ALPHI_IO(C32_PLX) DSP Library User's Guide

SOFTWARE REFERENCE

74501-004-4000
Revision B
01/25/01

ALPHI Technology Corp.
6202 S. Maple Ave., #120
Tempe, AZ 85283
Tel: (480) 838-2428
Fax: (480) 838-4477
info@alphitech.com
www.alphitech.com

Table of Contents

Description

This DSP input / output library allows a DSP program to make use of several features in a relatively independent manner. The features supported are as follows:

Use of the onboard serial port as a source for standard input and output.

Ability to easily utilize additional 8530 resources, such as those on SCC-04 IPs.

Ability to access card hardware, especially for access with the HOST, in a relatively independent manner.

The documentation is divided into the following sections.

Modules The source files required to build the DSP Library portion of this board support package. Also, any global variables are listed here. Full source is provided for these files in the DSP source library called ALPHI_IO.SRC. Use the TI supplied AR30 utility to extract the source files from this file as follows.

AR30 x alphi_io.src

Functions C functions implementing the functionality of this library.

Structures and Enumerations Data objects used in this library.

Constants and Typedefs Data types.

Defines Constant data.

Direct PCI Access by DSP (PLX)

The PLX chip can be configured to directly access HOST memory by correctly programming the registers. Several C routines are present in the alphi_io library (PLX Version) to accomplish this easily.

There are two significant tasks which can be accomplished when the DSP directly accesses PCI space. The first is when accessing a separate PCI/CPCI card, and the second is to make use of the HOST memory, shared with the HOST processor. The cases are discussed separately.

For the case where the DSP is accessing another card, several functions exist to help with programming the PLX passthru region. **Pci_MapPhysMemDword**, **Pci_MapPhysMemWord**, **Pci_MapPhysMemSint Pci_MapPhysMemByte**, and **Pci_MapPhysMemChar** map a region of DSP address space to PCI memory space. **Pci_MapPhysIoDword**, **Pci_MapPhysIoWord**, and **Pci_MapPhysIoByte** map a region of DSP address space to PCI I/O space. All these functions accept a physical PCI address (either memory or I/O) and return an appropriate DSP pointer through which the DSP can complete the access.

For short I/O accesses which will preserve any previous I/O or memory mapping, the functions **Pci_PhysReadIoByte** and **Pci_PhysWriteIoByte** will save the previous mapping, set up a mapping to the I/O address, perform the cycle, and restore the previous setting.

There are two ways that the DSP can determine the addresses of the other cards. First is to determine the addresses by performing configuration cycles. The second is to have

the HOST determine the addresses, and pass them to the DSP. There is no provision, at this time, to allow the device driver run configuration cycles on behalf of a user program.

There is a problem with having the PLX run its own config cycles in some cases. The PCI specification only allows the HOST to actually perform config cycles. So none of the available PCI bridges will pass a config cycle from a secondary bus to the primary bus. In other words, the PLX can only detect the PCI devices on its own bus, or any buses under that bus. So, for instance, the CPCI-AD320 can only see the PMC slot, since there is a bridge between the PLX and the CPCI bus. The CPCI-4IPM can see the CPCI bus, however, since there is no PCI bridge.

Config cycles are run by calling one of the following functions: **Pci_PhysReadConfigDword** or **Pci_PhysWriteConfigDword**.

The second main use of the ability to access PCI space, is to make use of the HOST memory as a large shared buffer. The HOST program will allocate a buffer to use, asigned to a 4k page boundary, and then make it accessible to the DSP. The driver will create a lookup table in HOST memory containing the starting physical address of each 4k page of HOST memory. The address of this table is then passed to the DSP (via the same interrupt and flag mechanism that the DMA code uses) and the memory functions will then be useable. The maximum amount of sharable memory is determined by a setting in the registry (setting the size of the lookup table) and is set in 4MB steps.

Once the HOST program has set up the shared buffer, the following functions can be used to setup a mapping to each 4k page of the HOST buffer. **Pci_MapSharedMemDword**, **Pci_MapSharedMemWord**, **Pci_MapSharedMemSint**, **Pci_MapSharedMemByte** and **Pci_MapSharedMemChar** all based on an offset into the shared buffer. Remember that each mapping is good for only a single 4kB page. Then the function must be called again to set up any access outside of that 4kB page.

There is one other set of functions which are useful for certain circumstances. **Pci_SaveCurrentMapping** and **Pci_RestoreCurrentMapping** allow the current state of the PCI mapping to be saved and restored, so that an interrupt routine, or an asychronous process, can borrow any current setup, and restore it before returning. The state is kept in the structure **PciMapping_t**.

DMA

The PLX chip has two DMA channels. The device driver and DSP library utilize one in each direction, for transferring data between the card and the HOST. The DSP usually programs these DMA channels directly through the PLX registers. This is taken care of by this library.

The HOST software is responsible for setting up the buffers for transfer.

The device driver is programmed to set up tables of physical addresses and sizes in DSP memory that describe each physical page of the buffer. The DSP is interrupted via a known doorbell, and the DSP then starts and throttles any transfer, a page at a time. When the DMA is complete, the DSP issues a known doorbell to the device driver. The driver then can set up the next transfer, and complete the previous one.

Making use of the DMA engine is very simple.

The DSP application first calls **Fifo_Init** on one or both of the FIFO classes which are part of this library. They are called **plx**:FifoToHost and **plx**:FifoFromHost. Then the application calls **host_InitializeDma** to enable interrupts and initialize the DMA variables.

To write values to the HOST, or to read values from the host, the application simply reads or writes the **FIFO** object using the access functions. These functions include:

Fifo_IsThereRoom Fifo_IsThereAny Fifo_Write Fifo_Read

Doorbells

The PLX has two 32 bit doorbell registers for communication between the HOST and the DSP. Certain bits are already used by the DSP library, the device driver, and this DLL.

Bit 31 of **PLX_REG**.l2pdbell is used to indicate that the hardware generated an interrupt.

Bits 30 - 9 are available for user applications.

Bit 8 is used for DMA communication between the driver and the DSP library.

Bits 7 - 4 indicate that the associated outgoing mailbox is empty. Bit 4 is for mailbox 0.

Bits 3 - 0 indicate that the associated incoming mailbox is full. Bit 0 is for mailbox 0.

Bits 31 - 9 of **PLX_REG**.p2ldbell are available for user applications.

Bit 8 is used for DMA communication between the driver and the DSP library.

Bits 7 - 4 tell the DSP that the associated incoming mailbox is empty. Bit 4 is for mailbox 0.

Bits 3 - 0 tell the DSP that the associated outgoing mailbox is full. Bit 0 is for mailbox 0.

Hardware Independence, PLX Access, Interrupts

When the alphi_io library is included in the build, several features are added to the DSP code running on the card. One main feature added is group of global variables which describe the environment that the DSP is operating in. Additional hardware functions allow for manipulation of the environment in a card independent manner. This facilitates an easy port of an application to a different ALPHI card or hardware platform.

The following functions will help in communication with the software running on the HOST.

host_ReadyIMbox host_ReadyOMbox host_ReadIMbox host_WriteOMbox

The following functions will help support DSP interrupts.

EnableIrq DisableIrq Forbid Permit c_int02 and the other default handlers.

To program DSP Timers, the #defines **TIMER_MICROSECONDS TIMER_MILLISECONDS TIMER_SECONDS** supply the values to be written to

the period registers **T0_PERIOD** and **T1_PERIOD**. **T0_CTRL** and **T1_CTRL** access the control register, and **TIMER_GO TIMER_HLD CP_MODE** and **CLK_INT** are helpful control bits. The timer count registers are **T0_CNTR** and **T1_CNTR**.

To delay a programmed amount, the #defines **MICRO_SEC MILLI_SEC SECONDS** supply the values to call **delay** with.

The functions **SetST GetST SetIE GetIE SetIF** and **GetIF** allow access to the DSP registers.

Linking and Initialization

The library can be set to either redirect console output to the serial port available on many products, or to keep the serial output directed at the DSP emulator, when it is being used for software development.

When a DSP application is linked with the alphi_io library before the C runtime library (RTS30.lib), a hardware identification function **hardware_FindSystemInfo** is called which initializes the global variables based on information found in the NVRAM used by the AMCC chip. The Serial port redirection is also set up and initialized. This is done by the function c_int00 in alphi_bt.asm. c_int00 in RTS30.lib is not linked in because the alphi_io.lib was used first to resolve undefined symbols.

If the redirection of the standard I/O to the serial port is not desired, link the RTS30.lib file first, then alphi_io.lib. In your main() function, make a call to **hardware_FindSystemInfo** to initialize the hardware library. c_int00 in RTS30.lib is used, and the initialization call is not made automatically before main().

Mailboxes

Mailboxes provide a primary means of transfering information between the host and the processor on the card.

First, it is necessary to discuss how the mailboxes function on an AMCC type device, because all of the support software is designed to interface with both AMCC and PLX based devices. On an AMCC based design, there are four 32 bit mailbox registers available for use.

It is possible to generate a host interrupt when a mailbox is accessed. An interrupt can be generated when a single mailbox is written to by the processor on the card or by the hardware as on the CPCI-SIP. Of less use, an additional interrupt can be generated when the processor on the card reads a mailbox. See Host Interrupts for more details.

On a PLX based design, there are eight mailboxes total, without any empty or full flags. This DLL and the alphi_io DSP library work together to allow 4 mailboxes in each direction, and they use certain doorbell bits to maintain empty/full flags. The doorbell registers will generate interrupts to the HOST device driver, which can be hooked by the user.

PLX Interrupt

This library hooks the PLX interupt in order to support Bus Master DMA directly. In order to support DMA, it was necessary to make use of certain mailbox registers on the

PLX to generate an interrupt to the DSP. It was intended that the user could also hook an interrupt routine, however that functionality has not been implemented as yet.

Serial Operation

When the alphi_io library is utilized to redirect the console output to the serial port, several features are added to the DSP code running on the card.

A SERIAL device handler is added to the runtime library allowing the user to open any device that the SERIAL driver knows about. Usually this is the two halves of the 8530 on the card, but other devices could be added by writing a device handler meeting the function prototype **Serial_Handler_t** and then calling **Serial_AddDevice** to add it to the list of devices.

An 8530 handler which knows how to handle 8530 class devices in a polled manner (without interrupt support) is made available to the SERIAL driver. This handler now supports backspaces and line kill (ESC).

And finally, a replacement HOST device for the TI supplied one (which normally communicates with the debugger through the emulator) redirects the console stdin, stdout, and stderr to the first serial device.

The 8530 driver initializes both SERIAL ports to 19200 baud, no parity, 8 bits, 1 stop bit. Textual mode is set which translates newlines into return newline pairs on output, and returns into newlines on input. Input characters are echoed. However, other modes can be set by changing the flags for the stream by calling **Serial_SetStreamFlags**.

