Texas Instruments

PAL SYS PCI Device Driver

Architecture/Design Document

April 2009

Platform Software Group

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address:
Texas Instruments
Post Office Box 655303, Dallas, Texas 75265

About This Document

This document discusses the TI device driver architecture for PAL SYS PCI Device Driver. The target audience includes device driver developers from TI as well as consumers of the driver.

Trademarks

Related Documents

- PCI controller specification

Notations

None

Terms and Abbreviations

Term	Description
IP	Intellectual Property
OS	Operating System
API	Application Programmer's Interface
CSL	Chip Support Library – TI primitive h/w abstraction
ISR	Interrupt Service Routine

Revision History

Date	Author	Comments	Version
20th January, 2007	Rinkal Shah	First draft	1.0
23rd March, 2007	Rinkal Shah	Document changed to reflect changes in driver due to new technical reference manual of PCI controller	1.1
4th June, 2007	Rinkal Shah	Soc C6452 added	1.2

October 22, 2007	Nagarjuna K	Updating for XDC and BIOS versions	1.3

Table of Contents

List of Figures

1 System Context

The PCI controller driver architecture presented in this document is running on DSP/BIOS operating system in master as well as slave mode.

1.1 Hardware

For detailed block diagram of module, please refer controller specification.

1.2 Software

The document provides an overall understanding of the TI PCI controller driver architecture.

1.2.1 Operating Environment and dependencies

Refer system level release notes for tools and BIOS versions.

1.3 Design Philosophy

Central to the philosophy of TI device driver architecture is clarity in separation of roles and responsibilities for the various parts of the device driver. Rather than treat the entire device driver as a monolithic block of code, effort is made to identify the portions of the device driver that are involved in:

- Coupling or handshaking with the specific OS
- Performing primitive, directed read/write access to the h/w device
- Modeling the crux of the driver behavior – protocol, state machine etc this in itself is regardless of any given OS.

With this view, the device driver functionality can be defined here under:

- OS Specific layer
- PAL SYS PCI layer.
- PAL SYS PCI register access layer.

There exists a clear separation of roles and responsibilities of these sub-components of the driver. The prescribed architecture helps in creating robust device drivers through tested/reusable pieces. Besides, it ensures in maintaining uniform semantics for similar services, supported across different drivers, for different platforms.

The figure below further elucidates the driving philosophy in partitioning the device driver into distinct functional sub-components – PAL SYS PCI layer, PAL SYS PCI register access layer, PAL SYS PCI OS dependent layer and CSL (Register layer only).

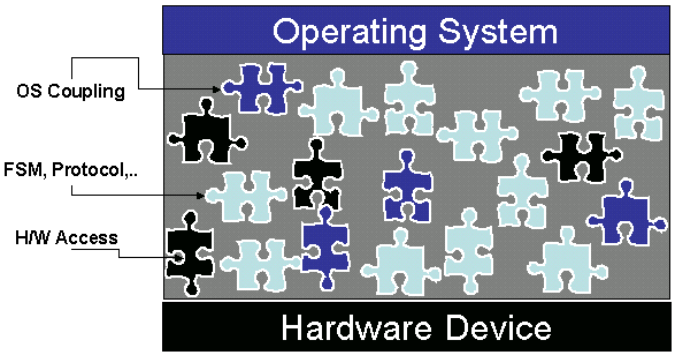

Figure-1: Monolithic Device Driver – Amorphous structure

1.3.1 Design Goals

The following are the key device driver design goals being factored by proposed TI architecture:

1. Uniformly styled drivers, promoting increased reuse and code familiarity to developers
2. Minimal overheads in integrating TI device driver into any Software System
3. Device driver must support Synchronous as well as Asynchronous interfaces to the user where appropriate
4. Device driver must operate both with and without Interrupts capability as appropriate
5. Device driver must leverage the H/W DMA capability where available to improve performance of device driver in case of block-oriented transfers

1.3.2 Assumptions

- It is assumed that TI Device Drivers are required to be fully functional in their native OS (BIOS in case of DSP side devices). Integration of TI Device Drivers into a given software system is beyond the immediate scope of the proposed architecture.

1.3.3 Design Principles

The guiding principles for the TI Device Driver design, drawn in the context of afore mentioned philosophy, goals and constraints can be enumerated as follows:

- Clear separation of H/W-dependent and H/W-independent parts. The H/W-dependent part - CSL must be lightweight. It should not make use of dynamic memory allocation scheme from within.

- Clear separation of OS dependent and independent parts
- Modular design to effectively address multiple device instances – avoids needless code size penalty and also to isolate device specific driver Implementation details from usage policies.

2 PCI Driver Software Architecture

This chapter deals with the overall architecture of TI PCI device driver, including the device driver partitioning as well as deployment considerations. We'll first examine the system decomposition into functional units and the interfaces presented by these units. Following this, we'll discuss the deployed driver or the dynamic view of the driver where the driver operational scenarios are presented.

2.1 Static View

2.1.1 Functional Decomposition

The device driver is partitioned into distinct sub-components, consistent with the roles and responsibilities already discussed in section 1.3. In the following sub-sections, each of these functional sub-components of the device driver is further elaborated.

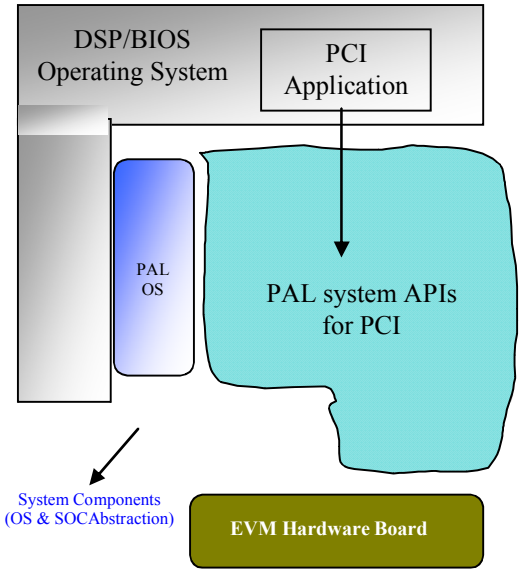

Figure-2: TI Device Driver Functional decomposition

The central portion shown constitutes the mainline PCI driver component; the surrounding modules constitute the supporting system components. The modules in the later part, do not specifically deal with PCI, but assist the driver by providing other abstracted (OS and H/W) and utility services as shall be discussed in following sub-sections.

