PLX PCI SDK
SOFTWARE DEVELOPMENT KIT

IOP 480 CPU API
Programmer's Reference Manual

Version 3.0

Copyright© 1999 PLX Technology, Inc. All rights reserved.

Notice

Copyright © 1999, PLX Technology, Inc.. All rights reserved.

This document contains proprietary and confidential information of PLX Technology Inc. (PLX). The contents of this document may not be copied nor duplicated in any form, in whole or in part, without prior written consent from PLX Technology, INC..

PLX provides the information and data included in this document for your benefit, but it is not possible for us to entirely verify and test all of this information in all circumstances, particularly information relating to non-PLX manufactured products. PLX makes no warranties or representations relating to the quality, content or adequacy of this information. Every effort has been made to ensure the accuracy of this manual, however, PLX assumes no responsibility for any errors or omissions in this document. PLX shall not be liable for any errors or for incidental or consequential damages in connection with the furnishing, performance, or use of this manual or the examples herein. PLX assumes no responsibility for any damage or loss resulting from the use of this manual; for any loss or claims by third parties which may arise through the use of this SDK; and for any damage or loss caused by deletion of data as a result of malfunction or repair. The information in this document is subject to change without notice.
Product and Company names are trademarks or registered trademarks of their respective owners.

Document number: IOP480/CPUAPI-SDK-PRM-P1-3.0

PLX SDK License Agreement

PLX SOFTWARE LICENSE AGREEMENT

THIS PLX SOFTWARE DEVELOPMENT KIT INCLUDES PLX SOFTWARE THAT IS LICENSED TO YOU UNDER SPECIFIC TERMS AND CONDITIONS. CAREFULLY READ THE TERMS AND CONDITIONS PRIOR TO USING THIS SOFTWARE DEVELOPMENT KIT. OPENING THIS PACKAGE OR INITIAL USE OF THIS SOFTWARE DEVELOPMENT KIT INDICATES YOUR ACCEPTANCE OF THE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THEM, YOU SHOULD RETURN THE ENTIRE SOFTWARE DEVELOPMENT KIT TO PLX.

LICENSE Copyright (c) 1999 PLX Technology, Inc.

This PLX Software License agreement is a legal agreement between you and PLX Technology, Inc. for the PLX Software Development Kit (SDK), also referred to as "PLX SDK" which is provided on the enclosed PLX CD-ROM, or may be recorded on other media included in this PLX SDK. PLX Technology owns this PLX SDK. The PLX SDK is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties, and is licensed, not sold. If you are a rightful possessor of the PLX SDK, PLX grants you a license to use the PLX SDK as part of or in conjunction with a PLX chip on a **per project basis**. PLX grants this permission provided that the above copyright notice appears in all copies and derivatives of the PLX SDK. Use of any supplied runtime object modules or derivatives from the included source code in any product without a PLX Technology, Inc. chip is strictly prohibited. You obtain no rights other than those granted to you under this license. You may copy the PLX SDK for backup or archival purposes. You are not authorized to use, merge, copy, display, adapt, modify, execute, distribute or transfer, reverse assemble, reverse compile, decode, or translate the PLX SDK except to the extent permitted by law.

GENERAL

If you do not agree to the terms and conditions of this PLX Software License Agreement, do not install or use the PLX SDK and promptly return the entire unused PLX SDK to PLX Technology, Inc. You may terminate your license at any time. PLX Technology may terminate your license if you fail to comply with the terms and conditions of this License Agreement. In either event, you must destroy all your copies of this PLX SDK. Any attempt to sub-license, rent, lease, assign or to transfer the PLX SDK except as expressly provided by this license, is hereby rendered null and void.

WARRANTY

PLX Technology, Inc. provides this PLX SDK AS IS, WITHOUT ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTIBILITY OR FITNESS FOR A PARTICULAR PURPOSE. PLX makes no guarantee or representations regarding the use of, or the results based on the use of the software and documentation in terms of correctness, or otherwise; and that you rely on the software, documentation, and results solely at your own risk. In no event shall PLX be liable for any loss of use, loss of business, loss of profits, incidental, special or, consequential damages of any kind. In no event shall PLX's total liability exceed the sum paid to PLX for the product licensed here under.

PLX Copyright Message Guidelines

The following copyright message along with the following text must appear in all software products generated and distributed which use the PLX API libraries:

"Copyright (c) 1999 PLX Technology, Inc."

Requirements:

- Arial font
- Font size 12 (minimum)
- Bold type
- Must appear as shown above in the first section or the so called "Introduction Section" of all manuals
- Must also appear as shown above in the beginning of source code as a comment

This page is intentionally left blank.

Table of Contents

1. INTRODUCTION 1-1

1.1 ABOUT THE API AND THIS MANUAL 1-1
1.2 WHERE TO GO FROM HERE 1-1

2. IOP 480 CPU API 2-1

2.1 INTRODUCTION 2-1
2.2 CONVENTIONS 2-1
2.2.1 Windows Programming Conventions 2-1
2.3 TERMINOLOGY 2-1
2.4 DEVELOPMENT REQUIREMENTS 2-1
2.5 API DESIGN 2-1
2.5.1 IOP Static Libraries 2-1
2.5.2 Function Quick Reference List 2-1
2.6 API FUNCTIONS DETAILS 2-3
IOP480CpuRegisterRead 2-5
IOP480CpuRegisterWrite 2-7
IOP480CpuDCacheEnable 2-8
IOP480CpuDCacheDisable 2-9
IOP480CpuICacheEnable 2-10
IOP480CpuICacheDisable 2-11
IOP480CpuDCacheFlush 2-12
IOP480CpuDCacheInvalidate 2-13
IOP480CpuICacheInvalidate 2-14
IOP480CpuDCacheLock 2-15
IOP480CpuDCacheUnlock 2-16
IOP480CpuICacheLock 2-17
IOP480CpuICacheUnlock 2-19
IOP480CpuWriteThroughEnable 2-20
IOP480CpuWriteBackEnable 2-21
IOP480CpuGuardEnable 2-22
IOP480CpuGuardDisable 2-23
IOP480CpuConfigBigEndian 2-24
IOP480CpuConfigLittleEndian 2-25
IOP480CpuExceptionEnable 2-26
IOP480CpuExceptionDisable 2-27
IOP480CpuExceptionHandler 2-28
IOP480CpuExceptionStatusGet 2-29
IOP480CpuSetupPIT 2-30
IOP480CpuSetupFIT 2-31
IOP480CpuSetupWatchDog 2-32
IOP480CpuTimeBaseGet 2-34
IOP480CpuSystemCall 2-35
IOP480CpuExtDebugModeEnable 2-36
IOP480CpuExtDebugModeDisable 2-37
IOP480CpuIntDebugModeEnable 2-38
IOP480CpuIntDebugModeDisable 2-39
IOP480CpuDEOnExceptionEnable 2-40
IOP480CpuDEOnExceptionDisable 2-41
IOP480CpuDEOnBranchEnable 2-42
IOP480CpuDEOnBranchDisable 2-44
IOP480CpuDEOnDACEnable 2-46
IOP480CpuDEOnDACDisable 2-48

IOP480CpuDEOnIACEnable *2-50*
IOP480CpuDEOnIACDisable *2-51*
IOP480CpuDEOnInstrEnable *2-52*
IOP480CpuDEOnInstrDisable *2-53*
IOP480CpuDEOnTrapEnable *2-54*
IOP480CpuDEOnTrapDisable *2-55*
IOP480CpuTrap *2-56*
IOP480CpuSetTrap *2-57*
IOP480CpuGetDebugStatus *2-58*
IOP480CpuGetLastReset *2-60*
IOP480CpuReset *2-61*
IOP480CpuDEDisableAll *2-62*
IOP480CpuMMUInvalidateAll *2-63*
IOP480CpuMMUEnable *2-64*
IOP480CpuMMUMap *2-65*
IOP480CpuMMUValidate *2-66*
IOP480CpuMMUInvalidate *2-67*
IOP480CpuMMUSearch *2-68*
IOP480CpuZoneProtectionRead *2-69*
IOP480CpuZoneProtectionWrite *2-70*
IOP480CpuLockIopBus *2-71*
IOP480CpuUnlockIopBus *2-72*
IOP480CpuGoWaitState *2-73*
IOP480CpuGetVersion *2-74*

3. IOP 480 CPU DATA STRUCTURES AND CONSTANTS 3-1

IOP 480 CPU Register Constants *3-2*
Exception Type Constants *3-4*
FIT_TIMER_PERIOD Enum Data Type *3-6*
WDG_TIMER_PERIOD Enum Data Type *3-7*
RESET_TYPE Enum Data Type *3-8*
DAC_SIZE Enum Data Type *3-9*
DEBUG_STATUS Data Structure *3-10*
TLB_ENTRY_DESC Data Structure *3-11*
PROCESSOR_VERSION Data Structure *3-13*

List of Tables

Table 1. CPU Register Access Functions 2-2
Table 2. Cache Operation Related Functions 2-2
Table 3. Exceptions//Timers/ Interrupts Functions 2-2
Table 4. Debugging Functions 2-2
Table 5. Memory Management Unit Functions 2-3
Table 6. Miscellaneous Functions 2-3
Table 7. IOP 480 CPU Registers 2-5

This page is intentionally left blank.

