of performing the operation as returned by the PMU-FW

Note

Interface exists but the functionality is not supported in the current release.

XStatus XPm_ClockGetRate (const enum XPmClock *clock*, u32 **const rate*)

Call this function to get rate of a clock.

Parameters

<i>clock</i>	Identifier of the target clock
<i>rate</i>	Location where the rate should be stored

Returns

Status of performing the operation as returned by the PMU-FW

Note

Interface exists but the functionality is not supported in the current release.

XStatus XPm_PllSetParameter (const enum XPmNodeId *node*, const enum XPmPllParam *parameter*, const u32 *value*)

Call this function to set a PLL parameter.

Parameters

<i>node</i>	PLL node identifier
<i>parameter</i>	PLL parameter identifier
<i>value</i>	Value of the PLL parameter

Returns

Status of performing the operation as returned by the PMU-FW

Note

If the access isn't permitted this function returns an error code.

XStatus XPm_PllGetParameter (const enum XPmNodeId *node*, const enum XPmPllParam *parameter*, u32 *const *value*)

Call this function to get a PLL parameter.

Parameters

<i>node</i>	PLL node identifier
<i>parameter</i>	PLL parameter identifier
<i>value</i>	Location to store value of the PLL parameter

Returns

Status of performing the operation as returned by the PMU-FW

XStatus XPm_PllSetMode (const enum XPmNodeId *node*, const enum XPmPllMode *mode*)

Call this function to set a PLL mode.

Parameters

<i>node</i>	PLL node identifier
<i>mode</i>	PLL mode to be set

Returns

Status of performing the operation as returned by the PMU-FW

Note

If the access isn't permitted this function returns an error code.

XStatus XPm_PllGetMode (const enum XPmNodeId *node*, enum XPmPllMode *const *mode*)

Call this function to get a PLL mode.

Parameters

<i>node</i>	PLL node identifier
<i>mode</i>	Location to store the PLL mode

Returns

Status of performing the operation as returned by the PMU-FW

XStatus XPm_PinCtrlRequest (const u32 *pin*)

Call this function to request a pin control.

Parameters

<i>pin</i>	PIN identifier (index from range 0-77)
------------	--

Returns

Status of performing the operation as returned by the PMU-FW

XStatus XPm_PinCtrlRelease (const u32 *pin*)

Call this function to release a pin control.

Parameters

<i>pin</i>	PIN identifier (index from range 0-77)
------------	--

Returns

Status of performing the operation as returned by the PMU-FW

XStatus XPm_PinCtrlSetFunction (const u32 *pin*, const enum XPmPinFn *fn*)

Call this function to set a pin function.

Parameters

<i>pin</i>	Pin identifier
<i>fn</i>	Pin function to be set

Returns

Status of performing the operation as returned by the PMU-FW

Note

If the access isn't permitted this function returns an error code.

XStatus XPm_PinCtrlGetFunction (const u32 *pin*, enum XPmPinFn *const *fn*)

Call this function to get currently configured pin function.

Parameters

<i>pin</i>	PLL node identifier
<i>fn</i>	Location to store the pin function

Returns

Status of performing the operation as returned by the PMU-FW

XStatus XPm_PinCtrlSetParameter (const u32 *pin*, const enum XPmPinParam *param*, const u32 *value*)

Call this function to set a pin parameter.

Parameters

<i>pin</i>	Pin identifier
<i>param</i>	Pin parameter identifier
<i>value</i>	Value of the pin parameter to set

Returns

Status of performing the operation as returned by the PMU-FW

Note

If the access isn't permitted this function returns an error code.

XStatus XPm_PinCtrlGetParameter (const u32 *pin*, const enum XPmPinParam *param*, u32 *const *value*)

Call this function to get currently configured value of pin parameter.

Parameters

<i>pin</i>	Pin identifier
<i>param</i>	Pin parameter identifier
<i>value</i>	Location to store value of the pin parameter

Returns

Status of performing the operation as returned by the PMU-FW

Error Status

Overview

This section lists the Power management specific return error statuses.

Macros

- #define [XST_PM_INTERNAL](#) 2000L
- #define [XST_PM_CONFLICT](#) 2001L
- #define [XST_PM_NO_ACCESS](#) 2002L
- #define [XST_PM_INVALID_NODE](#) 2003L
- #define [XST_PM_DOUBLE_REQ](#) 2004L
- #define [XST_PM_ABORT_SUSPEND](#) 2005L
- #define [XST_PM_TIMEOUT](#) 2006L
- #define [XST_PM_NODE_USED](#) 2007L

Macro Definition Documentation

#define XST_PM_INTERNAL 2000L

An internal error occurred while performing the requested operation

#define XST_PM_CONFLICT 2001L

Conflicting requirements have been asserted when more than one processing cluster is using the same PM slave

#define XST_PM_NO_ACCESS 2002L

The processing cluster does not have access to the requested node or operation

#define XST_PM_INVALID_NODE 2003L

The API function does not apply to the node passed as argument



#define XST_PM_DOUBLE_REQ 2004L

A processing cluster has already been assigned access to a PM slave and has issued a duplicate request for that PM slave

#define XST_PM_ABORT_SUSPEND 2005L

The target processing cluster has aborted suspend

#define XST_PM_TIMEOUT 2006L

A timeout occurred while performing the requested operation

#define XST_PM_NODE_USED 2007L

Slave request cannot be granted since node is non-shareable and used

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

Solution Centers

See the [Xilinx Solution Centers](#) for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <http://www.xilinx.com/legal.htm#tos>.



Automotive Applications Disclaimer

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

© Copyright 2018 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.