Xilinx Software Development Kit (SDK) Standalone Library Documentation

Xilinx Power Management (XilPM) v2.0

UG1225 (2016.3) October 5, 2016

XILINX
ALL PROGRAMMABLE™

Table of Contents

XILINX
ALL PROGRAMMABLE™

Chapter 1

Xilinx Power Management (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).

Data Structures

- struct XPm_Notifier
- struct XPm_NodeStatus

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)

Functions

- XStatus XPm_InitXilpm (XIpiPsu *IpiInst)
- void XPm_SuspendFinalize ()
- enum XPmBootStatus XPm_GetBootStatus ()
- XStatus XPm_RequestSuspend (const enum XPmNodeId node, const enum XPmRequestAck ack, const u32 latency, const u8 state)
- XStatus XPm_SelfSuspend (const enum XPmNodeId node, const u32 latency, const u8 state, const u64 address)
- XStatus XPm_ForcePowerDown (const enum XPmNodeId node, const enum XPmRequestAck ack)
- XStatus XPm_AbortSuspend (const enum XPmAbortReason reason)
- XStatus XPm_RequestWakeUp (const enum XPmNodeId node, 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 (const u8 restart)
- 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 node, 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)

Function Documentation

XStatus XPm_InitXilpm (XIpiPsu * *IpiInst*)

`#include <pm_api_sys.h>`
Initialize xilpm library.

Parameters

IpiInst	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)

`#include <pm_api_sys.h>`
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 XPmBootStatus XPm_GetBootStatus (void)

`#include <pm_api_sys.h>`
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*)

`#include <pm_api_sys.h>`
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

target	Node ID of the PU node to be suspended
ack	Requested acknowledge type
latency	Maximum wake-up latency requirement in us(micro sec)
state	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_CB_STANDARD, the requesting PU will be notified upon completion of suspend or if an error occurred, such as an abort or a timeout.

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

`#include <pm_api_sys.h>`

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

nid	Node ID of the CPU node to be suspended.
latency	Maximum wake-up latency requirement in us(microsecs)
state	Instead of specifying a maximum latency, a CPU can also explicitly request a certain power state.
address	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*)

`#include <pm_api_sys.h>`

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

target	Node ID of the PU node or power island/domain to be powered down.
ack	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*)

`#include <pm_api_sys.h>`
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

reason	Reason code why the suspend can not be performed or completed • 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*)

```
#include <pm_api_sys.h>
```

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

Parameters

target	Node ID of the CPU or PU to be powered/woken up.
setAddress	Specifies whether the start address argument is being passed. • 0 : do not set start address • 1 : set start address
address	Address from which to resume when woken up. Will only be used if set_address is 1.
ack	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*)

```
#include <pm_api_sys.h>
```

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

target	Node ID of the target to be woken up.
wkup_node	Node ID of the wakeup device.
enable	Enable flag: • 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 (const u8 *restart*)

```
#include <pm_api_sys.h>
```

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

Parameters

restart	Should the system be restarted automatically? • 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.

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

```
#include <pm_api_sys.h>
```

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

Parameters

reason	Suspend reason: • SUSPEND_REASON_PU_REQ : Request by another PU • SUSPEND_REASON_ALERT : Unrecoverable SysMon alert • SUSPEND_REASON_SHUTDOWN : System shutdown • SUSPEND_REASON_RESTART : System restart
latency	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.
state	Targeted sleep/suspend state.
timeout	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*)

```
#include <pm_api_sys.h>
```

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_CB_STANDARD.

Parameters

node	ID of the component or sub-system in question.
status	Status of the operation: • OK: the operation completed successfully • ERR: the requested operation failed
oppoint	Operating point of the node in question

Returns

None

Note

None

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

```
#include <pm_api_sys.h>
```

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

node	ID of the node the event notification is related to.
event	ID of the event
oppoint	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*)

```
#include <pm_api_sys.h>
```

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

node	Node ID of the PM slave requested
capabilities	Slave-specific capabilities required, can be combined • PM_CAP_ACCESS : full access / functionality • PM_CAP_CONTEXT : preserve context • PM_CAP_WAKEUP : emit wake interrupts
qos	Quality of Service (0-100) required
ack	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*)

`#include <pm_api_sys.h>`
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

node	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*)

`#include <pm_api_sys.h>`
This function is used by a PU to announce a change in requirements for a specific slave node which is currently in use.

Parameters

nid	Node ID of the PM slave.
capabilities	Slave-specific capabilities required.
qos	Quality of Service (0-100) required.
ack	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*)

`#include <pm_api_sys.h>`
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

node	Node ID of the PM slave.
latency	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*)

`#include <pm_api_sys.h>`
This function is used to request the version number of the API running on the power management controller.

Parameters

version	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*)

```
#include <pm_api_sys.h>
```

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

node	ID of the component or sub-system in question.
nodestatus	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*)

```
#include <pm_api_sys.h>
```

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

notifier	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, EVENT_ERROR_CONDITION),
- 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_RegisteredNotifier function. While notifier is registered, the notifier object shall not be modified by the caller.

XStatus XPm_UnregisterNotifier (XPm_Notifier *const *notifier*)

`#include <pm_api_sys.h>`
A PU calls this function to unregister for the previously requested notifications.

Parameters

notifier	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 *const *result*)

`#include <pm_api_sys.h>`

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

Parameters

node	ID of the component or sub-system in question.
type	Type of operating characteristic requested: • power (current power consumption), • latency (current latency in us to return to active state), • temperature (current temperature),
result	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*)

```
#include <pm_api_sys.h>
```

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

Parameters

reset	ID of the reset line
assert	Identifies action: • 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*)

`#include <pm_api_sys.h>`
Call this function to get the current status of the selected reset line.

Parameters

reset	Reset line
status	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*)

`#include <pm_api_sys.h>`
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

address	Physical 32-bit address of memory mapped register to write to.
mask	32-bit value used to limit write to specific bits in the register.
value	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*)

```
#include <pm_api_sys.h>
```

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

address	Physical 32-bit address of memory mapped register to read from.
value	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.

Appendix A

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

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.