1. Introduction

1.1 About The API And This Manual

This manual presents the IOP 480 CPU API. The API is a set of functions that access the IOP 480 CPU. The goal of the API is to present functions that will allow system programmers to debug and use the IOP 480 CPU under the IOP 480RDK environment.

This manual makes no references to the PCI interface of the IOP 480. If you need to use the PCI interface, please refer to the PCI SDK Programmer's Reference Manual.

Designers should use this manual as a reference to the Central Processing Unit (CPU) in the IOP 480.

This document is intended for IOP 480 embedded designers only. This document may not be of much use for other PLX chips or for host developers.

The complete set of source code is provided for the IOP 480 CPU API. The API is used to demonstrate how to access certain IOP 480 CPU General Purpose Registers (GPRs) and Special Purpose Registers (SPRs), how to use some IOP 480 CPU instructions. The source code is intended for the system programmer or compiler designer. It is probably meaningless for a normal software programmer of the IOP 480 PowerPC CPU because the PowerPC compiler and/or the real-time operating system will take care of using IOP 480 registers and handling PowerPC interrupts.

Because the assembly instructions or C-style code in the IOP 480 CPU API directory are directly related to the IOP 480 CPU state, it requires special instruments such as RiscWatch or other hardware and/or software debuggers to test the source code. The source code is provided here solely as a learning tool for those who really want to peek into the IOP 480 CPU itself. The interrupt service routines in the source code are not optimized. Care has to be taken to port the source code into your system since it makes no assumption about your memory layout for the executable code, data and heap sections at all.

Be warned that tempering with IOP 480 GPRs and SPRs can easily crash your IOP 480 CPU!

1.2 Where To Go From Here

Chapter 2 presents the general information about this document.
Chapter 3 lists all the software details, including details about the IOP 480 CPU API and about how to use these functions.

This page is intentionally left blank.

2. IOP 480 CPU API

2.1 Introduction

The API included in this PCI SDK provides an IOP 480 custom library that can be used only by an IOP 480 application. This section of the manual details the API design, and provides a reference of all the IOP 480 CPU API functions.

2.2 Conventions

Please note:

- *italics* are used to represent variables, and program names;
- `courier` is used to represent source code given as examples.

2.2.1 Windows Programming Conventions

Although the API presented in this document is designed for boards containing the IOP 480 chip from PLX Technology, we used some Windows programming conventions when we designed the API.
Some designers may not be familiar with Windows programming conventions. Therefore, a few conventions have been noted below, for example:

- *PU32 data* is analogous to *U32 *data* or *unsigned long *data;* and
- *IN* and *OUT* are used to distinguish between parameters that are being passed into API functions and parameters that are being returned by API functions.

2.3 Terminology

The totality of this document is intended for the IOP 480 embedded designers only. Therefore, all the functions presented here are available only for IOP 480 applications.

2.4 Development Requirements

The PCI SDK was developed for Window NT 4.0 and Windows 98 operating systems.
The IOP 480 CPU API was developed using a text editor, compiled by the following two compiler sets:

- IBM High C/C++ PowerPC Cross-Compiler, version 1.0 (7/31/96); or
- Compiler and Linker for the PowerPC, version 4.0b or version 4.3p6 by DIAB Data, Inc.

2.5 API Design

2.5.1 IOP Static Libraries

The IOP API included in the PCI SDK has been designed as standard link libraries that should be familiar to most designers. No special linking instructions are required because it should be linked into IOP applications just like other static libraries.

2.5.2 Function Quick Reference List

Presented below is a quick reference of all IOP 480 CPU API functions grouped in the following categories:

- CPU Register Access Functions (Table 1);
- Cache Operation Related Functions (Table 2);
- Exceptions/Timers/Interrupts Functions (Table 3);
- Debugging Functions (Table 4)
- Memory Management Unit functions (Table 5);

- Miscellaneous Functions (Table 6).

Table 1. CPU Register Access Functions

Function Name	Purpose
IOP480CpuReadRegister	Read from a CPU register (returns register value)
IOP480CpuWriteRegister	Write to a CPU register (takes value directly)

Table 2. Cache Operation Related Functions

Function Name	Purpose
IOP480CpuDCacheEnable	Enable the data cache for one or more memory regions.
IOP480CpuDCacheDisable	Flushes and disable the data cache for one or more memory regions.
IOP480CpuICacheEnable	Enable caching the instructions in one or more memory regions.
IOP480CpuICacheDisable	Invalidate and disable instruction caching in one or more memory regions.
IOP480CpuDCacheLock	Caches a part of memory in the data cache and lock it there.
IOP480CpuDCacheUnlock	Flushes and unlock a part of memory that was in the data cache.
IOP480CpuICacheLock	Cache instructions in the instruction cache and lock them there.
IOP480CpuICacheUnlock	Unlock and invalidate a part of the instruction cache.
IOP480CpuDCacheFlush	Flushes a part of the data cache to memory.
IOP480CpuDCacheInvalidate	Invalidate a part of the data cache.
IOP480CpuICacheInvalidate	Invalidate a part of the instruction cache.
IOP480CpuWriteThroughEnable	Configures a memory region for the write-through caching strategy.
IOP480CpuWriteBackEnable	Configures a memory region for the write-back caching strategy.
IOP480CpuGuardEanble	Configure the IOP 480 to prevent speculative accesses in a memory region.
IOP480CpuGuardDisable	Allow speculative accesses in one or more memory region.
IOP480CpuConfigBigEndian	Configure a memory region for big endian access.
IOP480CpuConfigLittleEndian	Configure a memory region for little endian access.

Table 3. Exceptions//Timers/ Interrupts Functions

Function Name	Purpose
IOP480CpuExceptionEnable	Enable an exception type to interrupt normal instruction processing.
IOP480CpuExceptionDisable	Prevents an exception to interrupt normal processing.
IOP480CpuExceptionHandler	Changes the exception handler to the one specified by the function.
IOP480CpuExceptionStatusGet	Determine if an exception has occurred and which ones.
IOP480CpuSetupPIT	Configure the Programmable Interval Timer (PIT).
IOP480CpuSetupFIT	Configure the Fixed Interval Timer (FIT).
IOP480CpuSetupWatchDog	Configure the WatchDog timer.
IOP480CpuSystemCall	Make a system call
IOP480CpuTimeBaseGet	Returns the current count of IOP 480 time base register.

Table 4. Debugging Functions

Function Name	Purpose
IOP480CpuExternalDebugModeEnable	Enable the external debug mode.
IOP480CpuExternalDebugModeDisable	Disable the external debug mode.
IOP480CpuInternalDebugModeEnable	Enable the internal debug mode.
IOP480CpuInternalDebugModeDisable	Disable the internal debug mode.

Function Name	Purpose
IOP480CpuDEOnExceptionEnable	Enable the exception to generate debug events.
IOP480CpuDEOnExceptionDisable	Disable the exceptions from generation debug events.
IOP480CpuDEOnBranchEnable	Enable debug events on branches.
IOP480CpuDEOnBranchDisable	Do not generate debug events on branches.
IOP480CpuDEOnDACEnable	Generate debug events on successful data address compare.
IOP480CpuDEOnDACDisable	Do not generate debug events on successful data address compare.
IOP480CpuDEOnIACEnable	Generate debug events on successful instruction address compare.
IOP480CpuDEOnIACDisable	Do not generate debug events on successful instruction address compare.
IOP480CpuDEOnInstrEnable	Generate debug events on instruction completion.
IOP480CpuDEOnInstrDisable	Do not generate debug events on instruction completion.
IOP480CpuDEOnTrapEnable	Generate debug events on traps.
IOP480CpuDEOnTrapDisable	Do not generate debug events on trap.
IOP480CpuTrap	Trap instruction.
IOP480CpuSetTrap	Place a trap instruction.
IOP480CpuGetDebugStatus	Get the debug event status.
IOP480CpuReset	Reset the IOP 480 CPU.
IOP480CpuGetLastReset	Retrieve the last reset type.
IOP480CpuDEDisableAll	Disable all debug events.

Table 5. Memory Management Unit Functions

Function Name	Purpose
IOP480CpuMMUEnable	Enable or disable the MMU.
IOP480CpuMMUInvalidateAll	Invalidate all entries in the MMU.
IOP480CpuMMUMap	Map addresses in an MMU entry.
IOP480CpuMMUInvalidate	Invalidate an MMU entry.
IOP480CpuMMUValidate	Validate an MMU entry.
IOP480CpuMMUSearch	Search an entry in the MMU.
IOP480CpuZoneProtectionRead	Read zone protection settings.
IOP480CpuZoneProtectionWrite	Set up zone protection settings.

Table 6. Miscellaneous Functions

Function Name	Purpose
IOP480CpuGoWaitState	Enter the wait state.
IOP480CpuGetVersion	Get the CPU version.

2.6 API Functions Details

This section contains a detailed description of each function in the IOP 480 CPU API. The functions are listed by category.

The following sample entry lists each section and describes the information therein.

Note: *The only device supported by the functions in this document is the IOP 480.*

Sample Function Entry

Syntax:
function(modifier *parameter*[,...]);
This gives the declaration syntax for each function. Each parameter is *italicized*.
Description:
Summary of the function's purpose followed by the parameters it takes. Also includes any relevant information pertaining to the function.
Return Value:
The value returned by the function.
Portability:
States whether this function can be used with Win32, IOP applications or both.
Usage:
A sample is provided to demonstrate the function's use.