Additional 8530 devices, such as those resident on the SCC-04 IP module, can be added to the SERIAL device by creating a **SC8530_t** structure and calling **Serial_AddDevice**.

Serial devices are opened by calling the standard IO library **fopen** or TI's **open** with a name "SERIAL:n" where n is the device number starting with 1. "SERIAL:1" is the serial port for the console device. "SERIAL:2" is the other half of the onboard 8530 chip. Any additional SERIAL devices, if present and made known to the SERIAL driver start with "SERIAL:3".

Remember that the standard IO routines buffer output by default. Sometimes it is necessary to call **fflush** to force output. Calling any of the input functions automatically flushes output. Output to stderr is not buffered in the standard I/O library, and can be used as a work around.

Shadowed Doorbells

The DSP is sometimes run with PLX generated interrupts turned on (such as when Bus Master DMA is being used). When this is the case, **l2pdbell** must be cleared to 0 in order to release the interrupt. The interrupt handler (**c_int01**) ORs the doorbell state into a hidden variable (except for the doorbell bits used by the library at interrupt time, such as for DMA change notification). The function **host_GetHostToDspDoorbellState** then uses the hidden variable for the current state.

When the PLX interrupt is disabled, then the **l2pdbell** register maintains the current state until **host_GetHostToDspDoorbellState** is called. Then the PLX register is copied to the hidden variable.

Therefore, customers desiring to directly query the doorbell state, should call **host_GetHostToDspDoorbellState** for this purpose.

Standard I/O Library

In the most recent version of the TI development tools (Version 5.0) there finally is support for the standard I/O library with the textual input and output directed to the debugger. The alphi_io library allows the user to replace the console supplied by the debugger with the serial port available on almost all of ALPHI Technology PCI cards. The library does this by replacing the lowest levels of the TI library with routines to direct output to the 8530 chip. No actual replacement of TI libraries are performed, instead, by placing the alphi_io.lib file before the rts30.lib file in the link command, the TI supplied functionality of debugger output is replaced by output to the serial port.

Look at the SerialEx DSP example to demonstrate simple debugging output to the serial port.

Software Modules

The source files required to build the DSP Library portion of this board support package. Also, any global variables are listed here. Full source is provided for these files in the DSP source library called ALPHI_IO.SRC. Use the TI supplied AR30 utility to extract the source files from this file as follows.

AR30 x alphi_io.src

Module 8530.c

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/8530.C

Description Implementation file for an 8530 class serial device.

Module 8530.h

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/8530.H

Description Include file for an 8530 class serial device.

Module alphi_io.c

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/ALPHI_IO.C

Description Implementation of the redirection of the standard input and output to the 8530 serial device.

Global Variables

boolean HOSTinitialized
A flag indicating whether initialization has been performed.

Module alphi_io.h

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/ALPHI_IO.H

Description Declaration of the default serial device for standard I/O.

Module def_int02.asm

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT02.ASM

Description Default interrupt handler.

Module def_int03.asm

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT03.ASM

Description Default interrupt handler.

Module def_int04.asm

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT04.ASM

Description Default interrupt handler.

Module def_int05.asm

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT05.ASM

Description Default interrupt handler.

Module def_int06.asm

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT06.ASM

Description Default interrupt handler.

Module def_int09.asm

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT09.ASM

Description Default interrupt handler.

Module def_int10.asm

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT10.ASM

Description Default interrupt handler.

Module def_int11.asm

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT11.ASM

Description Default interrupt handler.

Module def_int12.asm

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT12.ASM

Description Default interrupt handler.

Module def_int99.asm

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT99.ASM

Description Interrupt handler for unexpected interrupts.

Module fifo.c

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Description Implementation of a fifo of DWORDs used to move data from the HOST to the DSP and vice versa.

Module Fifo.h

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.H

Description Include file for DWORD FIFOs.

Module flash.c

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FLASH.C

Description Implementation of FLASH programming functions.

Module hardware.c

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.C

Description

Hardware dependent variables initialized from PLX NVRAM.

Global Variables

hardware_VendorId_t hardware_VendorId
Vendor ID of this board.

hardware_DeviceId_t hardware_DeviceId
Device ID of this board.

ulong hardware_SizeStaticRam
Number of DWORDS of DSP static memory.

void * hardware_AddrDualPortRam
Location of the dual ported RAM.

ulong hardware_SizeDualPortRam
Size of the dual ported RAM, or 0 if not present.

ulong hardware_ClockSpeedDsp
Clock speed of the DSP in Hertz.

ulong hardware_ClockSpeedSerial
Clock speed of the serial device in Hertz.

ubyte * hardware_AddrSerial
Address of the serial device.

hardware_SerialType_t hardware_SerialType
Type of the serial device. See **hardware_SerialType_t**.

PLX_REG * hardware_PlxRegisters
Address of the PLX add-on registers.

char * hardware_BoardName
Character stream identifying board name.

ulong hardware_NumberIpSlots
Number of IP slots accessible by DSP.

hardware_IpSlot_t hardware_IpSlot[]
Information describing each IP slot.

ubyte * hardware_pSystemFlash
Location of the system area of the FLASH.

ubyte * hardware_pUserFlash
Location of the user area of the FLASH.

ulong hardware_UserBoardId
User's board identification.

ulong hardware_BootOption
What to do at boot time.

Module hardware.h

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Description Declarations of the hardware identification code.

Module host.h

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HOST.H

Description Declarations of the command interface between the HOST and the DSP bootloader via the mailboxes.

Module Int_Ser.c

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/INT_SER.C

Description Implementation file for an Internal class serial device.

Module Int_Ser.h

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/INT_SER.H

Description Include file for an Internal class serial device.

Module nvram.c

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/NVRAM.C

Description Routines to access the NVRAM of the PLX.

Module pci.c

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Description Definitions of the PCI access routines.

Module pci.h

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.H

Description Declarations of the PCI access routines.

Module plx.c

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PLX.C

Description Routines to access the local bus side of the PLX.

Global Variables

Fifo FifoToHost
Fifo object for the data stream to the HOST from the DSP.

Fifo FifoFromHost
Fifo object for the data stream from the HOST to the DSP.

Module regs.asm

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/REGS.ASM

Description C callable functions to alter TMS machine registers.

Module serial.c

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/SERIAL.C

Description Implementation of a serial device.

Module serial.h

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/SERIAL.H

Description Declarations of the generic SERIAL device.

Module vecs.asm

Filename: D:/ALPHIPCI/ALPHI_IO/C32_PLX/VECS.ASM

Description Interrupt vector table.

C Functions and Callbacks

C functions implementing the functionality of this library.

c_int01

void c_int01()

Handle interrupt for External Interrupt 0 from PLX.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PLX.C

Comments This function is called to handle interrupts from the PLX.

c_int02

void c_int02()

Interrupt handler for External Interrupt 1.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT02.ASM

Comments This default handler does nothing.

The customer replaces this function by defining his own function with this name. The linker will not extract this function if it is previously defined.

See Also **EnableIrq DisableIrq Permit Forbid CPU_EINT1**

c_int03

void c_int03()

Interrupt handler for External Interrupt 2.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT03.ASM

Comments This default handler does nothing.

The customer replaces this function by defining his own function with this name. The linker will not extract this function if it is previously defined.

See Also **EnableIrq DisableIrq Permit Forbid CPU_EINT2**

c_int04

void c_int04()

Interrupt handler for External Interrupt 3.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT04.ASM

Comments This default handler does nothing.

The customer replaces this function by defining his own function with this name. The linker will not extract this function if it is previously defined.

See Also **EnableIrq DisableIrq Permit Forbid CPU_EINT3**

c_int05

void c_int05()

Interrupt handler for Serial port 0 Transmitter.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT05.ASM

Comments This default handler does nothing.

The customer replaces this function by defining his own function with this name. The linker will not extract this function if it is previously defined.

See Also **EnableIrq DisableIrq Permit Forbid CPU_EXINT0**

c_int06

void c_int06()

Interrupt handler for Serial port 0 Receiver.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT06.ASM

Comments This default handler does nothing.

The customer replaces this function by defining his own function with this name. The linker will not extract this function if it is previously defined.

See Also **EnableIrq DisableIrq Permit Forbid CPU_ERINT0**

c_int09

void c_int09()

Interrupt handler for DSP Timer 0.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT09.ASM

Comments This default handler does nothing.

The customer replaces this function by defining his own function with this name. The linker will not extract this function if it is previously defined.

See Also **EnableIrq DisableIrq Permit Forbid CPU_ETINT0**

c_int10

void c_int10()

Interrupt handler for DSP Timer 1.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT10.ASM

Comments This default handler does nothing.

The customer replaces this function by defining his own function with this name. The linker will not extract this function if it is previously defined.

See Also **EnableIrq DisableIrq Permit Forbid CPU_ETINT1**

c_int11

void c_int11()

Interrupt handler for DSP DMA engine number 0.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT11.ASM

Comments This default handler does nothing.

The customer replaces this function by defining his own function with this name. The linker will not extract this function if it is previously defined.

See Also **EnableIrq DisableIrq Permit Forbid CPU_EDINT0**

c_int12

void c_int12()

Interrupt handler for DSP DMA engine number 1.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT12.ASM

Comments This default handler does nothing.

The customer replaces this function by defining his own function with this name. The linker will not extract this function if it is previously defined.

See Also **EnableIrq DisableIrq Permit Forbid CPU_EDINT1**

c_int99

void c_int99()

Interrupt handler for unexpected interrupts.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/DEF_INT99.ASM

Comments This handler does nothing.

delay

void delay(
ulong *NumCycles***)**

Delay for the specified number of H1/H3 cycles.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/REGS.ASM

Parameters *NumCycles*
Number of cycles to delay.

See Also **MICRO_SEC MILLI_SEC SECONDS**

DisableIrq

void DisableIrq(
register uint *mask***)**

Select interrupt causes to be disabled.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.C

Parameters *mask*
Interrupt causes to be disabled.

Comments More than one interrupt cause can be selected by ORing them together.

See Also **Defines**:CPU_E* and DMA_E* bits.

EnableIrq

void EnableIrq(
register uint *mask***)**

Select interrupt causes to be enabled.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.C

Parameters *mask*
Interrupt causes to be enabled.

Comments More than one interrupt cause can be selected by ORing them together.

See Also **Defines**:CPU_E* and DMA_E* bits.

EventHandler_t

void EventHandler_t(
typedef *void (*EventHandler_t)***)**

Form of the Event Handler function.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.H

Parameters *void (*EventHandler_t)*
User Data.

Comments This function is provided by the user to be called when certain **Fifo** events occur.