2.1.2 PAL SYS PCI layer functionality

The DDC module forms the OS abstracted portion of the driver which provides basic behavior of the driver, modeling the main FSMs and Protocols. PCI does not make direct reference to any OS services, it glues to the OS via a well specified OS adaptation interface.

The PCI layer specifies a base set of interfaces that standardizes the common aspects of all device drivers such as – Initializing the driver (creation), terminating the driver (deletion) , opening the instance and closing the instance initialization and the basic data transfer operation.

In PCI layer, during the driver creation phase, relevant driver configuration parameters are obtained from the application.

Following table describes the APIs that are exposed to the application:

PAL SYS PCI layer APIs	DESCRIPTION
`PAL_Result PAL_sysPCICreate(Uint instId, Ptr param)`	Driver instance Creation
`PAL_Result PAL_sysPCIDelete(Uint instId)`	Driver instance deletion
`PSP_Handle PAL_sysPCIOpen(Uint instId, PAL_sysPCIInitConfig *param)`	Open driver instance
`PAL_Result PAL_sysPCIClose(PSP_Handle hPci, Ptr param)`	Close driver instance

PAL SYS PCI layer APIs	DESCRIPTION
PAL_Result PAL_sysPCIEnableInterrupt(PSP_Handle hPCI, Uint intEnableMask)	Enable interrupt depending upon the mask that is passed to the function. If a bit is set, corresponding interrupt is enabled
PAL_Result PAL_sysPCIDisableInterrupt(PSP_Handle hPCI, Uint intDisableMask)	Disable interrupt depending upon the mask that is passed to the function. If a bit is set, corresponding interrupt is disabled
PAL_Result PAL_sysPCISetMemMapReg(PAL_sysPciMemMapField memMapField, PAL_sysPciConfigFieldSize writeSize, Uint newFieldVal)	Set the specified memory mapped register. The offset of configuration register is specified PAL_sysPciMemMapField and the size of write operation is specified by PAL_sysPciConfigFieldSize. This size can be byte, word or dword.
PAL_Result PAL_sysPCIGetMemMapReg(PAL_sysPciMemMapField memMapField, PAL_sysPciConfigFieldSize readSize, Uint* newFieldVal)	Read the specified memory mapped space register. The offset of configuration register is specified PAL_sysPciMemMapField and the size of write operation is specified by PAL_sysPciConfigFieldSize. This size can be byte, word or dword.
PAL_Result PAL_sysPCISetHookConfigReg(PSP_Handle hPci, PAL_sysPciHookConfigField hookConfigField, PAL_sysPciConfigFieldSize writeSize, Uint newFieldVal)	Set the specified hook configuration register. The offset of configuration register is specified PAL_sysPCISetHookConfigReg and the size of write operation is specified by PAL_sysPciConfigFieldSize. This size can be byte, word or dword.
PAL_Result PAL_sysPCIGetHookConfigReg(PSP_Handle hPci, PAL_sysPciHookConfigField hookConfigField, PAL_sysPciConfigFieldSize writeSize, Uint *newFieldVal)	Read the specified hook configuration register. The offset of configuration register is specified PAL_sysPCISetHookConfigReg and the size of write operation is specified by PAL_sysPciConfigFieldSize. This size can be byte, word or dword.
PAL_Result PAL_sysPCIEnableBasePrefetch (PSP_Handle hPci, Uint32 baseId);	API to enable base address prefetching.
PAL_Result PAL_sysPCIDisableBasePrefetch (PSP_Handle hPci, Uint32 baseId);	API to disable base address prefetching.

PAL SYS PCI layer APIs	DESCRIPTION
PAL_Result PAL_sysPCIProgramCacheLineSize (PSP_Handle hPci, PAL_sysPciCacheLineSize valCacheLineSize);	API to configure the cache line size of PCI controller.
PAL_Result PAL_sysPCIProgramLatencyTimer (PSP_Handle hPci, Uint32 valLatencyTimer);	API to program Latency time value of PCI controller

2.1.3 OS Specific Adaptation

This layer “adapts” the driver core to the specific OS. It implements aspects such as – Interrupts registration and de-registration or providing the wrapper functions to PCI driver etc. This layer has full visibility to underlying OS services and is custom-built for a given OS.

2.1.4 Component Interfaces

In the following subsections, the interfaces implemented by each of the sub-component are specified.

2.1.4.1 OS specific Interface

The OS specific Interface constitutes the Device Driver Manifest to Application. This adapts the PCI driver to DSP BIOS.

2.1.4.2 PAL SYS PCI layer Interface

PCI layer forms the heart of Device driver also implements the interfaces exported to the application. The ISR would be registered thru’ the OS specific interface.

2.2 Dynamic View

2.2.1 The Execution Threads

The device driver is a service layer which provides access to configuration space registers, address translation registers, hook configuration registers, and memory mapped register of PCI controller when a device is acting as master. Device driver also provides access to configuration registers of slaves attached to the PCI bus in master mode of operation. For slave mode operation, this driver provides access to local configuration registers

2.2.2 Driver Initilization

Driver instance have to be created and opened before using the driver. This can be done by calling PAL_sysPCICreate and PAL_sysPCIOpen respectively.

3 Design goals

3.1 Initialization

Initialization of PCI device is from the host PC. From EVM side, software initialization of driver involves creation of protection semaphore and initialization of pointer to PCI configuration space.