IOP480CpuRegisterRead

Syntax:

```
U32 IOP480CpuRegisterRead(IN U32 registerNumber);
```

Description:

Reads any register of the IOP 480 CPU.

- *registerNumber* is the register number.

***Note**: This function is a general register-accessing function and it is better to use a more specific register-accessing function if it exists.*

Return Value:

This function returns the data value read from register *registerNumber* of the type *U32*.
This function does not return any error code. In the case the register's number is invalid, the results would be 0xFFFFFFFF;

Usage:

```
U32 value;

/* Read the LR register */
value = IOP480CpuRegisterRead(CPU480_LR);
PlxPrintf("Value of the link register is 0x%x \n", value);
```

IOP 480 CPU Registers

All the registers have the same format: CPU480_XXX where XXX is the IOP 480 CPU register name. These names are given in the table below. For example, Machine State Register would have the register number CPU480_MSR.

Table 7. IOP 480 CPU Registers

Register name	Category	Meaning
MSR*	MSR	Machine State Register
CR	CR	Condition Register
CDBCR*	SPR	Cache Debug Control Register
CTR	SPR	Count Register
DAC1*	SPR	Data Address Compare 1
DBCR*	SPR	Debug Control Register
DBSR*	SPR	Debug Status Register
DCCR*	SPR	Data Cache Cacheability Register
DCWR*	SPR	Data Cache Write-through Register
DEAR*	SPR	Data Error Address Register
ESR*	SPR	Exception Syndrome Register
EVPR*	SPR	Exception Vector Prefix Register
IAC1*	SPR	Instruction Address Compare 1
ICCR*	SPR	Instruction Cache Cacheability Register
ICDBDR*	SPR	Instruction Cache Debug Data Register
LR	SPR	Link Register
PID*	SPR	Process ID
PIT*	SPR	Programmable Interval Timer

PVR*	SPR	Processor Version Register
SGR*	SPR	Storage Guarded Register
SKR*	SPR	Storage Compression Register
SLER*	SPR	Storage Little-Endian Register
SPRG0*	SPR	SPR General Purpose Register G0
SPRG1*	SPR	SPR General Purpose Register G1
SPRG2*	SPR	SPR General Purpose Register G2
SPRG3*	SPR	SPR General Purpose Register G3
SRR0*	SPR	Save/Restore Register 0
SRR1*	SPR	Save/Restore Register 1
SRR2*	SPR	Save/Restore Register 2
SRR3*	SPR	Save/Restore Register 3
TBHI*	SPR	Time Base High
TBHU**	SPR	Time Base High User-mode
TBLO*	SPR	Time Base Low
TBLU**	SPR	Time Base Low User-mode
TCR*	SPR	Timer Control Register
TSR*	SPR	Timer Status Register
XER	SPR	Fixed Point Exception Register
ZPR*	SPR	Zone Protection Register

Note:
**: Access to this register is restricted to privileged mode only.*
***: This register cannot be written to from user mode.*
1) Some of these registers are read-only.
2) To see a more complete description, refer to the IOP 480 Data Sheet, chapter 29.

IOP480CpuRegisterWrite

Syntax:

```
RETURN_CODE IOP480CpuRegisterWrite(IN U32 registerNumber,
                                   IN U32 value);
```

Description:

Write to any register of the IOP 480 CPU.

- *registerNumber* is the register number; and,
- *value* is the value to write to the register.

***Note**: This function is a general register-accessing function and it is better to use a more specific register-accessing function if it exists.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480
`ApiInvalidRegister`	The *registerNumber* parameter is invalid.	IOP 480

Usage:

```
RETURN_CODE rc;

/* Write to the ICCR register */
rc = IOP480CpuRegisterWrite(CPU480_ICCR, 0x01000000);
if (rc != ApiSuccess)
  PlxPrintf("Failed to write to the register\n");
```

IOP 480 CPU Registers

To see a complete list of the IOP 480 CPU registers, please refer to the IOP480CpuRegisterRead function entry.

IOP480CpuDCacheEnable

Syntax:

RETURN_CODE IOP480CpuDCacheEnable(IN PU32 *address*);

Description:

Enable the data cache for a specific address.

- *address* is the address of the memory location to have his cache enabled.

Note:

1. *The data cache is not enabled at power up.*
2. *This function can only be called from privileged mode.*
3. *This function will mark the memory space indicated by* ***address*** *as cacheable in the Data Cache Cacheability Register (DCCR) and does not perform any check in the Memory Management Unit.*
4. *When data address translation is enabled(MSR[DR] = 1), data cacheability is controlled by the I bit in the TLB entry for the memory page. If TLB_Entry[I] = 1, caching is inhibited; otherwise, caching is enabled.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
/* Allow data cache for address 0x00000000 to 0x08000000 */
IOP480CpuDCacheEnable((PU32)0x00000000);
```

IOP480CpuDCacheDisable

Syntax:

RETURN_CODE IOP480CpuDCacheDisable(IN PU32 *address*);

Description:

Disable the data cache for a specific address.

- *address* is the address of the memory location to have his cache disabled.

Note:

1. *The data cache is not enabled at power up.*
2. *This function can only be called from privileged mode.*
3. *This function will mark the memory space indicated by* ***address*** *as non-cacheable in the Data Cache Cacheability Register (DCCR) and does not perform any check in the Memory Management Unit.*
4. *When data address translation is enabled(MSR[DR] = 1), data cacheability is controlled by the I bit in the TLB entry for the memory page. If TLB_Entry[I] = 1, caching is inhibited; otherwise, caching is enabled.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
/* Disable Data cache for address 0x10000000 to 0x18000000 */
IOP480CpuDCacheDisable((PU32)0x10000000);
```

IOP480CpuICacheEnable

Syntax:

RETURN_CODE IOP480CpuICacheEnable(IN PU32 *address*);

Description:

Enable the instruction cache for a specific address.

- *address* is the address of the memory location to have his cache enabled.

Note:

1. *The instruction cache is not enabled at power up.*
2. *This function can only be called from privileged mode.*
3. *This function will mark the memory space indicated by* ***address*** *as cacheable in the Instruction Cache Cacheability Register (ICCR) and does not perform any check in the Memory Management Unit.*
4. *When instruction address translation is enabled(MSR[IR] = 1), instruction cacheability is controlled by the I storage bit in the Translation Lookaside Buffer(TLB) entry for the memory page. If TLB_Entry[I] = 1, caching is inhibited; otherwise, caching is enabled.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
/* Allow data cache for address 0x00000000 to 0x08000000 */
IOP480CpuICacheEnable((PU32)0x00000000);
```

IOP480CpuICacheDisable

Syntax:

RETURN_CODE IOP480CpuICacheDisable(IN PU32 *address*);

Description:

Disable the instruction cache for a specific address.

- *address* is the address of the memory location to have his cache disabled.

Note:

1. *The instruction cache is not enabled at power up.*
2. *This function can only be called from privileged mode.*
3. *This function will mark the memory space indicated by* ***address*** *as non-cacheable in the Instruction Cache Cacheability Register (ICCR) and does not perform any check in the Memory Management Unit.*
4. *When instruction address translation is enabled(MSR[IR] = 1), instruction cacheability is controlled by the I storage bit in the Translation Lookaside Buffer(TLB) entry for the memory page. If TLB_Entry[I] = 1, caching is inhibited; otherwise, caching is enabled.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
/* Allow data cache for address 0x10000000 to 0x18000000 */
IOP480CpuICacheDisable((PU32)0x00000000);
```

IOP480CpuDCacheFlush

Syntax:

RETURN_CODE IOP480CpuDCacheFlush(IN PU32 *address*,
IN U32 *length*);

Description:

Flushes the data cache for a specific range of addresses.

- *address* is the starting address to start flushing.
- *range* is the number of bytes needed to be flushed.

Note: *The data cache line size is 16 bytes. This function makes sure that the whole range is flushed. So, if the starting and/or ending address are not on a cache line boundary, the whole cache line will be flushed, including the rest of the line. It is not an error to call this function even if the data cache is not enabled for the corresponding memory region.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
/* Flushes a sentence */
char PciMustReadThis[50];
IOP480CpuDCacheEnable((PU32)0x00000000);
strcpy(PciMustReadThis, "If the host can read me, then data cache
                         flushing works ok\n");
IOP480CpuDCacheFlush(PciMustReadThis, strlen(PciMustReadThis));
```

IOP480CpuDCacheInvalidate

Syntax:

RETURN_CODE IOP480CpuDCacheInvalidate(IN PU32 *address*,
IN U32 *length*);

Description:

Invalidates the data cache for a specific range of addresses.

- *address* is the starting address to start invalidating.
- *range* is the number of bytes needed to be invalidated.

Note: *The data cache line size is 16 bytes. This function makes sure that the whole range is invalidated. So, if the starting and/or ending address are not on a cache line boundary, the whole cache line will be invalidated, including the rest of the line. Therefore, there is a danger of losing data that needs to be kept. To make sure this works OK, align the data that needs to be invalidated on a 16 bytes boundary. It is not an error to call this function even if the data cache is not enabled for the corresponding memory region. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

/*
  Invalidate whatever is in the data cache for addresses 0x0 to
  0x01000000.
*/
rc = IOP480CpuDCacheInvalidate((PU32)0x00000000, 0x01000000);
if (rc != ApiSuccess)
  PlxPrintf("Error invalidating the data cache\n");
```

IOP480CpuICacheInvalidate

Syntax:

RETURN_CODE IOP480CpuICacheInvalidate(IN PU32 *address*,
IN U32 *length*);

Description:

Invalidates the instruction cache for a specific range of addresses.