See Also **Fifo_SetSpaceAvailableEvent Fifo_SetDataAvailableEvent**

Fifo_CompleteReadBlock

void Fifo_CompleteReadBlock(
Fifo * *pFifo***)**

Complete a block read by **Fifo_ReadBlock**.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Parameters *pFifo*
Pointer to the **Fifo**.

Comments Marks the space as now empty. If the previous call to **Fifo_ReadBlock** did not map the total desired number of samples because the buffer was not contiguous, then **Fifo_ReadBlock** should be called after this function for the rest of the transfer.

The Space Available event function is called, if the correct conditions are met. If this function is called on FifoFromHost, the DMA event handler which has been installed by the library, must be called with global interrupts disabled. Therefore, if this function is called outside of an interrupt handler, surround this call with **Forbid** and **Permit**.

Fifo_CompleteWriteBlock

void Fifo_CompleteWriteBlock(
Fifo * *pFifo***)**

Complete a block write by **Fifo_WriteBlock**.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Parameters

pFifo
Pointer to the **Fifo**.

Comments Marks the space as now full. If the previous call to **Fifo_WriteBlock** did not map the total desired number of samples because the buffer was not contiguous, then **Fifo_WriteBlock** should be called after this function for the rest of the transfer.

The Data Available event function is called, if the correct conditions are met. If this function is called on FifoToHost, the DMA event handler which has been installed by the library, must be called with global interrupts disabled. Therefore, if this function is called outside of an interrupt handler, surround this call with **Forbid** and **Permit**.

Fifo_Init

boolean Fifo_Init(
Fifo * *pFifo***,**
ulong *Size***)**

Initialize the **Fifo** structure.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Return Value Returns FALSE if memory was successfully allocated, TRUE if there was a problem.

Parameters

pFifo
Pointer to the **Fifo** structure.

Size
Desired size of the **Fifo** in DWORDs. The size must be a power of 2.

Comments Initializes the **Fifo** structure to support streams to and from the HOST. Allocates the specified amount of space from the heap for the FIFO.

Fifo_IsThereAny

boolean Fifo_IsThereAny(
Fifo * *pFifo***)**

Determine if there is any, and if so, how much, data is available in the **Fifo**.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Return Value Returns FALSE if there is no data available, otherwise returns the amount of data available in DWORDs.

Parameters

pFifo
Pointer to the **Fifo** structure.

Fifo_IsThereRoom

boolean Fifo_IsThereRoom(
Fifo * *pFifo***)**

Determine if there is any, and if so, how much, empty space in the **Fifo**.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Return Value Returns FALSE if there is no room available, otherwise returns the amount of space available in DWORDs.

Parameters

pFifo
Pointer to the **Fifo** structure.

Fifo_Read

void Fifo_Read(
int *NumToRead***,**
Fifo * *pFifo***,**
volatile ulong * *pData***)**

Copy the specified number of DWORDs from the **Fifo** to a hardware register.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Parameters

NumToRead
Number of DWORDs to be copied.

pFifo
Pointer to the **Fifo** structure.

pData
Pointer to the hardware register to receive the data to be copied.

Comments

This function is used on designs which have a 32 bit hardware register which contains the target of the data to be copied from the **Fifo**. The pointer to the register is NOT incremented, as would be appropriate for a block transfer from memory.

The Space Available event function is called, if the correct conditions are met. If this function is called on FifoFromHost, the DMA event handler which has been installed by the library, must be called with global interrupts disabled. Therefore, if this function is called outside of an interrupt handler, surround this call with **Forbid** and **Permit**.

Developer Notes

It is assumed that the user has already called **Fifo_IsThereAny** to ensure that sufficient data is available.

Fifo_ReadBlock

int Fifo_ReadBlock(
int *DesiredNumToRead***,**
Fifo * *pFifo***,**
ulong ** *ppData***)**

Return a pointer to a block of data in the FIFO that is contiguous.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Return Value

Actual number of DWORDs that can be read at this address.

Parameters

DesiredNumToRead
Desired size of the block to be read.

pFifo
Pointer to the **Fifo**.

ppData
Returned pointer to the contiguous block from the **Fifo**.

Comments

Presumably, the caller has already determined that there is at least *DesiredNumToRead* DWORDS available for reading (using **Fifo_IsThereAny**).

Note that this function does not actually copy any data. Presumable some external method is actually used, such as the hardware DMA in the PLX.

The function will return with fewer than *DesiredNumToRead* if the read would wrap back to the beginning of the buffer.

This function needs to be followed by a call to **Fifo_CompleteReadBlock** to increment the pointers and call the event function.

Fifo_Reset

void Fifo_Reset(
Fifo * *pFifo***)**

Reset the **Fifo** to the empty state.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Parameters

pFifo
Pointer to the **Fifo** structure.

Developer Notes

This function probably should call the empty event.

Fifo_SetDataAvailableAmount

void Fifo_SetDataAvailableAmount(
Fifo * *pFifo***,**
ulong *AmountData***)**

Set the number of DWORDs which must be stored in order that the event function is called.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Parameters

pFifo
Pointer to the **Fifo**.

AmountData
Number of DWORDs which must be available before the event function is called.

Comments

This function and **Fifo_SetDataAvailableEvent** sets up a function to be called when the specified amount of data is available in the **Fifo**. The function will be called in the context of the **Fifo** function which makes the data available. So, if **Fifo_Write** was called from an interrupt routine with global interrupts off, then the event function will be called with interrupts off. Note that this is the case for the library provided FifoFromHost.

Call **Fifo_SetDataAvailableEvent** to set the called function. If the specified amount is already present when this function is called, the event function will be called at this time.

Fifo_SetDataAvailableEvent

void Fifo_SetDataAvailableEvent(
Fifo * *pFifo***,**
EventHandler_t *pDataAvailable***,**
void * *pDataAvailableParm***)**

Set up an event function to be called when the a specified amount of data is available in the **Fifo**.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Parameters

pFifo
Pointer to the **Fifo**.

pDataAvailable
User function to be called when the specified amount of data is available. Use NULL to disable future notifications. See **EventHandler_t**.

pDataAvailableParm
Parameter to be passed to *pDataAvailable* when it is to be called.

Comments

This function sets up a function to be called when the specified amount of data is available in the **Fifo**. The function will be called in the context of the **Fifo** function which makes the data available. So, if **Fifo_Write** was called from an interrupt routine with global interrupts off, then the event function will be called with interrupts off. Note that this is the case for the library provided FifoFromHost.

Call this function with NULL as the *pDataAvailable* function to disable future notifications. Only one function can be called.

Call **Fifo_SetDataAvailableAmount** to set the specified amount. If the specified amount is already present when this function is called, the event function will be called at this time.

Fifo_SetSpaceAvailableAmount

void Fifo_SetSpaceAvailableAmount(
Fifo * *pFifo***,**
ulong *AmountSpace***)**

Set the number of DWORDs which must be free in order that the event function is called.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Parameters

pFifo
Pointer to the **Fifo**.

AmountSpace
Number of DWORDs which must be free before the event function is called.

Comments

This function and **Fifo_SetSpaceAvailableEvent** sets up a function to be called when the specified amount of free space is available in the **Fifo**. The function will be called in the context of the **Fifo** function which makes the data available. So, if **Fifo_Read** was called from an interrupt routine with global interrupts off, then the event function will be called with interrupts off. Note that this is the case for the library provided FifoToHost.

Call **Fifo_SetSpaceAvailableEvent** to set the called function. If the specified amount is already present when this function is called, the event function will be called at this time.

Fifo_SetSpaceAvailableEvent

void Fifo_SetSpaceAvailableEvent(
Fifo * *pFifo***,**
EventHandler_t *pSpaceAvailable***,**
void * *pSpaceAvailableParm***)**

Set up an event function to be called when the a specified amount of free space is available in the **Fifo**.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Parameters

pFifo
Pointer to the **Fifo**.

pSpaceAvailable
User function to be called when the specified amount of free space is available. Use NULL to disable future notifications. See **EventHandler_t**.

pSpaceAvailableParm
Parameter to be passed to *pSpaceAvailable* when it is to be called.

Comments

This function sets up a function to be called when the specified amount of free space is available in the **Fifo**. The function will be called in the context of the **Fifo** function which makes the space available. So, if **Fifo_Read** was called from an interrupt routine with global interrupts off, then the event function will be called with interrupts off. Note that this is the case for the library provided FifoToHost.

Call this function with NULL as the *pSpaceAvailable* function to disable future notifications. Only one function can be called.

Call **Fifo_SetSpaceAvailableAmount** to set the specified amount. If the specified amount is already present when this function is called, the event function will be called at this time.

Fifo_Write

void Fifo_Write(
int *NumToWrite***,**
Fifo * *pFifo***,**
volatile ulong * *pData***)**

Copy the specified number of DWORDs from a hardware register to the **Fifo**.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Parameters

NumToWrite
Number of DWORDs to be copied.

pFifo
Pointer to the **Fifo** structure.

pData
Pointer to the hardware register containing the data to be copied.

Comments

This function is used on designs which have a 32 bit hardware register which contains the data to be copied to the **Fifo**. The pointer to the register is NOT incremented, as would be appropriate for a block transfer from memory.

The Data Available event function is called, if the correct conditions are met. If this function is called on FifoToHost, the DMA event handler which has been installed by the library, must be called with global interrupts disabled. Therefore, if this function is called outside of an interrupt handler, surround this call with **Forbid** and **Permit**.

Developer Notes It is assumed that the user has already called **Fifo_IsThereRoom** to ensure that sufficient room is available.

Fifo_WriteBlock

int Fifo_WriteBlock(
int *DesiredNumToWrite***,**
Fifo * *pFifo***,**
ulong ** *ppData***)**

Return a pointer to a block of empty space in the FIFO that is contiguous.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Return Value Actual number of DWORDs that can be written to this address.

Parameters

DesiredNumToWrite
Desired size of the block to be written.

pFifo
Pointer to the **Fifo**.

ppData
Returned pointer to the contiguous block from the **Fifo**.

Comments Presumably, the caller has already determined that there is at least *DesiredNumToWrite* DWORDS available for writing (using **Fifo_IsThereRoom**).

Note that this function does not actually copy any data. Presumable some external method is actually used, such as the hardware DMA in the PLX.

The function will return with fewer than *DesiredNumToWrite* if the write would wrap back to the beginning of the buffer.

This function needs to be followed by a call to **Fifo_CompleteWriteBlock** to increment the pointers and call the event function.

Fifo_WriteTwoWords

void Fifo_WriteTwoWords(
int *NumToWrite***,**
Fifo * *pFifo***,**
volatile ulong * *pData***)**

Copy the specified number of pairs of WORDs from a hardware register to the **Fifo**.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.C

Parameters

NumToWrite
Number of DWORDs to be copied.

pFifo
Pointer to the **Fifo** structure.

pData
Pointer to the hardware register containing the data to be copied.