3.2 Target device

When the system powers up, host configures all the target devices attached to the PCI bus. Host configures the base address registers of the target devices attached to PCI bus. Host can provide master access to any device.

This device can act as a master or a slave. Application can gain master control doing required setting for PCI controller.

3.3 Interrupts

There is a dedicated line for DSP (target) to host interrupt in PCI protocol. A target can interrupt the host through a dedicated interrupt line provide in the PCI bus. Driver provides access to a bit which can be set to interrupt the host device. Setting this bit asserts interrupt line and host is interrupted.

Host to DSP (target) interrupt line is not provided on PCI bus. But host can trigger a software interrupt by setting a bit in DSP's configuration space.

3.4 Address translation

There are 32 DSP to PCI address translation registers. These registers are used by PCI controller to translate the DSP address to an address on PCI bus. Each DSP to PCI address translation register corresponds to one master window. Each master window can map 8MB of PCI memory to its corresponding DSP's PCI memory address space through address translation register.

Below figure is the diagrammatic representation of address translation using DSP to PCI address translation register.

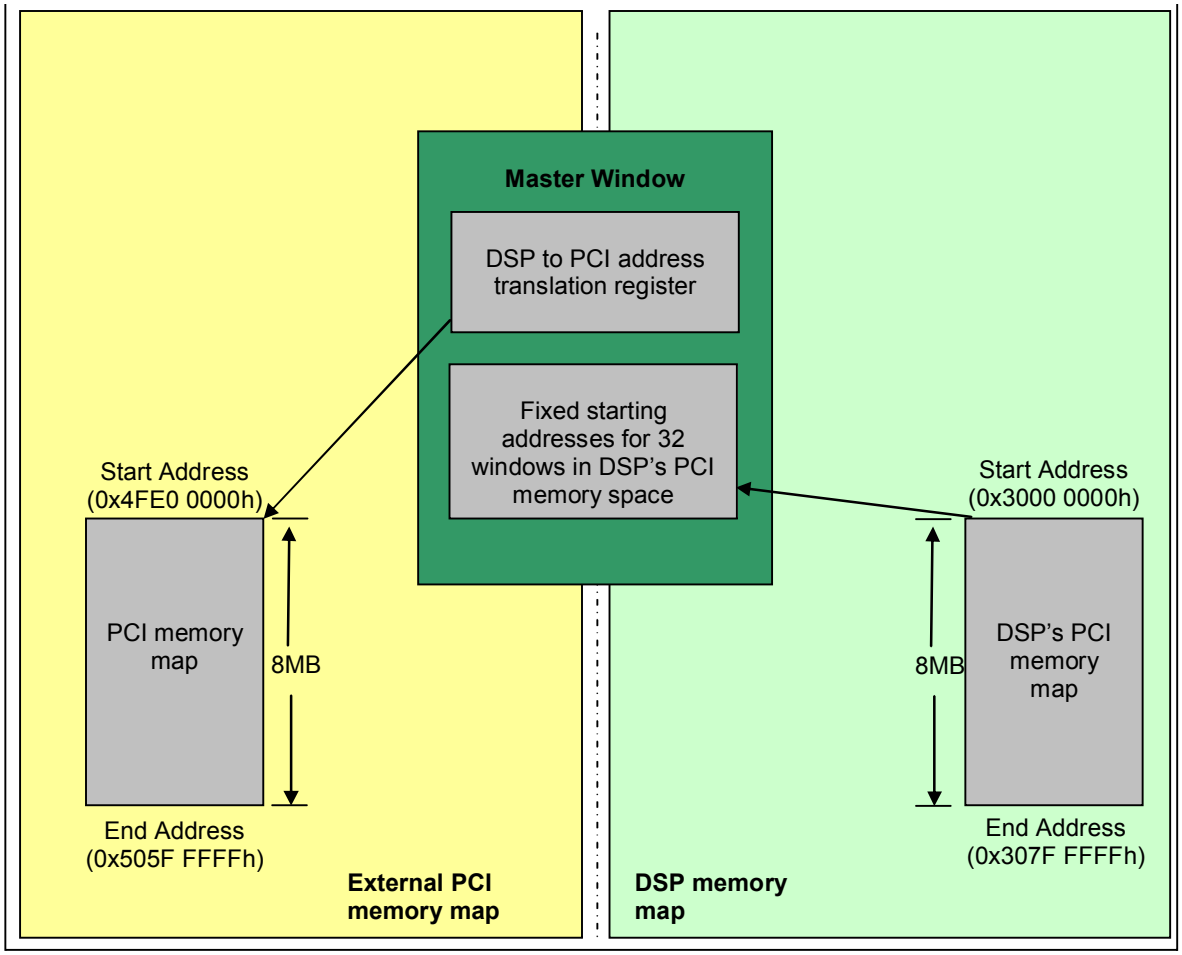

Figure 3: DSP to PCI address translation block diagram

Bits 31 to 23 of address translation register contain MSB of the PCI addresses within the corresponding window. The remaining reserved 23 bits are the size of the window and these bits have no function.

Address translation from the DSP to PCI domain is done using the address substitution registers 0 to 31.

During address translation the upper 4-bits (31 to 28) are used to reach to the DSP's PCI memory space. The next 5-bits decide which PCI window corresponds to the DSP address. Once the window is determined, the 23 LSBs of DSP address are allowed to pass through to the PCI address and the upper 9 bits are taken from the address translation register's 24 to 31 bits.