- *address* is the starting address to start instruction invalidation; and,
- *range* is the number of bytes needed to be invalidated.

Note: *The instruction cache line size is 16 bytes. This function makes sure that the whole range is invalidated. So, if the starting and/or ending address are not on a cache line boundary, the whole cache line will be invalidated, including the rest of the line. It is not an error to call this function even if the instruction cache is not enabled for the corresponding memory region.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

/*
  Invalidate whatever is in the instruction cache for addresses 0x0
  to 0x01000000.
*/
rc = IOP480CpuICacheInvalidate((PU32)0x00000000, 0x01000000);
if (rc != ApiSuccess)
  PlxPrintf("Error invalidating the data cache\n");
```

IOP480CpuDCacheLock

Syntax:

RETURN_CODE IOP480CpuDCacheLock(IN PU32 *address*,
IN U32 *length*);

Description:

Place and lock a range of data into the data cache. Use this for data that needs to be accessed quickly and often.

- *address* is the starting address of the data range to be locked; and,
- *range* is the number of bytes needed to be locked.

Note: *The data cache line size is 16 bytes. This function tries to lock the whole range. So, if the starting and/or ending address are not on a cache line boundary, the whole cache line may be locked, including the rest of the line. It is not an error to call this function if the data cache is not enabled for the corresponding memory region. This function can only be called from privileged mode.*

Programmer note:

Due to the limited size of the data cache, you might not be able to lock all the lines of data in cache. Use this API call carefully. Another thing to remember is that this API call only implements the following three steps:

1. Enable the line locking in the CDBCR register;
2. Use dcbt to touch the cache blocks;
3. Disable the line locking in the CDBCR register.

Therefore, this API call does not guarantee that the locked lines in cache will remain locked all the times since subsequent data cache instructions can easily remove the locked lines from the cache. For more information, consult the IOP 480 data book on **IOP 480 CPU Cache Operations.**

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
U32 fastAccess[0x10];
RETURN_CODE rc;

/* Lock fastAccess array into data cache */
rc = IOP480CpuDCacheLock(fastAccess, 0x100 * sizeof(U32));
if (rc != ApiSuccess)
  PlxPrintf("Error locking fastAccess into the data cache\n");
```

IOP480CpuDCacheUnlock

Syntax:

RETURN_CODE IOP480CpuDCacheUnlock(IN PU32 *address*,
IN U32 *length*);

Description:

Flushes and unlock a range of data that was previously locked in the data cache.

- *address* is the starting address of the data to be unlocked; and,
- *range* is the number of bytes needed to be unlocked.

Note: *The data cache line size is 16 bytes. This function makes sure that the whole range is flushed and unlocked. So, if the starting and/or ending address are not on a cache line boundary, the whole cache line will be flushed, including the rest of the line. If the function finds the range or part of the range in the data cache but not locked, it simply flushes it, removing it from the data cache. It is not an error to call this function if the data cache is not enabled for the corresponding memory region. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
VOID LocalFunction(VOID)
{
  U32 fastAccess[0x10];
  RETURN_CODE rc;

  /* Lock fastAccess into cache */
  rc = IOP480CpuDCacheLock(fastAccess, 0x10 * sizeof(U32));
  if (rc != ApiSuccess)
    PlxPrintf("Error locking fastAccess into the data cache\n");

  ...
  /* Use fastAccess a lot and fast */
  ...

  rc = IOP480CpuDCacheUnlock(fastAccess, 0x10 * sizeof(U32));
  if (rc != ApiSuccess)
    PlxPrintf("Error unlocking fastAccess from the data cache\n");
}
```

IOP480CpuICacheLock

Syntax:

RETURN_CODE IOP480CpuICacheLock(IN PU32 *address*,
IN U32 *length*);

Description:

This function lock instructions into the instruction cache. A programmer should use this only for a function that needs to be executed quickly and often.

- *address* is the starting address of the code section to be locked.
- *range* is the number of bytes needed to be locked.

Note: *The instruction cache line size is 16 bytes. This function makes sure that the whole range is placed and locked in the instruction cache. So, if the starting and/or ending address are not on a cache line boundary, the whole cache line will be placed and locked, including the rest of the line. It is not an error to call this function if the instruction cache is not enabled for the memory region. In that case, nothing would be done. This function can only be called from privileged mode.*

Programmer note:

Due to the limited size of the instruction cache, you might not be able to lock all the lines of instructions in cache. Use this API call carefully. Another thing to remember is that this API call only implements the following three steps:

1. Enable the line locking in the CDBCR register;
2. Use icbt to touch the cache blocks;
3. Disable the line locking in the CDBCR register.

Therefore, this API call does not guarantee that the locked lines in cache will remain locked all the times since subsequent instruction cache instructions can easily remove the locked lines from the cache. For more information, consult the IOP 480 data book on **IOP 480 CPU Cache Operations.**

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
extern void INeedToRunFast(void);  /* size is 64 bytes */

/* Lock INeedToRunFast in the instruction cache */
rc = IOP480CpuICacheLock((PU32)INeedToRunFast, 64);
if (rc != ApiSuccess)
{
  PlxPrintf("Error locking into the instruction cache\n");
  return;
}

...
/* Use IneedToRunFast as necessary */
...

/* Unlock IneedToRunFast from the I-cache */
rc = IOP480CpuICacheUnlock((PU32)INeedToRunFast, 64);
```

```
if (rc != ApiSuccess)
  PlxPrintf("Error unlocking the instruction cache\n");
```

IOP480CpuICacheUnlock

Syntax:

RETURN_CODE IOP480CpuICacheUnlock(IN PU32 *address*,
IN U32 *length*);

Description:

Unlock and invalidate code from the cache.

- *address* is the starting address of the section to unlock; and,
- *range* is the number of bytes needed to be unlocked.

Note: *The instruction cache line size is 16 bytes. This function makes sure that the whole range is unlocked and invalidated from the instruction cache. So, if the starting and/or ending address are not on a cache line boundary, the whole cache line will be unlocked and invalidated, including the rest of the line. If the function finds the code in the instruction cache, but not locked, the code will be simply invalidated. It is not an error to call this function if the instruction cache is not enabled for the memory region. In that case, nothing would be done. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
extern void INeedToRunFast(void);  /* size is 64 bytes */

/* Lock INeedToRunFast in the instruction cache */
rc = IOP480CpuICacheLock((PU32)INeedToRunFast, 64);
if (rc != ApiSuccess)
{
  PlxPrintf("Error locking into the instruction cache\n");
  return;
}

...
/* Use IneedToRunFast as necessary */
...

/* Unlock IneedToRunFast from the I-cache */
rc = IOP480CpuICacheUnlock((PU32)INeedToRunFast, 64);
if (rc != ApiSuccess)
  PlxPrintf("Error unlocking the instruction cache\n");
```

IOP480CpuWriteThroughEnable

Syntax:

RETURN_CODE IOP480CpuWriteThroughEnable(IN PU32 *address*);

Description:

Specify the write-through strategy for managing the data cache on the memory region.

- *address* is an address within the memory region to have his data cache managed with the write-through strategy.

Note: *When the system boots up, the default strategy for the whole memory ranges is undefined. The write-back strategy is usually faster than write-through strategy. The write-through strategy may cause exceptions with cache related instructions. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

/* Selects the write-through strategy for address 0x10000000 */
rc = IOP480CpuWriteThroughEnable((PU32)0x10000000);
if (rc != ApiSuccess)
  PlxPrintf("Unable to set up the write-through strategy\n");
```

IOP480CpuWriteBackEnable

Syntax:

RETURN_CODE IOP480CpuWriteBackEnable(IN PU32 *address*);

Description:

Selects the write-back strategy for managing the data cache on the specified memory region.

- *address* specify an address of the memory region to be cache-managed with the write-back strategy..

Note: *When the system boots up, the default strategy for the whole memory ranges is undefined. The write-back strategy is usually faster than write-through strategy. The write-through strategy may cause exceptions with cache related instructions. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

/* Selects the write-back strategy for address 0x10000000 */
rc = IOP480CpuWriteBackEnable((PU32)0x10000000);
if (rc != ApiSuccess)
  PlxPrintf("Unable to set up the write-back strategy\n");
```

IOP480CpuGuardEnable

Syntax:

RETURN_CODE IOP480CpuGuardEnble(IN PU32 *address*);

Description:

Prevents speculative accesses on a memory region.

- *address* is an address of the memory region to be guarded.

Note: *At power up, all the memory regions are guarded. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

/* Enable memory guarding for addresses 0x10000000 to 0x18000000 */
rc = IOP480CpuGuardEnable((PU32)0x10000000);
if (rc != ApiSuccess)
  PlxPrintf("Unable to guard\n");
```

IOP480CpuGuardDisable

Syntax:

RETURN_CODE IOP480CpuGuardDisable(IN PU32 *address*);

Description:

Allows speculative accesses on a specific memory region.