Comments

This function is used on designs which have a 16 bit hardware register which contains the data to be copied to the **Fifo**. The pointer to the register is NOT incremented, as would be appropriate for a block transfer from memory. Values are copied 16 bits at a time (from the lower WORD) and merged into a single 32 bit DWORD before put into the FIFO. The first WORD read becomes the least significant WORD, and the second becomes the most significant. Therefore, a little-endian HOST will see the WORDs in the correct order.

The Data Available event function is called, if the correct conditions are met. If this function is called on FifoToHost, the DMA event handler which has been installed by the library, must be called with global interrupts disabled. Therefore, if this function is called outside of an interrupt handler, surround this call with **Forbid** and **Permit**.

Developer Notes

It is assumed that the user has already called **Fifo_IsThereRoom** to ensure that sufficient room is available.

Forbid

void Forbid()

Disable global interrupts.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.C

Comments

Use this function to disable global interrupts. TI errata state that there is a race condition where certain operations which affect the GIE flag concurrent with a hardware interrupt will cause the GIE to be forced to a 0, disabling further interrupts. Using this function to control the global interrupts alleviates the problem.

See Also

Permit

GetIE

ulong GetIE()

Get the present value of the DSP IE register.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/REGS.ASM

Return Value

The register contents.

GetIF

ulong GetIF()

Get the present value of the DSP IF register.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/REGS.ASM

Return Value The register contents.

GetST

ulong GetST()

Get the present value of the DSP ST register.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/REGS.ASM

Return Value The register contents.

Handler_t

void Handler_t()

Interrupt Vector Table Entry.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Comments This is the form that the user should form the interrupt handler to.

Developer Notes Although this library will automatically hook into user provided interrupt handlers, if they are named correctly, occasionally it is necessary to directly change the interrupt vector table directly. This is the form of the entries in that table.

See Also **InterruptVectors_t**

---

hardware_FindSystemInfo

void hardware_FindSystemInfo()

Initialize all the hardware global variables based on values found in the PLX's NVRAM.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.C

Comments This function is normally called by the overall library initialization code in **HOSTinit**, which is in turn called by the alphi_io C startup routine in alphi_bt.asm. This normally occurs before even the user's main() function is called.

However, sometimes it is desired to utilize alphi_io without redirecting the stdio output to the serial port, and thereby allowing for a smaller code image. To do this, the user should place the C runtime library BEFORE the alphi_io library, and call this function first in main() before doing anything else.

hardware_SelectAddressSpace

int hardware_SelectAddressSpace()

Switch between IP memory spaces.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.C

Return Value

Returns 0 or greater if successful. Returns less than zero if failed due to unrecognized hardware or parameter out of range.

Comments

On intelligent IP carriers, each IP memory space resides at the same starting address (0x400000). This function virtualizes switching between IP memory spaces and the FLASH device in a device independent manner. This allows code to perform this operation regardless of the platform differences.

This function is presently applicable to the CPCI-IPM.

The *AddressSpace* 0 refers to the FLASH memory.

The *AddressSpace* 1 - 4 refers to IP memory spaces for A - D in 16 bit mode. A and B only are supported on the CPCI-IPM.

The *AddressSpace* 5 - 6 refers to IP memory spaces for AB and CD in 32 bit mode. AB only is supported on the CPCI-IPM.

The *AddressSpace* 7 refers to presenting the first 2 MB of each IP memory space. IP A is at 0x400000, IP B is at 0x500000, IP C is at 0x600000, and IP D is at 0x700000. Access is in 16 bit mode. This is only supported on the CPCI-IPM.

The *AddressSpace* 8 refers to presenting the first 2 MB of each IP memory space. IP A/B is at 0x400000, and IP C/D is at 0x600000. Access is in 32 bit mode. This is only supported on the CPCI-IPM.

See Also

hardware_IpSlot_t hardware:hardware_NumberIpSlots

host_GetHostToDspDoorbellState

int host_GetHostToDspDoorbellState(
int *DesiredBitMask***,**
boolean *bClearState***)**

Query state of the HOST to DSP doorbell register.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PLX.C

Return Value

Current state of the register ANDed with *DesiredBitMask*.

Parameters

DesiredBitMask
Bit(s) of interest.

bClearState
If TRUE, then clear the current state of the doorbell register (based on *DesiredBitMask*).

Comments This function provides a means of querying the doorbell register regardless of whether the PLX interrupt is enabled or disabled. *DesiredBitMask* is set to the bit(s) of interest. The return value contains the current state of the bit(s). If *bClearState* is TRUE, then the bits are cleared from the stored state. If it is FALSE, then the bits are not changed.

See Also Shadowed Doorbells Doorbells

---

host_InitializeDma

void host_InitializeDma()

Initialize the library to support DMA to and from the HOST.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PLX.C

Comments This function should be called by the customer DSP code to initialize the library for DMA.

See Also DMA

---

host_ReadIMbox

ulong host_ReadIMbox(
ushort *WhichMailbox***,**
boolean *wait***)**

Read a value from the specified inbound mailbox.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PLX.C

Return Value The read in value, or 0 if a parameter was out of range.

Parameters

WhichMailbox
Which mailbox is of interest.

wait
Should we wait for the HOST to write something fresh or return with the present value?

Developer Notes There is no provision for HOST timeouts.

See Also Mailboxes

---

host_ReadyIMbox

boolean host_ReadyIMbox(
ushort *WhichMailbox***)**

Is there fresh data in the specified inbound mailbox?

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PLX.C

Parameters

WhichMailbox
Which mailbox is of interest.

Return Codes

TRUE	Yes, there is.
FALSE	No, there isn't, or the parameter was out of range.

See Also Mailboxes

---

host_ReadyOMbox

boolean host_ReadyOMbox(
ushort *WhichMailbox***)**

Is it safe to write fresh data to the specified outbound mailbox because the HOST has read the previous value?

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PLX.C

Parameters

WhichMailbox
Which mailbox is of interest.

Return Codes

TRUE	Yes, it is.
FALSE	No, it isn't, or the parameter was out of range.

See Also Mailboxes

---

host_WriteOMbox

void host_WriteOMbox(
ushort *WhichMailbox***,**
ulong *val***,**
boolean *wait***)**

Write a value to the specified outbound mailbox.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PLX.C

Parameters

WhichMailbox
Which mailbox is of interest.

val
The value to be written.

wait
Should we wait for the HOST to read the present value or just overwrite?

Developer Notes There is no provision for HOST timeouts.

See Also Mailboxes

nvram_ReadWordBlock

void nvram_ReadWordBlock(
ushort *Address*,
ushort * *pBuffer*,
ulong *NumWords*)

Read a block of WORDs from the NVRAM.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/NVRAM.C

Parameters

Address
NVRAM Address to read.

pBuffer
Buffer to copy data to.

NumWords
Number of WORDs to copy.

nvram_WriteWord

void nvram_WriteWord(
ushort *Address*,
ushort *Data*)

Write a WORD to the NVRAM.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/NVRAM.C

Parameters

Address
NVRAM address to write to.

Data
WORD to be written.

nvram_WriteWordBlock

void nvram_WriteWordBlock(
ushort *Address*,
ushort * *pBuffer*,
ulong *NumWords*)

Write a block of WORDs to the NVRAM.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/NVRAM.C

Parameters

Address
NVRAM address to write to.

pBuffer
Buffer to copy data from.

NumWords
Number of WORDs to copy.

Pci_MapPhysIoByte

byte * Pci_MapPhysIoByte(
ulong *PhysAddr***)**

Map PCI physical I/O address space (as 8 bit unsigned BYTEs) into DSP memory space.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value Valid DSP pointer to access the specified PCI physical address.

Parameters *PhysAddr*
PCI address to access.

Comments Since only 64 kB I/O addresses are valid, all I/O addresses are available with the same mapping. All DSP cycles will result in 8 bit PCI cycles. Reads will have upper 24 bits zeroed out.

Each 32 bit DWORD to the DSP is an 8 bit PCI access.

See Also Direct PCI Access by DSP (PLX)

Pci_MapPhysIoDword

ulong * Pci_MapPhysIoDword(
ulong *PhysAddr***)**

Map PCI physical I/O address space (as 32 bit DWORDs) into DSP memory space.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value Valid DSP pointer to access the specified PCI physical address.

Parameters *PhysAddr*
PCI address to access.

Comments Since only 64 kB I/O addresses are valid, all I/O addresses are available with the same mapping. All DSP cycles will result in 32 bit PCI cycles.

See Also Direct PCI Access by DSP (PLX)

Pci_MapPhysIoWord

ushort * Pci_MapPhysIoWord(
ulong *PhysAddr***)**

Map PCI physical I/O address space (as 16 bit unsigned WORDs) into DSP memory space.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value Valid DSP pointer to access the specified PCI physical address.

Parameters *PhysAddr*
PCI address to access.

Comments Since only 64 kB I/O addresses are valid, all I/O addresses are available with the same mapping. All DSP cycles will result in 16 bit PCI cycles. Reads will have upper 16 bits zeroed out.

Each 32 bit DWORD to the DSP is a 16 bit PCI access.

See Also Direct PCI Access by DSP (PLX)

Pci_MapPhysMemByte

byte * Pci_MapPhysMemByte(
ulong *PhysAddr***)**

Map PCI physical memory address space (as 8 bit unsigned BYTEs) into DSP memory space.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value Valid DSP pointer to access the specified PCI physical address.

Parameters *PhysAddr*
PCI address to access.

Comments Up to 4 MB of PCI memory space can be mapped at one time. All DSP cycles will result in 8 bit PCI cycles. Reads will have upper 24 bits zeroed out. Writes may be optimized into 32 or 16 bit writes.

Each 32 bit DWORD to the DSP is an 8 bit PCI access.

See Also Direct PCI Access by DSP (PLX)

Pci_MapPhysMemChar

char * Pci_MapPhysMemChar(
ulong *PhysAddr***)**

Map PCI physical memory address space (as 8 bit signed chars) into DSP memory space.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value Valid DSP pointer to access the specified PCI physical address.

Parameters *PhysAddr*
PCI address to access.

Comments Up to 4 MB of PCI memory space can be mapped at one time. All DSP cycles will result in 8 bit PCI cycles. Reads will have upper 24 bits sign extended. Writes may be optimized into 32 or 16 bit writes.

Each 32 bit DWORD to the DSP is an 8 bit PCI access.

See Also Direct PCI Access by DSP (PLX)

Pci_MapPhysMemDword

ulong * Pci_MapPhysMemDword(
ulong *PhysAddr***)**

Map PCI physical memory address space (as 32 bit DWORDs) into DSP memory space.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value Valid DSP pointer to access the specified PCI physical address.