Below diagram illustrates the address translation process with an example address:

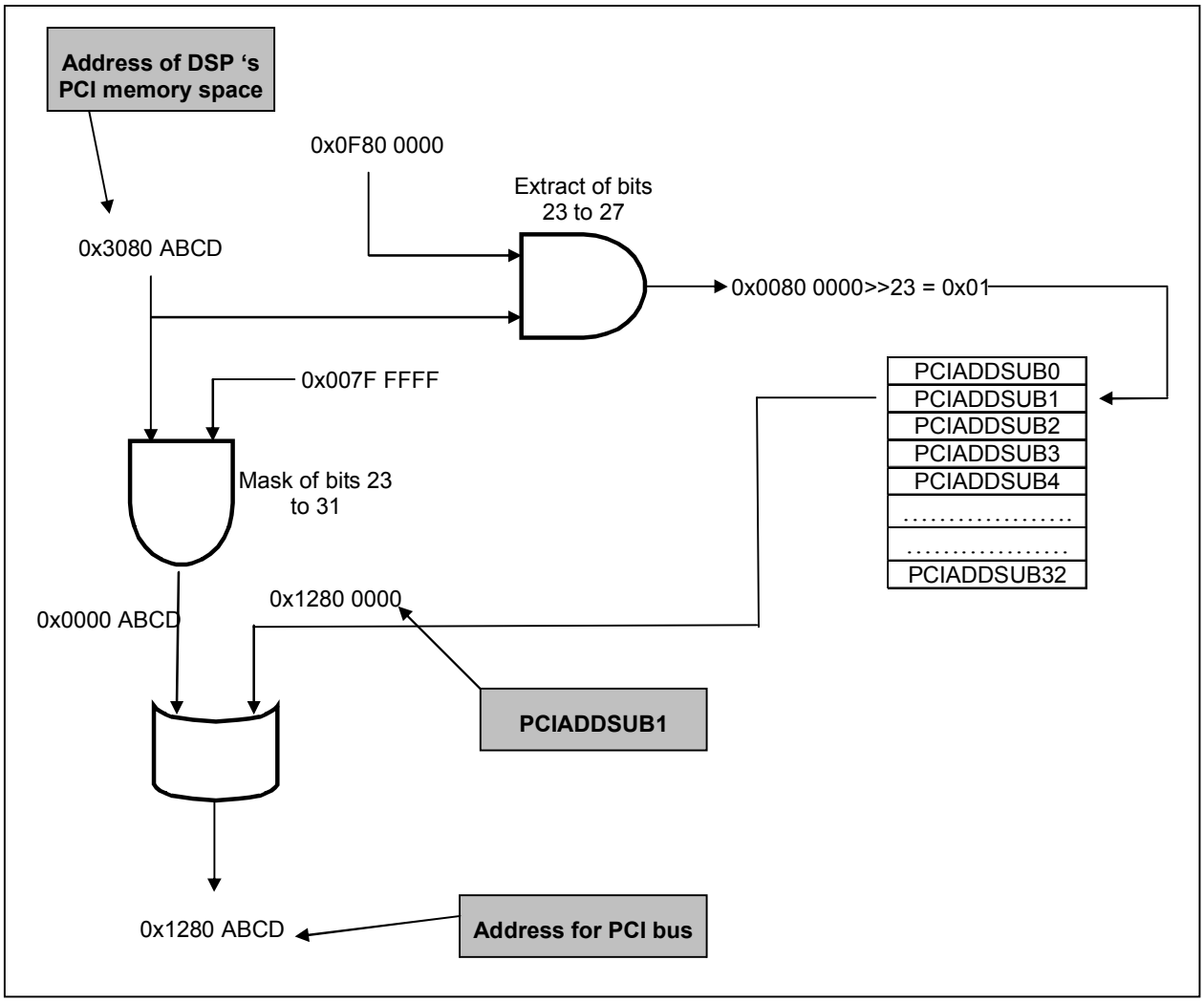

Figure 4: DSP to PCI address translation example

4 Application Programming Interface(s)

For information on data structures please refer to Appendix A.

4.1 Configuration API(s)

4.1.1 PAL_sysPCICreate ( ) - Create the instance of PCI driver.

PAL_Result	PAL_sysPCICreate	(	Uint32	instId,
			Ptr	param)

instId	Instance Id of PCI controller
param	Pointer parameters required for creating PCI driver handle
Returns	PAL_SOK on success or PAL Error Code on failure

Description:

PAL_sysPCICreate creates an instance of PCI driver. This function checks for the maximum instance number. If instance number is proper, it changes the state of driver. This function is implemented just to make PCI driver consistent with other PAL SYS drivers.

4.1.2 PAL_sysPCIDelete ( ) - Delete the instance of PCI driver.

PAL_Result	PAL_sysPCIDelete	(	Uint32	instId	)

instId	Instance Id of PCI controller
Returns	PAL_SOK on success or PAL Error Code on failure

Description:

PAL_sysPCIDelete deletes an instance of PCI driver. This function checks for the maximum instance number. If instance number is proper, it changes the state of driver. This function is implemented just to make PCI driver consistent with other PAL SYS drivers.

.

4.1.3 PAL_sysPCIOpen ( ) - Open the PCI driver instance.

PSP_Handle	PAL_sysPCIOpen	(	Uint32	instId,
			PAL_sysPciInitConfig	*param)

instId	Instance Id of PCI controller
param	Pointer parameters required for opening PCI driver handle. Application have to fill this param structure with proper values for initialization of driver
Returns	PSP_Handle on success or NULL on failure

Description:

This function opens the driver instance. If the handle is proper, it initializes the base address of PCI controller in the driver handle. This function also registers the interrupt callback for PCI interrupt. This callback will receive the interrupt status as one argument and application data pointer as other argument on occurrence of interrupt. Interrupt status is provided to application in application callback.

“param” should be a pointer to PAL_sysPCIinitConfig structure which should contain non-NULL value of application callback if application wants a callback on occurrence of interrupt. Application can also pass parameter to ISR. “param” contains one field as “appData”. “appData” should be pointing to the data that application wants to pass to callback function.