- *address* is the memory region to be unguarded.

Note: *At power up, all the memory regions are guarded. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

/* Disable the memory guard for addresses 0x10000000 to 0x18000000 */
rc = IOP480CpuGuardDisable((PU32)0x10000000);
if (rc != ApiSuccess)
  PlxPrintf("Unable to unguard\n");
```

IOP480CpuConfigBigEndian

Syntax:

RETURN_CODE IOP480CpuConfigBigEndian(IN PU32 *address*);

Description:

Configure for big endian accesses one memory region.

- *address* is the memory region to be configured with big-endian.

Note: *This function can only be called from privileged mode. In real mode, the Storage Little-Endian Register (SLER) controls the setting of the E storage attribute.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

/* Set up big endian for addresses 0x10000000 to 0x18000000 */
rc = IOP480CpuConfigBigEndian((PU32)0x10000000);
if (rc != ApiSuccess)
  PlxPrintf("Unable to set big-endian\n");
```

IOP480CpuConfigLittleEndian

Syntax:

RETURN_CODE IOP480CpuConfigLittleEndian(IN PU32 *address*);

Description:

Configure for little endian accesses one memory region.

- *region* is the memory region to be set up with little-endian.

Note: *This function can only be called from privileged mode. In real mode, the Storage Little-Endian Register (SLER) controls the setting of the E storage attribute.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

/* Set up little endian for addresses 0x10000000 to 0x18000000 */
rc = IOP480CpuConfigLittleEndian((PU32)0x10000000);
if (rc != ApiSuccess)
  PlxPrintf("Unable to set little-endian\n");
```

IOP480CpuExceptionEnable

Syntax:

RETURN_CODE IOP480CpuExceptionEnable(IN U32 *exceptionType*);

Description:

This function allows an exception to interrupt normal instruction processing.

- *exceptionType*: one or more exception types. *The exception type constants can be combined together with OR.*

Note: *Some exceptions are always enabled and cannot be disabled. Many exceptions can be enabled by other functions. This function can only be called from privileged mode.*

Cross Reference:

The constants for the exception type are listed on page 3-4.

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

/* Enable external interrupts */
rc = IOP480CpuExceptionEnable(CPU480EX_EXTERNAL);
if (rc != ApiSuccess)
  PlxPrintf("Unable to enable external interrupts\n");
```

IOP480CpuExceptionDisable

Syntax:

RETURN_CODE IOP480CpuExceptionDisable(IN U32 *exceptionType*);

Description:

This function prevents an exception from interrupting normal instruction processing.

- *exceptionType*: one or more exception types. *The exception type constants can be combined together with OR.*

Note: *Some exceptions are always enabled and cannot be disabled. Calling this function with a non-maskable exception will result in the function failure. This function can only be called from privileged mode.*

Cross Reference:

The constants for the exception type are listed on page 3-4.

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

/* Enable external interrupts */
rc = IOP480CpuExceptionDisable(CPU480EX_EXTERNAL);
if (rc != ApiSuccess)
  PlxPrintf("Unable to disable external interrupts\n");
```

IOP480CpuExceptionHandler

Syntax:

void (* IOP480CpuExceptionHandler(IN U32 *exceptionType*,
IN void(**function*)(void)))(void);

Description:

This function sets up a handler for an exception. It returns the old exception handler.

- *exceptionType* is the exception types; and,
- *function* is the new function to be executed for the exception.

Cross Reference:

The constants for the exception type are listed on page 3-4.

Return Value:

On success, this function returns the old exception function.
On error, this function returns 0xFFFFFFFF.

Usage:

```
void (*OldHandler)(void) = (void (*)(void))0xFFFFFFFF;
void MyISR(void)
{
  PlxPrintf("An external interrupt occurred\n");
  if ((U32)OldHandler != 0xFFFFFFFF)
  (*OldHandler)();
}

void main(void)
{
  /* Set up our function for the external interrupt */
  OldHandler = IOP480CpuExceptionHandler(CPU480EX_EXTERNAL, MyISR);
  if (OldHandler != 0xFFFFFFFF)
    IOP480CpuExceptionEnable(EXTERNAL_INTERRUPT);
  else
  PlxPrintf("External interrupt set up failed\n");
}
```

IOP480CpuExceptionStatusGet

Syntax:

RETURN_CODE IOP480CpuExceptionStatusGet(OUT PU32 *pExceptionType*);

Description:

Determines which exceptions are currently active.

- *pExceptionType* is a pointer to an U32 data type.

***Note**: This function can only be called from privileged mode. Due to the fact the interrupt status registers are not be used to indicate all the exceptions when they happen, this API can NOT tell all the exception types except the followings:*

- *CPU480EX_PROGRAM;*
- *CPU480EX_MACHINE;*
- *CPU480EX_DSTORAGE;*
- *CPU480EX_WATCHDOG;*
- *CPU480EX_PIT;*
- *CPU480EX_FIT;*

Cross Reference:

The constants for the exception type are listed on page 3-4.

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480
ApiNullParam	The caller provided a NULL pointer in the paramter.	IOP 480

Usage:

```
RETURN_CODE rc;
U32 exception;

rc = IOP480CpuExceptionStatusGet(&exception);

if (rc != ApiSuccess)
PlxPrintf("Error: unable to get the exception status\n");

if (exception & CPU480EX_WATCHDOG)
    PlxPrintf("A watchdog interrupt is active\n");
```

IOP480CpuSetupPIT

Syntax:

RETURN_CODE IOP480CpuSetupPIT(IN U32 *numberOfCycles*,
IN BOOLEAN *autoReload*);

Description

This function sets up the Programmable Interval Timer parameters. It does not set up a function handling the PIT exception nor enables it.

- *numberOfCycles* is the number of bus cycles to wait before a PIT exception is generated.
- *autoReload* reloads the PIT timer after a PIT exception occurred, allowing it to occur regularly.

***Note**: This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

/* Set up the PIT for one second, auto-reload */
rc = IOP480CpuSetupPIT(0x017D1840, TRUE);  /* Assume clock is 25 Mhz */
if (rc != ApiSuccess)
    PlxPrintf("Failed to set up the PIT\n");
```

IOP480CpuSetupFIT

Syntax:

RETURN_CODE IOP480CpuSetupFIT(IN FIT_TIMER_PERIOD *period*);

Description:

This function sets up the Fixed Interval Timer. It does not enable the fixed interval timer exception nor set up a user exception handler for the FIT exception.

- *period* is the period when each FIT exception would occur.

***Note**: This function can only be called from privileged mode.*

Cross Reference:

Referenced Item	Page
FIT_TIMER_PERIOD	3-6

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

/* Set up the FIT for 0.00524 sec. With a clock of 25 Mhz */
rc = IOP480CpuSetupFIT(Period2E17Clocks);
if (rc != ApiSuccess)
  PlxPrintf("Failed to set up the FIT\n");
```

IOP480CpuSetupWatchDog

Syntax:

RETURN_CODE IOP480CpuSetupWatchDog(IN WDG_TIMER_PERIOD *period*,
IN RESET_TYPE *reset*);

Description:

This function sets up the watchdog timer. It does not enable the watchdog timer exception nor set up an exception handler for the watchdog exception.

- *period* is the period when each watchdog exception would occur.
- *reset* is the reset type that should occur when a timeout would occur and a watchdog interrupt is still pending.

Note: *The watchdog is typically used to recover from software or hardware faults. For more information, please consult the IOP 480 Data Sheet on watchdog timer. Once this function has been called with the reset parameter other than NoReset, then the watchdog reset control cannot be cleared by software except by a soft-induced reset. This function can only be called from privileged mode.*

Cross Reference:

Referenced Item	Page
WDG_TIMER_PERIOD	3-7
RESET_TYPE	3-8

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
void MyWatchDog(void)
{
  /*
    Clear the watchdog interrupt pending bit. Remove that line and the
    chip would reset shortly after the IOP480CpuExceptionEnable(WATCHDOG)
    instruction.
  */
  IOP480CpuRegisterWrite(CPU480_TSR, 0x40000000);
}

void main(void)
{
  RETURN_CODE rc;
  void (*OldWatchDog)(void);

  /* Set up the watchdog timer */
  OldWatchDog = IOP480CpuExceptionHandler(WATCHDOG, MyWatchDog);
  rc = IOP480CpuSetupWatchDog(Period2E21Clocks, SystemReset);
  if (rc != ApiSuccess)
  {
    PlxPrintf("Failed to set up the Watchdog\n");
```

```
    return;
  }

  /* Enable the watchdog */
  rc = IOP480CpuExceptionEnable(WATCHDOG);
  if (rc != ApiSuccess)
  {
    PlxPrintf("Failed to enable the Watchdog\n");
    return;
  }


  ... /* main operation; if the CPU hang, the watchdog will reset. */

  exit(0);
}
```

IOP480CpuTimeBaseGet

Syntax:

RETURN_CODE IOP480CpuTimeBaseGet(OUT PU32 pT*imeBaseUpper*,
OUT PU32 *pTimeBaseLower*);

Description:

This function returns the current count of IOP 480 time base register.