Parameters *PhysAddr*
PCI address to access.

Comments Up to 16 MB of PCI memory space can be mapped at one time. All DSP cycles will result in 32 bit PCI cycles.

See Also Direct PCI Access by DSP (PLX)

Pci_MapPhysMemSint

short * Pci_MapPhysMemSint(
ulong *PhysAddr***)**

Map PCI physical memory address space (as 16 bit signed shorts) into DSP memory space.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value Valid DSP pointer to access the specified PCI physical address.

Parameters *PhysAddr*
PCI address to access.

Comments Up to 8 MB of PCI memory space can be mapped at one time. All DSP cycles will result in 16 bit PCI cycles. Reads will have upper 16 bits sign extended. Writes may be optimized into 32 bit writes.

Each 32 bit DWORD to the DSP is a 16 bit PCI access.

See Also Direct PCI Access by DSP (PLX)

Pci_MapPhysMemWord

ushort * Pci_MapPhysMemWord(
ulong *PhysAddr***)**

Map PCI physical memory address space (as 16 bit unsigned WORDs) into DSP memory space.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value Valid DSP pointer to access the specified PCI physical address.

Parameters

PhysAddr
PCI address to access.

Comments Up to 8 MB of PCI memory space can be mapped at one time. All DSP cycles will result in 16 bit PCI cycles. Reads will have upper 16 bits zeroed out. Writes may be optimized into 32 bit writes.

Each 32 bit DWORD to the DSP is a 16 bit PCI access.

See Also Direct PCI Access by DSP (PLX)

Pci_MapSharedMemByte

byte * Pci_MapSharedMemByte(
ulong *LogicalAddr***)**

Set the PLX and DSP registers to access the HOST shared memory as unsigned BYTEs.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value DSP pointer to access the specified *LogicalAddress*.

Parameters

LogicalAddr
Byte offset into shared buffer.

Comments This function makes the necessary changes to map the specified page of shared HOST memory into the DSP's address space, and returns a pointer that the DSP can use to access the HOST memory. Each 32 bit DSP address corresponds to an 8 bit PCI access. Therefore, each increment of 1 in DSP address corresponds to an increment of 1 byte in PCI address. Upper 24 bits will be zeroed out on reads. Upper bits on writes are ignored.

There is an optimization, so that addresses which reside in the same physical page, will return the corrected DSP address, but not modify any registers. Therefore, if desired, random access to all of the HOST buffer could require that every access go through this function.

Developer Notes This function must be called prior to the first access for each page in the HOST memory. Once a new page is selected, the earlier page is no longer accessible. Each page is 4096 BYTEs long. Failure, deliberate or unplanned, to call this function prior to

each page accessed may result in unpredicted behavior of the HOST system, since indiscriminately writing physical memory bypasses all operating system protection.

See Also Direct PCI Access by DSP (PLX)

Pci_MapSharedMemChar

char * Pci_MapSharedMemChar(
ulong *LogicalAddr***)**

Set the PLX and DSP registers to access the HOST shared memory as signed chars.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value DSP pointer to access the specified *LogicalAddress*.

Parameters

LogicalAddr
Byte offset into shared buffer.

Comments This function makes the necessary changes to map the specified page of shared HOST memory into the DSP's address space, and returns a pointer that the DSP can use to access the HOST memory. Each 32 bit DSP address corresponds to an 8 bit PCI access. Therefore, each increment of 1 in DSP address corresponds to an increment of 1 byte in PCI address. Upper 24 bits will be sign extended bit 7 on reads. Upper bits on writes are ignored.

There is an optimization, so that addresses which reside in the same physical page, will return the corrected DSP address, but not modify any registers. Therefore, if desired, random access to all of the HOST buffer could require that every access go through this function.

Developer Notes This function must be called prior to the first access for each page in the HOST memory. Once a new page is selected, the earlier page is no longer accessible. Each page is 4096 BYTEs long. Failure, deliberate or unplanned, to call this function prior to each page accessed may result in unpredicted behavior of the HOST system, since indiscriminately writing physical memory bypasses all operating system protection.

See Also Direct PCI Access by DSP (PLX)

Pci_MapSharedMemDword

ulong * Pci_MapSharedMemDword(
ulong *LogicalAddr***)**

Set the PLX and DSP registers to access the HOST shared memory as unsigned DWORDs.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value DSP pointer to access the specified *LogicalAddress*.

Parameters

LogicalAddr
Byte offset into shared buffer.

Comments

This function makes the necessary changes to map the specified page of shared HOST memory into the DSP's address space, and returns a pointer that the DSP can use to access the HOST memory. Each 32 bit DSP address corresponds to a 32 bit PCI access. Therefore, each increment of 1 in DSP address corresponds to an increment of 4 bytes in PCI address.

There is an optimization, so that addresses which reside in the same physical page, will return the corrected DSP address, but not modify any registers. Therefore, if desired, random access to all of the HOST buffer could require that every access go through this function.

Developer Notes

This function must be called prior to the first access for each page in the HOST memory. Once a new page is selected, the earlier page is no longer accessible. Each page is 1024 DWORDs long. Failure, deliberate or unplanned, to call this function prior to each page accessed may result in unpredicted behavior of the HOST system, since indiscriminately writing physical memory bypasses all operating system protection.

See Also

Direct PCI Access by DSP (PLX)

Pci_MapSharedMemSint

short * Pci_MapSharedMemSint(
ulong *LogicalAddr***)**

Set the PLX and DSP registers to access the HOST shared memory as signed 16 bit shorts.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value

DSP pointer to access the specified *LogicalAddress*.

Parameters

LogicalAddr
Byte offset into shared buffer.

Comments

This function makes the necessary changes to map the specified page of shared HOST memory into the DSP's address space, and returns a pointer that the DSP can use to access the HOST memory. Each 32 bit DSP address corresponds to a 16 bit PCI access. Therefore, each increment of 1 in DSP address corresponds to an increment of 2 bytes in PCI address. Upper 16 bits will be sign extended from bit 15 on reads. Upper bits on writes are ignored.

There is an optimization, so that addresses which reside in the same physical page, will return the corrected DSP address, but not modify any registers. Therefore, if desired, random access to all of the HOST buffer could require that every access go through this function.

Developer Notes

This function must be called prior to the first access for each page in the HOST memory. Once a new page is selected, the earlier page is no longer accessible. Each page is 2048 shorts long. Failure, deliberate or unplanned, to call this function prior to each page accessed may result in unpredicted behavior of the HOST system, since indiscriminately writing physical memory bypasses all operating system protection.

See Also Direct PCI Access by DSP (PLX)

Pci_MapSharedMemWord

unsigned short * Pci_MapSharedMemWord(
ulong *LogicalAddr***)**

Set the PLX and DSP registers to access the HOST shared memory as unsigned WORDs.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value DSP pointer to access the specified *LogicalAddress*.

Parameters

LogicalAddr
Byte offset into shared buffer.

Comments This function makes the necessary changes to map the specified page of shared HOST memory into the DSP's address space, and returns a pointer that the DSP can use to access the HOST memory. Each 32 bit DSP address corresponds to a 16 bit PCI access. Therefore, each increment of 1 in DSP address corresponds to an increment of 2 bytes in PCI address. Upper 16 bits will be zeroed out on reads. Upper bits on writes are ignored.

There is an optimization, so that addresses which reside in the same physical page, will return the corrected DSP address, but not modify any registers. Therefore, if desired, random access to all of the HOST buffer could require that every access go through this function.

Developer Notes This function must be called prior to the first access for each page in the HOST memory. Once a new page is selected, the earlier page is no longer accessible. Each page is 2048 WORDs long. Failure, deliberate or unplanned, to call this function prior to each page accessed may result in unpredicted behavior of the HOST system, since indiscriminately writing physical memory bypasses all operating system protection.

See Also Direct PCI Access by DSP (PLX)

Pci_PhysReadConfigDword

ulong Pci_PhysReadConfigDword(
ConfigAddr_t *ConfigAddr***)**

Read a DWORD from configuration space.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value Value read from the port.

Parameters

ConfigAddr
Configuration register to read.

Comments Save the current state of the bridge, set up the specified configuration address, perform the read, restore the state, and return the value.

Useful to find the configuration of the other PCI card. See the PCI specifications for details of the configuration header.

See Also Direct PCI Access by DSP (PLX)

Pci_PhysReadIoByte

byte Pci_PhysReadIoByte(
ulong *PhysAddr***)**

Read a byte from the specified I/O port.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value Value read from the port.

Parameters *PhysAddr*
PCI address to access.

Comments Save the current state of the bridge, set up the specified I/O port, perform the read, restore the state, and return the value read.

Useful when another PCI device needs to be controlled both by I/O and memory accesses. This function will not disturb the previous memory mapping.

See Also Direct PCI Access by DSP (PLX)

Pci_PhysWriteConfigDword

void Pci_PhysWriteConfigDword(
ConfigAddr_t *ConfigAddr***,**
ulong *Data***)**

Write a DWORD to configuration space.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Return Value Value read from the port.

Parameters *ConfigAddr*
Configuration register to write.

Data
Value to write.

Comments Save the current state of the bridge, set up the specified configuration address, perform the write, restore the state, and return.

Useful to set the configuration of the other PCI card. See the PCI specifications for details of the configuration header.

Developer Notes Setting the PCI configuration of other cards is best left up to the HOST processor. There is no means to notify a windows operating system of a third party changing the PCI configuration. Use extreme care.

See Also Direct PCI Access by DSP (PLX)

Pci_PhysWriteIoByte

void Pci_PhysWriteIoByte(
ulong *PhysAddr***,**
byte *Data***)**

Write a byte to the specified I/O port.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Parameters

PhysAddr
PCI address to access.

Data
Value to write.

Comments Save the current state of the bridge, set up the specified I/O port, perform the write, restore the state, and return.

Useful when another PCI device needs to be controlled both by I/O and memory accesses. This function will not disturb the previous memory mapping.

See Also Direct PCI Access by DSP (PLX)

Pci_RestoreCurrentMapping

void Pci_RestoreCurrentMapping(
PciMapping_t * *pPciMapping***)**

Restore the PLX and DSP registers to the state defined in the structure.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Parameters

pPciMapping
Pointer to the structure containing the state to restore.

Comments This function restores the entire context of the PLX and DSP related to the bridge to the PCI bus. This function is useful when it is necessary to briefly access a memory or I/O location without disturbing the existing setup of these registers.

See Also Direct PCI Access by DSP (PLX)

Pci_SaveCurrentMapping

void Pci_SaveCurrentMapping(
PciMapping_t * *pPciMapping***)**

Save the current setup of the bridge to the PCI bus.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.C

Parameters

pPciMapping
Pointer to the structure to save state into.