4.1.4 PAL_sysPCIClose ( ) - Close the PCI driver instance.

PAL_Result	PAL_sysPCIClose	(	Uint32	instId,
			Ptr	param)

instId	Instance Id of PCI controller
param	Pointer parameters required for closing PCI driver handle if any.
Returns	PAL_SOK on success or PAL Error Code on failure

Description:

This function closes the PCI driver instance. It checks the instance number and state of driver. If state and instance number are proper, it removes the hooking of ISR with interrupt. The function also make pointer to base addresses NULL. After doing all this operations, this function changes the state of driver instance to CLOSED state.

4.1.5 PAL_sysPCIEnableInterrupt ( ) - Enables PCI interrupt/s.

PAL_Result	PAL_sysPCIEnableInterrupt	(	PSP_Handle	hPci,
			Uint32	intEnableCode)
		hPci	Handle of PCI controller driver	
		intEnableCode	Interrupt enable mask. 5 LSBs of intEnableCode will correspond to 5 interrupts of PCI. If a particular bit is set, corresponding interrupt would be enabled.	
		Returns	PAL_SOK on success or PAL Error Code on failure	

Description:

This function enables interrupt/s. Interrupts are enabled depending upon the mask that is passed as argument to this function. Each bit of “intEnableCode” signifies one interrupt. Particular bit have to be set by the application to enable corresponding interrupt. Multiple interrupts can be set by setting multiple bits in the “intEnableCode”.

4.1.6 PAL_sysPCIDisableInterrupt ( ) - Disables PCI interrupt/s.

PAL_Result	PAL_sysPCIEnableInterrupt	(	PSP_Handle	hPci,
			Uint32	intEnableCode)
		hPci	Handle of PCI controller driver	
		intDisableCode	Interrupt disable mask. 5 LSBs of intDisableCode will correspond to 5 interrupts of PCI. If a particular bit is set, corresponding interrupt would be disabled.	
		Returns	PAL_SOK on success or PAL Error Code on failure	

Description:

This function disables interrupt/s. Interrupts are disabled depending upon the mask that is passed as argument to this function. Each bit of “intDisableCode” signifies one interrupt. Particular bit have to be set by the application to disable corresponding interrupt. Multiple interrupts can be disabled by setting multiple bits in the “intDisableCode”.

4.1.7 PAL_sysPCISetMemMapReg ( ) - Set memory mapped configuration register.

PAL_Result	PAL_sysPCISetMemMapReg	(	PSP_Handle hPci,
			PAL_sysPciMemMapField memMapField,
			PAL_sysPciConfigFieldSize writeSize,
			Uint32 newFieldVal)
		hPci	Handle of PCI controller driver
		memMapField	Field of memory mapped register that has to be set
		writeSize	Size of data to be set (byte/word/dword)
		newFieldVal	New value of field that is to be set
		Returns	PAL_SOK on success or PAL Error Code on failure

Description:

This function sets values in memory mapped registers. Register whose value has to be set should be passed as “memMapField” argument to PAL_sysPCISetMemMapReg function. The values of “memMapField” are restricted to the values available in “PAL_sysPciMemMapField” enum. “writeSize” specifies the number of bytes to written on to the field. Number of bytes can be 1, 2 or 4. The value that has to be written to the field shall to be specified as “newFieldVal” argument.

4.1.8 PAL_sysPCIGetMemMapReg ( ) - Get memory mapped configuration register.

PAL_Result	PAL_sysPCIGetMemMapReg	(	PSP_Handle hPci,
			PAL_sysPciMemMapField memMapField ,
			PAL_sysPciConfigFieldSize readSize,
			Uint32 *newFieldVal)

hPci	Handle of PCI controller driver
memMapFieId	Field of memory mapped register that has to be set
readSize	Size of data to be read (byte/word/dword))
*newFieldVal	Pointer to new value of field that is to be read from hardware
Returns	PAL_SOK on success or PAL Error Code on failure

Description:

This function reads values in memory mapped registers. Register whose value has to be read should be passed as “memMapField” argument to PAL_sysPCISetMemMapReg function. The values of “memMapField” are restricted to the values available in “PAL_sysPciMemMapField” enum. “readSize” specifies the number of bytes to read from the specified field. Number of bytes can be 1, 2 or 4. The value that has to be read from the field will be store at 4 byte memory pointer by “newFieldVal” pointer.

4.1.9 PAL_sysPCISetHookReg( ) - Set Hook register.

PAL_Result PAL_sysPCISetHookReg	(	PSP_Handle	hPci,
		PAL_sysPciHookRegister	hookRegField,
		PAL_sysPciConfigFieldSize	writeSize,
		Uint32	newFieldVal)
	hPci	Handle of PCI controller driver	
	hookRegField	Field of hook register that has to be set	
	writeSize	Size of data to be set (byte/word/dword)	
	newFieldVal	New value of field that is to be set	
	Returns	PAL_SOK on success or PAL Error Code on failure	

Description:

This function sets values in hook registers. Register whose value has to be set should be passed as “hookRegField” argument to PAL_sysPCISetHookReg function. The values of “hookRegField” are restricted to the values available in “PAL_sysPciHookRegister” enum. “writeSize” specifies the number of bytes to written on to the field. Number of bytes can be 1, 2 or 4. The value that has to be written to the field shall be specified as “newFieldVal” argument.

4.1.10 PAL_sysPCIGetHookReg( ) - Get Hook register.

PAL_Result	PAL_sysPCIGetHookReg (	PSP_Handle	hPci,
		PAL_sysPciHookRegister	hookRegField,
		PAL_sysPciConfigFieldSize	readSize,
		Uint32	*newFieldVal)
	hPci	Handle of PCI controller driver	
	hookRegField	Field of hook register that has to be set	
	readSize	Size of data to be read (byte/word/dword)	
	*newFieldVal	Pointer to new value of field that is to be read from hardware	
	Returns	PAL_SOK on success or PAL Error Code on failure	

Description:

This function reads values from hook registers. Register whose value has to be read should be passed as “hookRegField” argument to PAL_sysPCISetHookReg function. The values of “hookRegField” are restricted to the values available in “PAL_sysPciHookRegister” enum. “readSize” specifies the number of bytes to read from the field. Number of bytes can be 1, 2 or 4. The value that is read from the field will be stored at the 4 byte memory location pointed by “newFieldVal” pointer.