- *pTimeBaseUpper* is a pointer to a 32 bit data which will hold the upper 32-bit count of the time base.
- *pTimeBaseLower* is a pointer to a 32 bit data which will hold the lower 32-bit count of the time base.

Note: *Most usually, the time base indicates how long since the last reset occurred. However, the time base can be adjusted to support Time-of-day functions.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
U32 timeHigh, timeLow;
RETURN_CODE rc;

rc = IOP480CpuTimeBaseGet(&timeHigh, &timeLow);
if (rc != ApiSuccess)
    PlxPrintf("Error: unable to read timer\n");
else
    PlxPrintf("The board has been on since 0x%lX%08lX bus"
              "cycles\n", timeHigh, timeLow);
```

IOP480CpuSystemCall

Syntax:

void IOP480CpuSystemCall(void);

Description:

This function generates a system call exception.
Note: *The system call exception is not maskable.*

Return Value:

Usage:

```
void OurESR(void)
{
      PlxPrintf("A system call occurred!\n");
}

void main(void)
{
  void (*OldESR)(void);
  RETURN_CODE rc;
  OldESR = IOP480CpuExceptionHandler(CPU480EX_SYSTEM, OurESR);

  ...

  rc = IOP480CpuSystemCall();

  /* The OurESR function should have been called */

  ...
}
```

IOP480CpuExtDebugModeEnable

Syntax:

RETURN_CODE IOP480CpuExtDebugModeEnable(void);

Description:

This function enables the external debug mode.
Note: *External debug mode requires the use of the JTAG port. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

rc = IOP480CpuExtDebugModeEnable();
if (rc != ApiSuccess)
  PlxPrintf("Could not enable external debug mode\n");
```

IOP480CpuExtDebugModeDisable

Syntax:

RETURN_CODE IOP480CpuExtDebugModeDisable(void);

Description:

This function disables the external debug mode.
Note: *External debug mode requires the use of the JTAG port. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

rc = IOP480CpuExtDebugModeDisable();
if (rc != ApiSuccess)
  PlxPrintf("Could not disable external debug mode\n");
```

IOP480CpuIntDebugModeEnable

Syntax:

RETURN_CODE IOP480CpuIntDebugModeEnable(void);

Description:

This function enables the internal debug mode. This mode allows debug events to generate debug exceptions.
Note: *Only a debugger should use this function. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

rc = IOP480CpuIntDebugModeEnable();
if (rc != ApiSuccess)
  PlxPrintf("Could not enable internal debug mode\n");
```

IOP480CpuIntDebugModeDisable

Syntax:

RETURN_CODE IOP480CpuIntDebugModeDisable(void);

Description:

This function disables the internal debug mode. This mode does not allow debug events to generate debug exceptions.
Note: *Only a debugger should use this function. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

rc = IOP480CpuIntDebugModeDisable();
if (rc != ApiSuccess)
  PlxPrintf("Could not disable internal debug mode\n");
```

IOP480CpuDEOnExceptionEnable

Syntax:

RETURN_CODE IOP480CpuDEOnExceptionEnable(void);

Description:

This function allows debug events to be generated when an exception occurs.
Note: *Only a debugger should use this function. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

rc = IOP480CpuIntDebugModeEnable();
if (rc != ApiSuccess)
{
  PlxPrintf("Could not enable internal debug mode\n");
  return;
}

rc = IOP480CpuExceptionEnable(CPU480EX_DEBUG);
if (rc != ApiSuccess)
{
  PlxPrintf("Could not enable debug exception\n");
  return;
}

rc = IOP480CpuDEOnExceptionEnable();
if (rc != ApiSuccess)
{
  PlxPrintf("Could not enable debug events on exceptions\n");
  return;
}
```

IOP480CpuDEOnExceptionDisable

Syntax:

RETURN_CODE IOP480CpuDEOnExceptionDisable(void);

Description:

This function prevents debug events from being generated when an exception occurs.
Note: *Only a debugger should use this function. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
RETURN_CODE rc;

rc = IOP480CpuDEOnExceptionDisable();
if (rc != ApiSuccess)
{
  PlxPrintf("Could not disable debug events on exceptions\n");
  return;
}
```

IOP480CpuDEOnBranchEnable

Syntax:

RETURN_CODE IOP480CpuDEOnBranchEnable(void);

Description:

This function allows debug events to be generated before a pre-determined branch is about to occur.
Note: *Only a debugger should use this function. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
void (*OldDebugESR)(void);
extern void OtherFunction(void);

void OurDebugESR(void)
{
  PlxPrintf("A debug event occurred\n");
  if ((U32)OldDebugESR != 0xFFFFFFFF)
    (*OldDebugESR)();
}

void main(void)
{
  RETURN_CODE rc;

  OldDebugESR = IOP480CpuExceptionHandler(CPU480EX_DEBUG, OurDebugESR);
  rc = IOP480CpuIntDebugModeEnable();
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not enable internal debug mode\n");
    return;
  }

  rc = IOP480CpuExceptionEnable(CPU480EX_DEBUG);
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not enable debug exceptions\n");
    return;
  }

  rc = IOP480CpuDEOnBranchEnable();
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not enable debug events on branches\n");
    return;
  }

  /* This instruction will generate a debug event on branch */
```

```
  OtherFunction();
  ...

  rc = IOP480CpuDEOnBranchDisable();
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not disable debug events on branches\n");
    return;
  }

  rc = IOP480CpuExceptionDisable(CPU480EX_DEBUG);
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not disable debug exceptions\n");
    return;
  }

  rc = IOP480CpuIntDebugModeDisable();
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not disable internal debug mode\n");
    return;
  }

  IOP480CpuExceptionHandler(CPU480EX_DEBUG, OldDebugESR);
  /* Now normal processing resume */

  ...
}
```

IOP480CpuDEOnBranchDisable

Syntax:

RETURN_CODE IOP480CpuDEOnBranchDisable(void);

Description:

This function prevents debug events from being generated when a determined branch instruction is about to occur.
Note: *Only a debugger should use this function. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
ApiSuccess	The function returned successfully.	IOP 480

Usage:

```
void (*OldDebugESR)(void);
extern void OtherFunction(void);

void OurDebugESR(void)
{
  PlxPrintf("A debug event occurred\n");
  if ((U32)OldDebugESR != 0xFFFFFFFF)
    (*OldDebugESR)();
}

void main(void)
{
  RETURN_CODE rc;

  OldDebugESR = IOP480CpuExceptionHandler(CPU480EX_DEBUG, OurDebugESR);
  rc = IOP480CpuIntDebugModeEnable();
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not enable internal debug mode\n");
    return;
  }

  rc = IOP480CpuExceptionEnable(CPU480EX_DEBUG);
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not enable debug exceptions\n");
    return;
  }

  rc = IOP480CpuDEOnBranchEnable();
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not enable debug events on branches\n");
    return;
  }

  /* This instruction will generate a debug event on branch */
```

```
  OtherFunction();
  ...

  rc = IOP480CpuDEOnBranchDisable();
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not disable debug events on branches\n");
    return;
  }

  rc = IOP480CpuExceptionDisable(CPU480EX_DEBUG);
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not disable debug exceptions\n");
    return;
  }

  rc = IOP480CpuIntDebugModeDisable();
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not disable internal debug mode\n");
    return;
  }

  IOP480CpuExceptionHandler(CPU480EX_DEBUG, OldDebugESR);
  /* Now normal processing resume */

  ...
}
```

IOP480CpuDEOnDACEnable

Syntax:

RETURN_CODE IOP480CpuDEOnDACEnable(IN U32 *address*,
IN DAC_SIZE dacSize,
IN BOOLEAN *debugOnReads*,
IN BOOLEAN *debugOnWrites*);

Description:

This function allows debug events to be generated when a successful data address comparison occurs.

- *address* is the address of the data to be checked;
- *dacSize* is the data address compare size defined in Debug Control Register (DBCR) bits 18:19;
- *debugOnReads* tells the function to trigger the event on read accesses; and,
- *debugOnWrites* tells the function to trigger the event on write accesses.

Note: *Only a debugger should use this function. It is an error to call this function with the last 2 parameters set to FALSE. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Cross Reference:

Referenced Item	Page
DAC_SIZE	3-9

Usage:

```
void (*OldDebugESR)(void);

void OurDebugESR(void)
{
  PlxPrintf("A debug event occurred\n");
  if ((U32)OldDebugESR != 0xFFFFFFFF)
    (*OldDebugESR)();
}

void main(void)
{
  RETURN_CODE rc;
  U32 verifyMe;

  OldDebugESR = IOP480CpuExceptionHandler(CPU480EX_DEBUG, OurDebugESR);
  rc = IOP480CpuIntDebugModeEnable();
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not enable internal debug mode\n");
    return;
  }
```

```
  rc = IOP480CpuExceptionEnable(CPU480EX_DEBUG);
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not enable debug exceptions\n");
    return;
  }

  rc = IOP480CpuDEOnDACEnable(&verifyMe, D1SIgnoreTwoLsbs, FALSE, TRUE);
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not enable debug events on DAC\n");
    return;
  }

  /* This instruction will generate a debug event */
  verifyMe++;
  ...

  rc = IOP480CpuDEOnDACDisable();
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not disable debug events on DAC\n");
    return;
  }

  rc = IOP480CpuExceptionDisable(CPU480EX_DEBUG);
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not disable debug exceptions\n");
    return;
  }

  rc = IOP480CpuIntDebugModeDisable();
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not disable internal debug mode\n");
    return;
  }

  IOP480CpuExceptionHandler(CPU480EX_DEBUG, OldDebugESR);
  /* Now normal processing resume */

  ...
}
```

IOP480CpuDEOnDACDisable

Syntax:

```
RETURN_CODE IOP480CpuDEOnDACDisable(IN BOOLEAN debugOnReads,
                                    IN BOOLEAN debugOnWrites);
```

Description:

This function prevents debug events from being generated when a successful data address comparison occurs.