Comments

This function saves the entire context of the PLX and DSP related to the bridge to the PCI bus. This function is useful when it is necessary to briefly access a memory or I/O location without disturbing the existing setup of these registers.

See Also

Direct PCI Access by DSP (PLX)

Permit

void Permit()

Enable global interrupts.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.C

Comments

Use this function to enable global interrupts. TI errata state that there is a race condition where certain operations which affect the GIE flag concurrent with a hardware interrupt will cause the GIE to be forced to a 0, disabling further interrupts. Using this function to control the global interrupts alleviates the problem.

See Also

Forbid

ReadFlashSized

void ReadFlashSized(
ubyte * *src***,**
ulong * *dst***,**
ulong *N***,**
int *size***)**

Read a entire buffer to the device, unpacking to size.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FLASH.C

Parameters

src
Location in the FLASH to read from.

dst
Location in the user's buffer.

N
Number of [bytes;words;longs] to transfer.

size
1, 2, or 4 for size of src. [bytes;words;longs].

Return Codes

0	Success.
-1	Failure.

SetIE

void SetIE(
ulong *Value***)**

Set the DSP IE register.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/REGS.ASM

Parameters

Value
New value for the IE register.

SetIF

void SetIF(
ulong *Value***)**

Set the DSP IF register.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/REGS.ASM

Parameters

Value
New value for the IF register.

SetST

void SetST(
ulong *Value***)**

Set the DSP ST register.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/REGS.ASM

Parameters

Value
New value for the ST register.

WriteFlash

int WriteFlash(
ubyte * *src***,**
ubyte * *dst***,**
ulong *N***,**
boolean *fOutput***)**

Write a entire buffer to the device.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FLASH.C

Parameters

src
Location in the user's buffer.

dst
Location in the FLASH to write the buffer.

N
Number of bytes to transfer

fOutput
Do we output to the console port?

Comments

Tracks progress or failure to the standard output.

Return Codes

0	Success.
-1	Failure.

WriteFlashSized

int WriteFlashSized(
ulong * *src***,**
ubyte * *dst***,**
ulong *N***,**
boolean *fOutput***,**
int *size***)**

Write a entire buffer to the device, packing from larger size.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FLASH.C

Parameters

src
Location in the user's buffer.

dst
Location in the FLASH to write the buffer.

N
Number of [bytes;words;longs] to transfer.

fOutput
Do we output to the console port?

size
1, 2, or 4 for size of src. [bytes;words;longs].

Comments Tracks progress or failure to the standard output.

Return Codes

0	Success.
-1	Failure.

xf0_off

void xf0_off()

Set hardware bit XF0 low.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/REGS.ASM

xf0_on

void xf0_on()

Set hardware bit XF0 high.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/REGS.ASM

Constants and Typedefs

Data Types.

DMA0_CNT constant

extern volatile int * const DMA0_CNT;

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

DSP DMA Engine 0 Count Register Address.

DMA0_CTRL constant

extern volatile int * const DMA0_CTRL;

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

DSP DMA Engine 0 Control Register Address.

DMA0_DEST constant

extern volatile int * const DMA0_DEST;

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

DSP DMA Engine 0 Destination Register Address.

DMA0_SRC constant

extern volatile int * const DMA0_SRC;

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

DSP DMA Engine 0 Source Register Address.

DMA1_CNT constant

extern volatile int * const DMA1_CNT;

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

DSP DMA Engine 1 Count Register Address.

DMA1_CTRL constant

extern volatile int * const DMA1_CTRL;

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

DSP DMA Engine 1 Control Register Address.

DMA1_DEST constant

extern volatile int * const DMA1_DEST;

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

DSP DMA Engine 1 Destination Register Address.

DMA1_SRC constant

extern volatile int * const DMA1_SRC;

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

DSP DMA Engine 1 Source Register Address.

T0_CNTR constant

extern volatile int * const T0_CNTR;

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

DSP Timer 0 Count Register Address.

T0_CTRL constant

extern volatile int * const T0_CTRL;

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

DSP Timer 0 Control Register Address.

T0_PERIOD constant

extern volatile int * const T0_PERIOD;

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

DSP Timer 0 Period Register Address.

T1_CNTR constant

extern volatile int * const T1_CNTR;

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

DSP Timer 1 Count Register Address.

T1_CTRL constant

extern volatile int * const T1_CTRL;

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

DSP Timer 1 Control Register Address.

T1_PERIOD constant

extern volatile int * const T1_PERIOD;

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

DSP Timer 1 Period Register Address.

Defines

Constant data.

ACCESS_FLASH

#define ACCESS_FLASH (Appropriate BUS configuration)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Set the external bus control registers to access FLASH memory at 0x900000.

ACCESS_HOST_BYTE

#define ACCESS_HOST_BYTE (Appropriate BUS configuration)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Set the external bus control registers to access pass-thru region to HOST memory in BYTE unsigned mode.

Comments Consecutive BYTEs of HOST memory are presented in consecutive DSP DWORDS with the MSBs zeroed.

ACCESS_HOST_CHAR

#define ACCESS_HOST_CHAR (Appropriate BUS configuration)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Set the external bus control registers to access pass-thru region to HOST memory in signed 8 bit CHAR mode.

Comments Consecutive CHARs of HOST memory are presented in consecutive DSP DWORDS with the MSBs sign extended from bit 7.

ACCESS_HOST_DWORD

#define ACCESS_HOST_DWORD (Appropriate BUS configuration)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Set the external bus control registers to access pass-thru region to HOST memory in unsigned DWORD mode.

Comments Consecutive DWORDs of HOST memory are presented in consecutive DSP DWORDS. All 32 bits are significant.

ACCESS_HOST_INT

#define ACCESS_HOST_INT (Appropriate BUS configuration)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Set the external bus control registers to access pass-thru region to HOST memory in signed INT mode.

Comments Consecutive DWORDs of HOST memory are presented in consecutive DSP DWORDS. All 32 bits are significant.

ACCESS_HOST_SINT

#define ACCESS_HOST_SINT (Appropriate BUS configuration)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Set the external bus control registers to access pass-thru region to HOST memory in signed 16 bit INT mode.

Comments Consecutive SINTs of HOST memory are presented in consecutive DSP DWORDS with the MSBs sign extended from bit 15.

ACCESS_HOST_WORD

#define ACCESS_HOST_WORD (Appropriate BUS configuration)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Set the external bus control registers to access pass-thru region to HOST memory in unsigned WORD mode.

Comments Consecutive WORDs of HOST memory are presented in consecutive DSP DWORDS with the MSBs zeroed.

CACHE_CLEAR

#define CACHE_CLEAR (1 << 12)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

ST bit to clear the cache contents.

CACHE_ENABLE

#define CACHE_ENABLE (1 << 11)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

ST bit to enable fetching from the DSP cache.

CACHE_FREEZE

#define CACHE_FREEZE (1 << 10)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

ST bit to freeze the DSP cache.

CLK_FREQ

#define CLK_FREQ (hardware_ClockSpeedDsp)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Clock speed supplied to DSP.

CLK_INT

#define CLK_INT (1 << 9)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Timer control bit to select the clock source.

Comments When set to 1, the DSP clock is used by the timer. When set to 0, an external source provides the clock.

See Also **T0_CTRL T1_CTRL**

CNT_PERIOD

#define CNT_PERIOD (H1_PERIOD * 2.0)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Clock period supplied to internal timers.

CP_MODE

#define CP_MODE (1 << 8)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Timer control bit to set pulse or square wave mode.

Comments When set to 1, square wave mode is selected. When set to 0, pulse mode is selected. Note that the output frequency is one half when in square wave mode.

See Also **T0_CTRL T1_CTRL**

CPU_EDINT0

#define CPU_EDINT0 (1 << 10)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP CPU interrupt from DSP DMA engine 0.

CPU_EDINT1

#define CPU_EDINT1 (1 << 11)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP CPU interrupt from DSP DMA engine 1.

CPU_EINT0

#define CPU_EINT0 (1 << 0)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP CPU interrupt from External Interrupt 0.

CPU_EINT1

#define CPU_EINT1 (1 << 1)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP CPU interrupt from External Interrupt 1.

CPU_EINT2

#define CPU_EINT2 (1 << 2)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP CPU interrupt from External Interrupt 2.

CPU_EINT3

#define CPU_EINT3 (1 << 3)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP CPU interrupt from External Interrupt 3.

CPU_ERINT0

#define CPU_ERINT0 (1 << 5)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP CPU interrupt from DSP Serial 0 Receive.

CPU_ETINT0

#define CPU_ETINT0 (1 << 8)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP CPU interrupt from DSP Timer 0.

CPU_ETINT1

#define CPU_ETINT1 (1 << 9)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP CPU interrupt from DSP Timer 1.

CPU_EXINT0

#define CPU_EXINT0 (1 << 4)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP CPU interrupt from DSP Serial 0 Transmit.

DMA0_EDINT1

#define DMA0_EDINT1 (1 << 26)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 0 interrupt from DSP DMA engine 1.

DMA0_EINT0

#define DMA0_EINT0 (1 << 16)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 0 interrupt from External Interrupt 0.

DMA0_EINT1

#define DMA0_EINT1 (1 << 17)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 0 interrupt from External Interrupt 1.

DMA0_EINT2

#define DMA0_EINT2 (1 << 18)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 0 interrupt from External Interrupt 2.

DMA0_EINT3

#define DMA0_EINT3 (1 << 19)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 0 interrupt from External Interrupt 3.

DMA0_ETINT0

#define DMA0_ETINT0 (1 << 24)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 0 interrupt from DSP Timer 0.

DMA0_ETINT1

#define DMA0_ETINT1 (1 << 25)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 0 interrupt from DSP Timer 1.

DMA0_EXINT0

#define DMA0_EXINT0 (1 << 20)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 0 interrupt from DSP Serial 0 Transmit.

DMA1_EDINT0

#define DMA1_EDINT0 (1 << 27)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 1 interrupt from DSP DMA engine 0.

DMA1_EINT0

#define DMA1_EINT0 (1 << 28)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 1 interrupt from External Interrupt 0.

DMA1_EINT1

#define DMA1_EINT1 (1 << 29)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 1 interrupt from External Interrupt 1.

DMA1_EINT2

#define DMA1_EINT2 (1 << 30)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 1 interrupt from External Interrupt 2.

DMA1_EINT3

#define DMA1_EINT3 (1 << 31)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 1 interrupt from External Interrupt 3.

DMA1_ERINT0

#define DMA1_ERINT0 (1 << 21)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 1 interrupt from DSP Serial 0 Receive.

DMA1_ETINT0

#define DMA1_ETINT0 (1 << 22)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 1 interrupt from DSP Timer 0.