4.1.11 PAL_sysPCIEnableBasePrefetch ( ) - API to Enable BasePrefetch.

PAL_Result	PAL_sysPCIEnableBasePrefetch	(	PSP_Handle	hPci,
			Uint32	baseId)
		hPci	Handle of PCI controller driver	
		baseId	ID of BASE for which prefetch have to be enabled	
		Returns	PAL_SOK on success or PAL Error Code on failure	

Description:

This function will enable pre-fetchable functionality of BAR memory space. “baseId” will specify the ID of BAR whose pre-fetchable functionality have to be enabled. The value of “baseId” can vary from 0 to 5.

4.1.12 PAL_sysPCIDisableBasePrefetch ( ) - API to Disable Base Prefetch.

PAL_Result	PAL_sysPCIDisableBasePrefetch	(	PSP_Handle	hPci,
			Uint32	baseId)
		hPci	Handle of PCI controller driver	
		baseId	ID of BASE for which pre-fetch have to be disabled	
		Returns	PAL_SOK on success or PAL Error Code on failure	

Description:

This function will disable pre-fetchable functionality of BAR memory space. “baseId” will specify the ID of BAR whose pre-fetchable functionality have to be disabled. The value of “baseId” can vary from 0 to 5.

4.1.13 PAL_sysPCIProgramCacheLineSize ( ) - API to set Cache line size.

PAL_Result	PAL_sysPCIDisableBasePrefetch	(	PSP_Handle	hPci,	
			PAL_sysPciCacheLineSize	valCacheLineSize	)
		hPci	Handle of PCI controller driver		
		valCacheLineSize	Size of cache line to be set		
		Returns	PAL_SOK on success or PAL Error Code on failure		

Description:

This function will program the length of cache line. “valCacheLineSize” will specify the new size of cache line. The values of “valCacheLineSize” can vary from 16 bytes, 32 bytes, 64 bytes, 128 bytes or cache can be disabled. The values of “valCacheLineSize” are restricted by “PAL_sysPciCacheLineSize” enum.

4.1.14 PAL_sysPCIProgramLatencyTimer ( ) - API to program Latency timer.

PAL_Result	PAL_sysPCIProgramLatencyTimer	(	PSP_Handle	hPci,
			Uint32	valLatencyTimer)
		hPci	Handle of PCI controller driver	
		valLatencyTimer	Value of latency timer	
		Returns	PAL_SOK on success or PAL Error Code on failure	

Description:

This function will set the value of latency timer. The new value of latency timer has to be specified as “valLatencyTimer” argument.