- *debugOnReads* disables the event on read accesses; and,
- *debugOnWrites* disables the event on write accesses.

Note: *Only a debugger should use this function. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
ApiSuccess	The function returned successfully.	IOP 480

Usage:

```
void (*OldDebugESR)(void);

void OurDebugESR(void)
{
  PlxPrintf("A debug event occurred\n");
  if ((U32)OldDebugESR != 0xFFFFFFFF)
    (*OldDebugESR)();
}

void main(void)
{
  RETURN_CODE rc;
  U32 verifyMe;

  OldDebugESR = IOP480CpuExceptionHandler(CPU480EX_DEBUG, OurDebugESR);
  rc = IOP480CpuIntDebugModeEnable();
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not enable internal debug mode\n");
    return;
  }

  rc = IOP480CpuExceptionEnable(CPU480EX_DEBUG);
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not enable debug exceptions\n");
    return;
  }

  rc = IOP480CpuDEOnDACEnable(&verifyMe, D1SIgnoreTwoLsbs, FALSE, TRUE);
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not enable debug events on DAC\n");
```

```
    return;
  }

  /* This instruction will generate a debug event */
  verifyMe++;
  ...

  rc = IOP480CpuDEOnDACDisable(TRUE, TRUE);
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not disable debug events on DAC\n");
    return;
  }

  rc = IOP480CpuExceptionDisable(CPU480EX_DEBUG);
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not disable debug exceptions\n");
    return;
  }

  rc = IOP480CpuIntDebugModeDisable();
  if (rc != ApiSuccess)
  {
    PlxPrintf("Could not disable internal debug mode\n");
    return;
  }

  IOP480CpuExceptionHandler(CPU480EX_DEBUG, OldDebugESR);
  /* Now normal processing resume */

  ...
}
```

IOP480CpuDEOnIACEnable

Syntax:

RETURN_CODE IOP480CpuDEOnIACEnable(IN U32 *address*);

Description:

This function allows debug events to be generated when a successful instruction address comparison occurs.

- *address* is the address of the instruction.

Note: *Only a debugger should use this function. The address parameter has to be word aligned. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
void (*OldDebugESR)(void);

void OurDebugESR(void)
{
      PlxPrintf("A debug event occurred\n");
      if ((U32)OldDebugESR != 0xFFFFFFFF)
      (*OldDebugESR)();
}

void main(void)
{
  RETURN_CODE rc;

      OldDebugESR = IOP480CpuExceptionHandler(CPU480EX_DEBUG, OurDebugESR);
      IOP480CpuIntDebugModeEnable();
      IOP480CpuExceptionEnable(CPU480EX_DEBUG);
      IOP480CpuDEOnIACEnable((ULONG)OtherFunction);

      ...
      /* This instruction will generate a debug event */
      OtherFunction();
      ...

      IOP480CpuDEOnIACDisable();
      IOP480CpuExceptionDisable(CPU480EX_DEBUG);
      IOP480CpuIntDebugModeDisable();
      IOP480CpuExceptionHandler(CPU480EX_DEBUG, OldDebugESR);
      /* Now normal processing resume */

      ...
}
```

IOP480CpuDEOnIACDisable

Syntax:

RETURN_CODE IOP480CpuDEOnIACDisable(void);

Description:

This function prevents debug events from being generated on a successful instruction address comparison.
Note: *Only a debugger should use this function.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
void (*OldDebugESR)(void);

void OurDebugESR(void)
{
      PlxPrintf("A debug event occurred\n");
      if ((U32)OldDebugESR != 0xFFFFFFFF)
      (*OldDebugESR)();
}

void main(void)
{
  RETURN_CODE rc;
      OldDebugESR = IOP480CpuExceptionHandler(CPU480EX_DEBUG, OurDebugESR);
      IOP480CpuIntDebugModeEnable();
      IOP480CpuExceptionEnable(CPU480EX_DEBUG);
      IOP480CpuDEOnIACEnable((ULONG)OtherFunction);

      ...
      /* This instruction will generate a debug event */
      OtherFunction();
      ...

      IOP480CpuDEOnIACDisable();
      IOP480CpuExceptionDisable(CPU480EX_DEBUG);
      IOP480CpuIntDebugModeDisable();
      IOP480CpuExceptionHandler(CPU480EX_DEBUG, OldDebugESR);
      /* Now normal processing resume */

      ...
}
```

IOP480CpuDEOnInstrEnable

Syntax:

RETURN_CODE IOP480CpuDEOnInstrEnable(void);

Description:

This function allows debug events to be generated when an instruction is completed.
Note: *Only a debugger should use this function. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
void (*OldDebugESR)(void);

void OurDebugESR(void)
{
      PlxPrintf("A debug event occurred\n");
      if ((U32)OldDebugESR != 0xFFFFFFFF)
      (*OldDebugESR)();
}

void main(void)
{
  RETURN_CODE rc;
      OldDebugESR = IOP480CpuExceptionHandler(CPU480EX_DEBUG, OurDebugESR);
      IOP480CpuIntDebugModeEnable();
      IOP480CpuExceptionEnable(CPU480EX_DEBUG);
      IOP480CpuDEOnInstrEnable();

      /* All the following instructions will generate a debug event */
      ...

      IOP480CpuDEOnInstrDisable();
      IOP480CpuExceptionDisable(CPU480EX_DEBUG);
      IOP480CpuIntDebugModeDisable();
      IOP480CpuExceptionHandler(CPU480EX_DEBUG, OldDebugESR);
      /* Now normal processing resume */

      ...
}
```

IOP480CpuDEOnInstrDisable

Syntax:

RETURN_CODE IOP480CpuDEOnInstrDisable(void);

Description:

This function prevents debug events from being generated when an instruction is completed.
Note: *Only a debugger should use this function. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
void (*OldDebugESR)(void);

void OurDebugESR(void)
{
      PlxPrintf("A debug event occurred\n");
      if ((U32)OldDebugESR != 0xFFFFFFFF)
      (*OldDebugESR)();
}

void main(void)
{
  RETURN_CODE rc;
      OldDebugESR = IOP480CpuExceptionHandler(CPU480EX_DEBUG, OurDebugESR);
      IOP480CpuIntDebugModeEnable();
      IOP480CpuExceptionEnable(CPU480EX_DEBUG);
      IOP480CpuDEOnInstructionEnable();

      /* All the following instructions will generate a debug event */
      ...

      IOP480CpuDEOnInstructionDisable();
      IOP480CpuExceptionDisable(CPU480EX_DEBUG);
      IOP480CpuIntDebugModeDisable();
      IOP480CpuExceptionHandler(CPU480EX_DEBUG, OldDebugESR);
      /* Now normal processing resume */

      ...
}
```

IOP480CpuDEOnTrapEnable

Syntax:

RETURN_CODE IOP480CpuDEOnTrapEnable(void);

Description:

This function allows debug events to be generated when a trap instruction is executed.
Note: *Only a debugger should use this function. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
void (*OldDebugESR)(void);

void OurDebugESR(void)
{
      PlxPrintf("A debug event occurred\n");
      if ((U32)OldDebugESR != 0xFFFFFFFF)
      (*OldDebugESR)();
}

void main(void)
{
  RETURN_CODE rc;
      OldDebugESR = IOP480CpuExceptionHandler(CPU480EX_DEBUG, OurDebugESR);
      IOP480CpuIntDebugModeEnable();
      IOP480CpuExceptionEnable(CPU480EX_DEBUG);
      IOP480CpuDEOnTrapEnable();

      /* This instruction will generate a debug event */
      IOP480CpuTrap();
      ...