DMA1_ETINT1

#define DMA1_ETINT1 (1 << 23)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

IE bit to enable DSP DMA 1 interrupt from DSP Timer 1.

GLOBAL_IE

#define GLOBAL_IE (1 << 13)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

ST bit to enable global interrupts.

Comments Do not call **SetST** with this flag. Instead call **Permit** and **Forbid**.

H1_PERIOD

#define H1_PERIOD ((1.0 / CLK_FREQ) * 2.0)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

H1 / H3 clock period.

MBOX1

#define MBOX1 0

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Mailbox 1.

MBOX2

#define MBOX2 1

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Mailbox 2.

MBOX3

#define MBOX3 2

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Mailbox 3.

MBOX4

#define MBOX4 3

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Mailbox 4.

MICRO_SEC

#define MICRO_SEC ((int)((x*1E-6)/H1_PERIOD)-12)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Determine the number of H1/H3 cycles necessary for the specified delay.

Comments Since the calculations are somewhat lengthy, it is wise to calculate and save this value before delaying for short intervals.

See Also **delay**

MILLI_SEC

#define MILLI_SEC ((int)((x*1E-3)/H1_PERIOD)-12)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Determine the number of H1/H3 cycles necessary for the specified delay.

See Also **delay**

SECONDS

#define SECONDS ((int)((x)/H1_PERIOD)-12)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Determine the number of H1/H3 cycles necessary for the specified delay.

See Also **delay**

TIMER_GO

#define TIMER_GO (1 << 6)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Timer control bit to reset and start the timer, if it is not held.

See Also **T0_CTRL T1_CTRL**

TIMER_HLD

#define TIMER_HLD (1 << 7)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Timer control bit to hold the timer.

Comments If this bit is 0, the timer is held. If this bit is 1, the timer is free to start or continue.

See Also **T0_CTRL T1_CTRL**

TIMER_MICROSECONDS

#define TIMER_MICROSECONDS (int)((x*1E-6)/CNT_PERIOD + 0.50001)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Determine the number of H1/H3 cycles necessary for the specified delay.

Comments This function calculates the value to be programmed in the timer register for the specified delay, given the DSP frequency. The value determined is that for pulse output mode. Square wave output will have a period twice as long.

TIMER_MILLISECONDS

#define TIMER_MILLISECONDS (int)((x*1E-3)/CNT_PERIOD + 0.50001)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Determine the number of H1/H3 cycles necessary for the specified delay.

Comments This function calculates the value to be programmed in the timer register for the specified delay, given the DSP frequency. The value determined is that for pulse output mode. Square wave output will have a period twice as long.

TIMER_SECONDS

#define TIMER_SECONDS (int)((x)/CNT_PERIOD + 0.50001)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Determine the number of H1/H3 cycles necessary for the specified delay.

Comments This function calculates the value to be programmed in the timer register for the specified delay, given the DSP frequency. The value determined is that for pulse output mode. Square wave output will have a period twice as long.

USERS_TEXT

#define USERS_TEXT (0x6000)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Starting location of the users code.

XFER_BUFFER_USERS_BSS

#define XFER_BUFFER_USERS_BSS (0x10000)

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Starting location of the Tektronix transfer area.

Structures and Enumerations

Data objects used in this library.

ConfigAddr_t Structure

```
typedef struct {
    unsigned ConfigType : 2;
    unsigned Register   : 6;
    unsigned Function   : 3;
    unsigned Device     : 5;
    unsigned Bus        : 8;
    unsigned Unused     : 7;
    unsigned Enable     : 1;
    ulong u;
} ConfigAddr_t;
```

This structure represents a PCI configuration space address.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.H

Members

ConfigType : 2
00 if device is on the same bus as the PLX, 01 if it is on a sub-bus of the PLX bus.

Register : 6
Register to access in device.

Function : 3
Function in the specified device. (typically 0)

Device : 5
Device number to access.

Bus : 8
Bus number to access.

Unused : 7
Not used.

Enable : 1
Set to 1 by functions.

u
The entire structure as an unsigned value.

Fifo Structure

```
typedef struct {
    ulong Write;
    ulong Read;
    ulong Mask;
    ulong * pBuffer;
    ulong ReadBlockSize;
    ulong WriteBlockSize;
```

```
    EventHandler_t pDataAvailable;
    ulong AmountData;
    void * pDataAvailableParm;
    EventHandler_t pSpaceAvailable;
    ulong AmountSpace;
    void * pSpaceAvailableParm;
} Fifo;
```

DWORD FIFO structure (object).

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/FIFO.H

Members

Write
Current write location.

Read
Current read location.

Mask
Mask ORed with the above for wrapping.

pBuffer
Allocated buffer located on heap.

ReadBlockSize
Size of the active block read.

WriteBlockSize
Size of the active block write.

pDataAvailable
Data Available Event.

AmountData
How many must be available for the event?

pDataAvailableParm
User parameter.

pSpaceAvailable
Space Available Event.

AmountSpace
How many must be available for the event?

pSpaceAvailableParm
User parameter.

Comments

Think of these as private class members.

hardware_DeviceId_t

```
enum hardware_DeviceId_t {
    DeviceID_CPCI_AD8,
    DeviceID_CPCI_IPM,
    DeviceID_CPCI_AD320,
    DeviceID_CPCI_SERVO,
    DeviceID_PMC_AD8,
    DeviceID_PCI_4IPM,
    DeviceID_Plx
};
```

Recognized PCI Device IDs.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Members

DeviceID_CPCI_AD8
CPCI-AD8

DeviceID_CPCI_IPM
CPCI-IPM

DeviceID_CPCI_AD320
CPCI-AD320

DeviceID_CPCI_SERVO
CPCI-SERVO

DeviceID_PMC_AD8
PMC-AD8

DeviceID_PCI_4IPM
PCI-4IPM

DeviceID_Plx
Default (unprogrammed) PLX.

---

hardware_IpSlot_t Structure

```
typedef struct {
    ulong * pIdSpace;
    ulong * pIoSpace;
    ulong * pMemSpace;
    ulong * pIntAckSpace;
} hardware_IpSlot_t;
```

Definition of an IP slot (as seen from the DSP).

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Members

pIdSpace
Where to find the ID space.

pIoSpace
Where to find the IO space.

pMemSpace
Where to find the Memory space. See **hardware_SelectAddressSpace** to switch address spaces.

pIntAckSpace
Where to find the interrupt vector space.

---

hardware_SerialType_t

```
enum hardware_SerialType_t {
    SERIAL_NONE,
    SERIAL_8530,
```

```
    SERIAL_INTERNAL
};
```

Recognized serial devices.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Members

SERIAL_NONE
None.

SERIAL_8530
Dual channel SC8530, with the first as console.

SERIAL_INTERNAL
Internal simulated serial device in ALTERA programmed logic.

---

hardware_VendorId_t

```
enum hardware_VendorId_t {
    VendorId_Alphi,
    VendorId_Plx
};
```

Recognized PCI Vendor IDs.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Members

VendorId_Alphi
ALPHI Technology PCI Card.

VendorId_Plx
Default (unprogrammed) PLX.

InterruptVectors_t Structure

```
typedef struct {
    Handler_t INT0;
    Handler_t INT1;
    Handler_t INT2;
    Handler_t INT3;
    Handler_t XINT0;
    Handler_t RINT0;
    Handler_t XINT1;
    Handler_t RINT1;
    Handler_t TINT0;
    Handler_t TINT1;
    Handler_t DINT0;
    Handler_t DINT1;
    Handler_t TRAP[32];
} InterruptVectors_t;
```

This is the form of the interrupt vector table hardware_InterruptVectors.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Members

INT0
Handler for external interupt 0.

INT1
Handler for external interupt 1.

INT2
Handler for external interupt 2.

INT3
Handler for external interupt 3.

XINT0
Handler for Serial 0 transmitter.

RINT0
Handler for Serial 0 receiver.

XINT1
Handler for Serial 1 transmitter.

RINT1
Handler for Serial 1 receiver.

TINT0
Handler for Timer 0.

TINT1
Handler for Timer 1.

DINT0
Handler for DMA Engine 0.

DINT1
Handler for DMA Engine 1.

TRAP[32]
Handlers for trap instructions.

Developer Notes

Although this library will automatically hook into user provided interrupt handlers, if they are named correctly, occasionally it is necessary to directly change the interrupt vector table directly. This is the form of the table. It is normally located in low memory.

See Also

Handler_t

PciMapping_t Structure

```
typedef struct {
    ulong strb0;
    ulong strb1;
    ulong dmrr;
    ulong dmlbam;
    ulong dmlbai;
    ulong dmpbam;
    ulong dmcfga;
} PciMapping_t;
```

Context of the PCI mapping for backup/restore.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/PCI.H

Members

strb0
DSP register strb0.

strb1
DSP register strb1.

dmrr
PLX register dmrr.

dmlbam
PLX register dmlbam.

dmlbai
PLX register dmlbai.

dmpbam
PLX register dmpbam.

dmcfga
PLX register dmcfga.

Comments

Pci_SaveCurrentMapping and **Pci_RestoreCurrentMapping** allow a temporary backup of the current mapping to be made, so that a short term alternative mapping can be set up. It is most helpful to perform this inside of an interrupt routine, when both the main routine and the interrupt routine will be affecting the PCI mapping.

Developer Notes

Don't rely upon this structure not changing!