5 Appendix A

5.1 Enumerators and typedefs (pal_syspci.h)

```
typedef enum
{
       /** Read/write one byte */
       PAL_SYS_PCI_READ_WRITE_SIZE_BYTE             = 0u,
       /** Read/write two byte */
       PAL_SYS_PCI_READ_WRITE_SIZE_HALFWORD  = 1u,
       /** Read/write four bytes */
       PAL_SYS_PCI_READ_WRITE_SIZE_WORD             = 2u
}PAL_sysPciConfigFieldSize;

typedef enum
{
    /** status set register  */
    PAL_SYSPCI_STATUS_SET                   = 0x010u,
    /** status clear register  */
    PAL_SYSPCI_STATUS_CLEAR                 = 0x014u,
    /** Host interrupt enable set register  */
    PAL_SYSPCI_HOST_INT_ENABLE_SET          = 0x020u,
    /** Host interrupt enable clear register  */
    PAL_SYSPCI_HOST_INT_ENABLE_CLEAR        = 0x024u,
    /** DSP interrupt enable set register  */
    PAL_SYSPCI_DSP_INT_ENABLE_SET           = 0x030u,
    /** DSP interrupt enable clear register  */
    PAL_SYSPCI_DSP_INT_ENABLE_CLEAR         = 0x034u,

    /** Vendor ID mirror register  */
    PAL_SYSPCI_MIRROR_VENDOR_ID             = 0x100u,
    /** Device ID mirror register  */
    PAL_SYSPCI_MIRROR_DEVICE_ID             = 0x102u,
    /** Command mirror register  */
    PAL_SYSPCI_MIRROR_CMD_STATUS_REG        = 0x104u,
    /** Revision ID mirror register  */
    PAL_SYSPCI_MIRROR_REVISION_ID           = 0x108u,
    /** Class code mirror register  */
    PAL_SYSPCI_MIRROR_CLASS_CODE            = 0x109u,
    /** Cache size mirror register  */
    PAL_SYSPCI_MIRROR_CACHE_SIZE            = 0x10Cu,
    /** Latency timer mirror register  */
    PAL_SYSPCI_MIRROR_LATENCY_TIMER         = 0x10Du,
    /** Header type mirror register  */
    PAL_SYSPCI_MIRROR_HEADER_TYPE           = 0x10Eu,
    /** BIST mirror register  */
    PAL_SYSPCI_MIRROR_BIST_REG              = 0x10Fu,
    /** BAR 0 mask register  */
    PAL_SYSPCI_MASK_BAR_0                   = 0x110u,
    /** BAR 1 mask register  */
    PAL_SYSPCI_MASK_BAR_1                   = 0x114u,
    /** BAR 2 mask register  */
    PAL_SYSPCI_MASK_BAR_2                   = 0x118u,
    /** BAR 3 mask register  */
    PAL_SYSPCI_MASK_BAR_3                   = 0x11Cu,
    /** BAR 4 mask register  */
    PAL_SYSPCI_MASK_BAR_4                   = 0x120u,
    /** BAR 5 mask register  */
    PAL_SYSPCI_MASK_BAR_5                   = 0x124u,
    /** Subsytem vendor ID mirror register  */
```

```
    PAL_SYSPCI_MIRROR_SUBSYS_VENDOR_ID      = 0x12Cu,
    /** Subsytem ID mirror register  */
    PAL_SYSPCI_MIRROR_SUBSYS_ID             = 0x12Eu,
    /** Capabilities pointer mirror register  */
    PAL_SYSPCI_MIRROR_CAP_PTR_ID            = 0x134u,
    /** Interrupt line mirror register  */
    PAL_SYSPCI_MIRROR_INT_LINE              = 0x13Cu,
    /** Interrupt Pin mirror register  */
    PAL_SYSPCI_MIRROR_INT_PIN               = 0x13Du,
    /** Minimum grant bits mirror register  */
    PAL_SYSPCI_MIRROR_MIN_GRANT_BITS        = 0x13Eu,
    /** Maximum latency bits mirror register  */
    PAL_SYSPCI_MIRROR_MAX_LATENCY_BITS      = 0x13Fu,

    /** Slave control register  */
    PAL_SYSPCI_SLAVE_CNTL_REG                = 0x180u,

    /** Slave base address 0 translation register  */
    PAL_SYSPCI_SLAVE_BASE_ADDR_TRANS_REG_0   = 0x1C0u,
    /** Slave base address 1 translation register  */
    PAL_SYSPCI_SLAVE_BASE_ADDR_TRANS_REG_1   = 0x1C4u,
    /** Slave base address 2 translation register  */
    PAL_SYSPCI_SLAVE_BASE_ADDR_TRANS_REG_2   = 0x1C8u,
    /** Slave base address 3 translation register  */
    PAL_SYSPCI_SLAVE_BASE_ADDR_TRANS_REG_3   = 0x1CCu,
    /** Slave base address 4 translation register  */
    PAL_SYSPCI_SLAVE_BASE_ADDR_TRANS_REG_4   = 0x1D0u,
    /** Slave base address 5 translation register  */
    PAL_SYSPCI_SLAVE_BASE_ADDR_TRANS_REG_5   = 0x1D4u,

        /** Base address 0 mirror register  */
    PAL_SYSPCI_MIRROR_BAR_0                  = 0x1E0u,
    /** Base address 1 mirror register  */
    PAL_SYSPCI_MIRROR_BAR_1                  = 0x1E4u,
    /** Base address 2 mirror register  */
    PAL_SYSPCI_MIRROR_BAR_2                  = 0x1E8u,
    /** Base address 3 mirror register  */
    PAL_SYSPCI_MIRROR_BAR_3                  = 0x1ECu,
    /** Base address 4 mirror register  */
    PAL_SYSPCI_MIRROR_BAR_4                  = 0x1F0u,
    /** Base address 5 mirror register  */
    PAL_SYSPCI_MIRROR_BAR_5                  = 0x1F4u,

    /** Master configuration/IO access data register  */
    PAL_SYSPCI_MASTER_CONFIG_IO_DATA_REG     = 0x300u,
    /** Master configuration/IO access address register  */
    PAL_SYSPCI_MASTER_CONFIG_IO_ADDR_REG     = 0x304u,
    /** Master configuration/IO access command register  */
    PAL_SYSPCI_MASTER_CONFIG_IO_CMD_REG      = 0x308u,

    /** Master configuration register  */
    PAL_SYSPCI_MASTER_CONFIG_REG             = 0x310u,

    /** PCI Address Substitution register 0  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG0        = 0x314u,
    /** PCI Address Substitution register 1  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG1        = 0x318u,
    /** PCI Address Substitution register 2  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG2        = 0x31Cu,
    /** PCI Address Substitution register 3  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG3        = 0x320u,
    /** PCI Address Substitution register 4  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG4        = 0x324u,
    /** PCI Address Substitution register 5  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG5        = 0x328u,
    /** PCI Address Substitution register 6  */
```

```
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG6        = 0x32Cu,
    /** PCI Address Substitution register 7  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG7        = 0x330u,
    /** PCI Address Substitution register 8  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG8        = 0x334u,
    /** PCI Address Substitution register 9  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG9        = 0x338u,
    /** PCI Address Substitution register 10  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG10    = 0x33Cu,
    /** PCI Address Substitution register 11  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG11    = 0x340u,
    /** PCI Address Substitution register 12  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG12    = 0x344u,
    /** PCI Address Substitution register 13  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG13    = 0x348u,
    /** PCI Address Substitution register 14  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG14    = 0x34Cu,
    /** PCI Address Substitution register 15  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG15    = 0x350u,
    /** PCI Address Substitution register 16  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG16    = 0x354u,
    /** PCI Address Substitution register 17  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG17    = 0x358u,
    /** PCI Address Substitution register 18  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG18    = 0x35Cu,
    /** PCI Address Substitution register 19  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG19    = 0x360u,
    /** PCI Address Substitution register 20  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG20    = 0x364u,
    /** PCI Address Substitution register 21  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG21    = 0x368u,
    /** PCI Address Substitution register 22  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG22    = 0x36Cu,
    /** PCI Address Substitution register 23  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG23    = 0x370u,
    /** PCI Address Substitution register 24  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG24    = 0x374u,
    /** PCI Address Substitution register 25  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG25    = 0x378u,
    /** PCI Address Substitution register 26  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG26    = 0x37Cu,
    /** PCI Address Substitution register 27  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG27    = 0x380u,
    /** PCI Address Substitution register 28  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG28    = 0x384u,
    /** PCI Address Substitution register 29  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG29    = 0x388u,
    /** PCI Address Substitution register 30  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG30    = 0x38Cu,
    /** PCI Address Substitution register 31  */
    PAL_SYSPCI_ADDR_SUBSTITUTION_REG31    = 0x390u
}PAL_sysPciMemMapField;

typedef enum
{
    /** Vendor ID program register  */
    PAL_SYSPCI_HOOK_REG_PROGRAM_VENDOR_ID                        = 0x394u,
    /** Device ID program register */
    PAL_SYSPCI_HOOK_REG_PROGRAM_DEVICE_ID                        = 0x396u,

    /** Class code and revision ID program register  */
    PAL_SYSPCI_HOOK_REG_PROGRAM_CLASS_CODE_REVISION_ID           = 0x39Cu,

    /** Regsiter to program vendor ID  */
```

```
    PAL_SYSPCI_HOOK_REG_PROGRAM_SUB_SYS_VENDOR_ID_SUB_SYS_ID    = 0x3A0u,
    /** Max latency and min grant program register  */
    PAL_SYSPCI_HOOK_REG_PROGRAM_MAX_LATENCY_MIN_GRANT           = 0x3A4u,

    /** Configuration done registers  */
    PAL_SYSPCI_HOOK_REG_CONFIG_DONE_REG                         = 0x3ACu
}PAL_sysPciHookRegister;

typedef enum
{
        /** Cache line size - Disabled  */
        PAL_SYSPCI_CACHELINE_SIZE_DISABLED    = 0x00u,
        /** Cache line size - 16 bytes  */
        PAL_SYSPCI_CACHELINE_SIZE_16BYTES     = 0x04u,
        /** Cache line size - 32 bytes  */
        PAL_SYSPCI_CACHELINE_SIZE_32BYTES     = 0x08u,
        /** Cache line size - 64 bytes  */
        PAL_SYSPCI_CACHELINE_SIZE_64BYTES     = 0x10u,
        /** Cache line size - 128 bytes  */
        PAL_SYSPCI_CACHELINE_SIZE_128BYTES    = 0x20u
}PAL_sysPciCacheLineSize;
```