See Also

Pci_SaveCurrentMapping Pci_RestoreCurrentMapping

PLX_REG Structure

```
typedef struct {
    volatile ulong pciidr;
    volatile ulong pcicsr;
    volatile ulong pciccr;
    volatile ulong pcihtr;
    volatile ulong pcibar0;
    volatile ulong pcibar1;
    volatile ulong pcibar2;
    volatile ulong pcibar3;
    volatile ulong pcibar4;
    volatile ulong pcibar5;
    volatile ulong pcicis;
    volatile ulong pcisvid;
    volatile ulong pcierbar;
    volatile ulong pciintlat;
    volatile ulong las0rr;
    volatile ulong las0ba;
    volatile ulong barbr;
    volatile ulong bigend;
    volatile ulong eromrr;
    volatile ulong eromba;
    volatile ulong lbrd0;
    volatile ulong dmrr;
    volatile ulong dmlbam;
    volatile ulong dmlbai;
    volatile ulong dmpbam;
```

```
    volatile ulong dmcfga;
    volatile ulong oplfis;
    volatile ulong oplfim;
    volatile ulong mbox[8];
    volatile ulong p2ldbell;
    volatile ulong l2pdbell;
    volatile ulong intcsr;
    volatile ulong cntrl;
    volatile ulong pcihidr;
    volatile ulong pcihrev;
    volatile ulong dmamode0;
    volatile ulong dmapadr0;
    volatile ulong dmaladr0;
    volatile ulong dmasiz0;
    volatile ulong dmadpr0;
    volatile ulong dmamode1;
    volatile ulong dmapadr1;
    volatile ulong dmaladr1;
    volatile ulong dmasiz1;
    volatile ulong dmadpr1;
    volatile ulong dmacsr;
    volatile ulong dmaarb;
    volatile ulong dmathr;
    volatile ulong mqcr;
    volatile ulong qbar;
    volatile ulong ifhpr;
    volatile ulong iftpr;
    volatile ulong iphpr;
    volatile ulong ipfpr;
    volatile ulong ofhpr;
    volatile ulong oftpr;
    volatile ulong ophpr;
    volatile ulong opfpr;
    volatile ulong qsr;
    volatile ulong las1rr;
    volatile ulong las1ba;
    volatile ulong lbrd1;
} PLX_REG;
```

PLX 9080 Operation Registers as seen from the DSP.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Members

pciidr
PCI Configuration ID register.

pcicsr
PCI Command and status register.

pciccr
PCI Class code and revision register.

pcihtr
PCI Cache line size, latency timer, header type, BIST register.

pcibar0
PCI Base address for memory access to PLX registers.

pcibar1
PCI Base address for I/O access to PLX registers.

pcibar2
PCI Base address for memory access to local region 0.

pcibar3
PCI Base address for memory access to local region 1.

pcibar4
PCI Unused base address register.

pcibar5
PCI Unused base address register.

pcicis
PCI Unused register.

pcisvid
PCI Subsystem Vendor ID register.

pcierbar
PCI Unused base address register for expansion ROM.

pciintlat
PCI Interrupt and latency registers.

las0rr
PCI to local address space 0 range.

las0ba
PCI to local address space 0 base address.

barbr
Mode / Arbitration Register.

bigend
Big/Litle Endian Descriptor Register.

eromrr
Expansion ROM range register.

eromba
Expansion ROM base address.

lbrd0
local address space 0 / EROM descriptor register.

dmrr
Local to PCI range register.

dmlbam
Local to PCI base address memory.

dmlbai
Local to PCI base address I/O or CFG.

dmpbam
Local to PCI remap register for memory.

dmcfga
Local to PCI configuration register for I/O or CFG.

oplfis
Outbound post list FIFO interrupt status register.

oplfim
Outbound post list FIFO interrupt mask register.

mbox[8]
Mailbox registers.

p2ldbell
PCI to local doorbell register.

l2pdbell
Local to PCI doorbell register.

intcsr
Interrupt control / status register.

cntrl
Serial NVRAM, PCI Commands, User IO, Init.

pcihidr
Permanent configuration ID register.

pcihrev
Permanent revision register.

dmamode0
DMA 0 mode register.

dmapadr0
DMA 0 PCI address register.

dmaladr0
DMA 0 local address register.

dmasiz0
DMA 0 size register (BYTES).

dmadpr0
DMA 0 descriptor pointer register.

dmamode1
DMA 1 mode register.

dmapadr1
DMA 1 PCI address register.

dmaladr1
DMA 1 local address register.

dmasiz1
DMA 1 size register (BYTES).

dmadpr1
DMA 1 descriptor pointer register.

dmacsr
DMA 0/1 control / status register.

dmaarb
Mode / Arbitration Register (repeated).

dmathr
DMA threshold register.

mqcr
Messaging queue configuration register.

qbar
Queue base address register.

ifhpr
Inbound free head pointer register.

iftpr
Inbound free tail pointer register.

iphpr
Inbound post head pointer register.

ipfpr
Inbound post tail pointer register.

ofhpr
Outbound free head pointer register.

oftpr
Outbound free tail pointer register.

ophpr
Outbound post head pointer register.

opfpr
Outbound post tail pointer register.

qsr
Queue status / control register

las1rr
PCI to local address space 1 range.

las1ba
PCI to local address space 1 base address.

lbrd1
PCI to local address space 1 descriptor register.

PlxNvramImage_t Structure

```
typedef struct {
    ushort m_DeviceId;
    ushort m_VendorId;
    ushort m_ClassCode_H;
    ushort m_ClassCode_L;
    ushort m_MinGrant_MaxLatency;
    ushort m_IntLine_IntPin;
    ushort m_MB0_H;
    ushort m_MB0_L;
    ushort m_MB1_H;
    ushort m_MB1_L;
    ushort m_las0rr_H;
    ushort m_las0rr_L;
    ushort m_las0ba_H;
    ushort m_las0ba_L;
    ushort m_barbr_H;
    ushort m_barbr_L;
    ushort m_bigend_H;
    ushort m_bigend_L;
    ushort m_eromrr_H;
    ushort m_eromrr_L;
    ushort m_eromba_H;
    ushort m_eromba_L;
    ushort m_lbrd0_H;
    ushort m_lbrd0_L;
    ushort m_dmrr_H;
    ushort m_dmrr_L;
```

```
    ushort m_dmlbam_H;
    ushort m_dmlbam_L;
    ushort m_dmlbai_H;
    ushort m_dmlbai_L;
    ushort m_dmpbam_H;
    ushort m_dmpbam_L;
    ushort m_dmcfga_H;
    ushort m_dmcfga_L;
    ushort m_SubsysDeviceId;
    ushort m_SubsysVendorId;
    ushort m_las1rr_H;
    ushort m_las1rr_L;
    ushort m_las1ba_H;
    ushort m_las1ba_L;
    ushort m_lbrd1_H;
    ushort m_lbrd1_L;
    ushort m_ExpansionRom_H;
    ushort m_ExpansionRom_L;
    ushort m_UserId_L;
    ushort m_UserId_H;
    ushort m_SizeDualPortRam_L;
    ushort m_SizeDualPortRam_H;
    ushort m_ClockSpeedDsp_L;
    ushort m_ClockSpeedDsp_H;
    ushort m_ClockSpeedSerial_L;
    ushort m_ClockSpeedSerial_H;
    ushort m_BootOption_L;
    ushort m_BootOption_H;
    ushort m_SerialNumber_L;
    ushort m_SerialNumber_H;
    ushort m_HardwareRevision_L;
    ushort m_HardwareRevision_H;
    ushort m_ProgrammedLogicRevision_L;
    ushort m_ProgrammedLogicRevision_H;
} PlxNvramImage_t;
```

PLX 9080 NVRAM Image.

Defined in: D:/ALPHIPCI/ALPHI_IO/C32_PLX/HARDWARE.H

Members

m_DeviceId
Device ID. (0-1)

m_VendorId
Vendor ID. (2-3)

m_ClassCode_H
Class code. (4-5)

m_ClassCode_L
Class code. (6-7)

m_MinGrant_MaxLatency
Minimum grant. (8) Maximum latency. (9)

m_IntLine_IntPin
Interrupt line. (a) Interrupt pin. (b)

m_MB0_H
Mailbox0 Hi Word. (c-d)

m_MB0_L
Mailbox0 Lo Word. (e-f)

m_MB1_H
Mailbox1 Hi Word. (10-11)

m_MB1_L
Mailbox1 Lo Word. (12-13)

m_las0rr_H
PCI to local address space 0 range. (14-15)

m_las0rr_L
PCI to local address space 0 range. (16-17)

m_las0ba_H
PCI to local address space 0 base address. (18-19)

m_las0ba_L
PCI to local address space 0 base address. (1a-1b)

m_barbr_H
Mode / Arbitration Register. (1c-1d)

m_barbr_L
Mode / Arbitration Register. (1e-1f)

m_bigend_H
Big/Litle Endian Descriptor Register. (20-21)

m_bigend_L
Big/Litle Endian Descriptor Register. (22-23)

m_eromrr_H
Expansion ROM range register. (24-25)

m_eromrr_L
Expansion ROM range register. (26-27)

m_eromba_H
Expansion ROM base address. (28-29)

m_eromba_L
Expansion ROM base address. (2a-2b)

m_lbrd0_H
local address space 0 / EROM descriptor register. (2c-2d)

m_lbrd0_L
local address space 0 / EROM descriptor register. (2e-2f)

m_dmrr_H
Local to PCI range register. (30-31)

m_dmrr_L
Local to PCI range register. (32-33)

m_dmlbam_H
Local to PCI base address memory. (34-35)

m_dmlbam_L
Local to PCI base address memory. (36-37)

m_dmlbai_H
Local to PCI base address I/O or CFG. (38-39)

m_dmlbai_L
Local to PCI base address I/O or CFG. (3a-3b)

m_dmpbam_H
Local to PCI remap register for memory. (3c-3d)

m_dmpbam_L
Local to PCI remap register for memory. (3e-3f)

m_dmcfga_H
Local to PCI configuration register for I/O or CFG. (40-41)

m_dmcfga_L
Local to PCI configuration register for I/O or CFG. (42-43)

m_SubsysDeviceId
Device ID. (44-45)

m_SubsysVendorId
Vendor ID. (46-47)

m_las1rr_H
PCI to local address space 1 range. (48-49)

m_las1rr_L
PCI to local address space 1 range. (4a-4b)

m_las1ba_H
PCI to local address space 1 base address. (4c-4d)

m_las1ba_L
PCI to local address space 1 base address. (4e-4f)

m_lbrd1_H
PCI to local address space 1 descriptor register. (50-51)

m_lbrd1_L
PCI to local address space 1 descriptor register. (52-53)

m_ExpansionRom_H
Expansion ROM Address. (54-55)

m_ExpansionRom_L
Expansion ROM Address. (56-57)

m_UserId_L
User ID. (5a-5b)

m_UserId_H
User ID. (58-59)

m_SizeDualPortRam_L
Size of Dual Port RAM in bytes. (5e-5f)

m_SizeDualPortRam_H
Size of Dual Port RAM in bytes. (5c-5d)

m_ClockSpeedDsp_L
Clock speed of the DSP in Hertz. (62-63)

m_ClockSpeedDsp_H
Clock speed of the DSP in Hertz. (60-61)

m_ClockSpeedSerial_L
Clock speed of PCLK at the 8530 in Hertz. (66-67)

m_ClockSpeedSerial_H
Clock speed of PCLK at the 8530 in Hertz. (64-65)

m_BootOption_L
What to do at RESET. (6a-6b)

m_BootOption_H
What to do at RESET. (68-69)

m_SerialNumber_L
Serial number in decimal. (6e-6f)

m_SerialNumber_H
Serial number in decimal. (6c-6d)

m_HardwareRevision_L
Three character string describing hardware version. (72-73)

m_HardwareRevision_H
Three character string describing hardware version. (70-71)

m_ProgrammedLogicRevision_L
Three character string describing FPGA version. (76-77)

m_ProgrammedLogicRevision_H
Three character string describing FPGA version. (74-75)