5.2 PAL_sysPciInitConfig (pal_syspci.h)

```
typedef struct _PAL_sysPciInitConfig
{
        /** Instance wide callback function to catch errors*/
        PAL_sysPciAppCallback appCb;
        /** Application data to be passed back to the app callback */
        Ptr appData;
} PAL_sysPciInitConfig;
```

6 Appendix B

```
/** Maximum PCI driver instance supported */
#define PAL_SYSPCI_MAX_PCI_INSTANCE           (1u)
/** Maximum number of slaves supported by PCI */
#define PAL_SYSPCI_MAX_NUM_SLAVES             (4u)
/** Maximum base address registers supported by PCI controller */
#define PAL_SYSPCI_MAX_BASE_ADDR              (6u)
/** Interrupt mask to enable/disable Target abort interrupt */
#define PAL_SYSPCI_TARGET_ABORT_INT_MASK     (0x00000002u)
/** Interrupt mask to enable/disable Master abort interrupt */
#define PAL_SYSPCI_MASTER_ABORT_INT_MASK     (0x00000004u)
/** Interrupt mask to enable/disable system error interrupt */
#define PAL_SYSPCI_SYS_ERR_INT_MASK          (0x00000020u)
/** Interrupt mask to enable/disable parity error interrupt */
#define PAL_SYSPCI_PARITY_ERR_INT_MASK       (0x00000040u)

/** PCI driver error base define */
#define PAL_SYS_PCI_ERR_BASE                 (-40)
/**
 * PCI Driver Object Not Deleted yet.
 * So the object cannot be created.
 */
#define PAL_SYS_PCI_OBJ_NOT_DELETED         (PAL_SYS_PCI_ERR_BASE)
/**
 * PCI Driver Object Not Created yet.
```

```
 * So the object cannot be deleted.
 */
#define PAL_SYS_PCI_OBJ_NOT_CREATED         (PAL_SYS_PCI_ERR_BASE-1)
/**
 * PCI Driver Object Not Closed yet.
 * So the object cannot be deleted.
 */
#define PAL_SYS_PCI_OBJ_NOT_CLOSED          (PAL_SYS_PCI_ERR_BASE-2)
/**
 * PCI Driver Not Opened yet
 * So the object cannot be closed.
 */
#define PAL_SYS_PCI_OBJ_NOT_OPENED          (PAL_SYS_PCI_ERR_BASE-3)
/** Invalid Parameter passed to API */
#define PAL_SYS_PCI_INVALID_PARAM           (PAL_SYS_PCI_ERR_BASE-4)
/** Error encountered in PCI driver while semaphore operation */
#define PAL_SYS_PCI_SEM_ERR                 (PAL_SYS_PCI_ERR_BASE-5)

/**
 * Code to enable/disable Target abort interrupt of Host
 */
#define PAL_SYS_PCI_HOST_INT_TGT_ABORT                   (0x01u)
/**
 * Code to enable/disable Master abort interrupt of Host
 */
#define PAL_SYS_PCI_HOST_INT_MS_ABORT                    (0x02u)
/**
 * Code to enable/disable system error detect interrupt of Host
 */
#define PAL_SYS_PCI_HOST_INT_SYS_ERR_DETECT              (0x04u)
/**
 * Code to enable/disable parity error detect interrupt of Host
 */
#define PAL_SYS_PCI_HOST_INT_PARITY_ERR_DETECT           (0x08u)



/**
 * Code to enable/disable target abort error interrupt of DSP application
 */
#define PAL_SYS_PCI_DSP_INT_TGT_ABORT            (0x10u)
/**
 * Code to enable/disable master abort error interrupt of DSP application
 */
#define PAL_SYS_PCI_DSP_INT_MS_ABORT             (0x20u)
/**
 * Code to enable/disable system error detect interrupt of DSP application
 */
#define PAL_SYS_PCI_DSP_INT_SYS_ERR_DETECT       (0x40u)
/**
 * Code to enable/disable parity error detect interrupt of DSP application
 */
#define PAL_SYS_PCI_DSP_INT_PARITY_ERR_DETECT    (0x80u)
```