      IOP480CpuDEOnTrapDisable();
      IOP480CpuExceptionDisable(CPU480EX_DEBUG);
      IOP480CpuIntDebugModeDisable();
      IOP480CpuExceptionHandler(CPU480EX_DEBUG, OldDebugESR);
      /* Now normal processing resume */

      ...
}
```

IOP480CpuDEOnTrapDisable

Syntax:

RETURN_CODE IOP480CpuDEOnTrapDisable(void);

Description:

This function prevents debug events from being generated when a trap instruction is executed.
Note: *Only a debugger should use this function. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
void (*OldDebugESR)(void);

void OurDebugESR(void)
{
      PlxPrintf("A debug event occurred\n");
      if ((U32)OldDebugESR != 0xFFFFFFFF)
      (*OldDebugESR)();
}

void main(void)
{
  RETURN_CODE rc;
      OldDebugESR = IOP480CpuExceptionHandler(CPU480EX_DEBUG, OurDebugESR);
      IOP480CpuIntDebugModeEnable();
      IOP480CpuExceptionEnable(CPU480EX_DEBUG);
      IOP480CpuDEOnTrapEnable();

      /* This instruction will generate a debug event */
      IOP480CpuTrap();
      ...

      IOP480CpuDEOnTrapDisable();
      IOP480CpuExceptionDisable(CPU480EX_DEBUG);
      IOP480CpuIntDebugModeDisable();
      IOP480CpuExceptionHandler(CPU480EX_DEBUG, OldDebugESR);
      /* Now normal processing resume */

      ...
}
```

IOP480CpuTrap

Syntax:

RETURN_CODE IOP480CpuTrap(void);

Description:

Trap instruction. When the trap debug event is enabled, a trap instruction will generate a debug exception. Otherwise, this will generate a program exception.
Note: *Only a debugger should use this instruction.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
void (*OldDebugESR)(void);

void OurDebugESR(void)
{
      PlxPrintf("A debug event occurred\n");
      if ((U32)OldDebugESR != 0xFFFFFFFF)
      (*OldDebugESR)();
}

void main(void)
{
  RETURN_CODE rc;
      OldDebugESR = IOP480CpuExceptionHandler(CPU480EX_DEBUG, OurDebugESR);
      IOP480CpuIntDebugModeEnable();
      IOP480CpuExceptionEnable(CPU480EX_DEBUG);
      IOP480CpuDEOnTrapEnable();

      /* This instruction will generate a debug event */
      IOP480CpuTrap();
      ...

      IOP480CpuDEOnTrapDisable();
      IOP480CpuExceptionDisable(CPU480EX_DEBUG);
      IOP480CpuIntDebugModeDisable();
      IOP480CpuExceptionHandler(CPU480EX_DEBUG, OldDebugESR);
      /* Now normal processing resume */

      ...
}
```

IOP480CpuSetTrap

Syntax:

U32 IOP480CpuSetTrap(IN PU32 *address*);

Description:

This function sets up a trap at a specific address and return the opcode previously there.

- *address* is the address of the trap to place. This parameter must be word aligned.

Note: *Only a debugger should use this function. Before the trap is set, the instruction cache for the address is invalidated and the instruction queue is synchronized.*

Return Value:

On success, this function returns the opcode that was located at the address.
On error, this function returns 0xFFFFFFFF.

Usage:

```
#include “CPU480.h”
void (*OldDebugESR)(void);
extern void OtherFunction(void);
ULONG opcode;

void OurDebugESR(void)
{
      PlxKdPrint((S1DB_PARMS, “A debug event occurred\n”));
      /* Replace the correct opcode */
      *(ULONG *)OtherFunction = opcode;

      if ((ULONG)OldDebugESR != 0xFFFFFFFF)
            (*OldDebugESR)();
}

void main(void)
{
      OldDebugESR = IOP480CpuExceptionHandler(CPU480EX_DEBUG, OurDebugESR);
      IOP480CpuIntDebugModeEnable();
      IOP480CpuExceptionEnable(CPU480EX_DEBUG);
      IOP480CpuDEOnTrapEnable();

      opcode = IOP480CpuSetTrap(OtherFunction);
      /* This instruction will generate a debug event */
      OtherFunction();
      ...

      IOP480CpuDEOnTrapDisable();
      IOP480CpuExceptionDisable(CPU480EX_DEBUG);
      IOP480CpuIntDebugModeDisable();
      IOP480CpuExceptionHandler(CPU480EX_DEBUG, OldDebugESR);
      /* Now normal processing resume */

      ...
}
```

IOP480CpuGetDebugStatus

Syntax:

RETURN_CODE IOP480CpuGetDebugStatus(OUT PDEBUG_STATUS pDebugStatus);

Description:

Return a debug status word.
Note: *This function may be use by a normal privileged program, but is useful only to a debugger. This function can only be called from privileged mode.*

Cross Reference:

Referenced Item	Page
DEBUG_STATUS	3-10

Return Value:

Return Value	Description	IC Support
ApiSuccess	The function returned successfully.	IOP 480
ApiNullParam	The caller provided a NULL pointer in the paramter.	IOP 480

Usage:

```
#include “CPU480.h”
void (*OldDebugESR)(void);
extern void OtherFunction(void);
ULONG opcode;
void OurDebugESR(void)
{
    DEBUG_STATUS status;
    PlxPrintf("Debug events:\n");
    IOP480CpuGetDebugStatus(&status);
    if (status.InstrDE)
        PlxPrintf(, " Instruction completion\n");
    if (status.BranchDE)
        PlxPrintf(, " Branch\n");
    if (status.ExceptionDE)
        PlxPrintf(, " Exception occurred\n");
    if (status.TrapDE)
        PlxPrintf(, " Trap\n");
    if (status.IACDE)
        PlxPrintf(, " Instruction address compare\n");
    if (status.DACReadDE)
        PlxPrintf(, " Data address compare- read\n");
    if (status.DACWriteDE)
        PlxPrintf(, " Data address compare- write\n");
    /* Clear the debug status register by writing a mask to it */
    IOP480CpuRegisterWrite(CPU480_DBSR, (ULONG)status);
    if ((ULONG)OldDebugESR != 0xFFFFFFFF)
        (*OldDebugESR)();
}
```

```
void main(void)
{
    OldDebugESR = IOP480CpuExceptionHandler(CPU480EX_DEBUG, OurDebugESR);
    IOP480CpuIntDebugModeEnable();
    IOP480CpuExceptionEnable(CPU480EX_DEBUG);

    /* Instructions that may create debug events */
    ...

    IOP480CpuExceptionDisable(CPU480EX_DEBUG);
    IOP480CpuIntDebugModeDisable();
    IOP480CpuExceptionHandler(CPU480EX_DEBUG, OldDebugESR);
    /* Now normal processing resume */

    ...

}
```

IOP480CpuGetLastReset

Syntax:

RESET_TYPE IOP480CpuGetLastReset(void);

Description:

This function gets the last type of reset. This can be used by an initialization code to initiate a “safe mode”.
Note: *This function can only be called from privileged mode.*

Return Value:

On success, this function returns the last reset type.
On error, this function returns 0xFFFFFFFF.

Cross Reference:

Referenced Item	Page
RESET_TYPE	3-8

Usage:

```
#include "Cpu480.h"
void main(void)
{
    RESET_TYPE type;
    type = IOP480CpuGetLastReset();
    PlxPrint("The last reset was a ");
    switch(type)
    {
    case NoReset:
        PlxPrint("power-up\n");
        break;
    case CoreReset:
        PlxPrint("CPU reset\n");
        break;
    case ChipReset:
        PlxPrint("chip reset\n");
        break;
    case SystemReset:
        PlxPrint("system reset\n");
        break;
    default:
        PlxPrint("unknown, the API function failed\n");
        break;
    }
}
```

IOP480CpuReset

Syntax:

void IOP480CpuReset(IN RESET_TYPE *reset*);

Description:

Reset the IOP 480.

- *reset* is the kind of reset to be performed.

Note: *The NoReset reset type is, of course, not accepted by this function. This function can only be called from privileged mode.*

Cross Reference:

Referenced Item	Page
RESET_TYPE	3-8

Return Value:

Usage:

```
#include "CPU480.h"
void (*OldPanicFunction)(void);
void MyPanicFunction(void)
{
    PlxPrintf( "PANIC: a fault occurred. \n");
    PlxPrintf( "Resetting the board...\n");
    IOP480CpuReset(SystemReset);
}

void main(void)
{
    OldPanicFunction = IOP480CpuExceptionHandler(CPU480EX_PROGRAM |
                                                 CPU480EX_MACHINE |
                                                 CPU480EX_DSTORAGE |
                                                 CPU480EX_ALIGNMENT,
                                                 MyPanicFunction);
    IOP480CpuExceptionEnable(CPU480EX_MACHINE | CPU480EX_DSTORAGE);

    ...

    Something happens and the board resets itself.
}
```

IOP480CpuDEDisableAll

Syntax:

RETURN_CODE IOP480CpuDEDisableAll(void);

Description:

This function disables all debug events.
Note: *This function should be used only by a debugger. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
#include “CPU480.h”
extern void OtherFunction(void);
void (*OldDebugESR)(void);

void OurDebugESR(void)
{
      PlxKdPrint((S1DB_PARMS, “A debug event occurred\n”));
      if ((ULONG)OldDebugESR != 0xFFFFFFFF)
            (*OldDebugESR)();
}

void main(void)
{
      OldDebugESR = IOP480CpuExceptionHandler(CPU480EX_DEBUG, OurDebugESR);
      IOP480CpuIntDebugModeEnable();
      IOP480CpuExceptionEnable(CPU480EX_DEBUG);
      IOP480CpuDEOnIACEnable((ULONG)OtherFunction);
      IOP480CpuDEOnBranchEnable();
      IOP480CpuDEOnTrapEnable();

      /* Instructions that may generate a debug event */
      ...

      IOP480CpuDEDisableAll();
      IOP480CpuExceptionDisable(CPU480EX_DEBUG);
      IOP480CpuIntDebugModeDisable();
      IOP480CpuExceptionHandler(CPU480EX_DEBUG, OldDebugESR);
      /* Now normal processing resume */

      ...
}
```

IOP480CpuMMUInvalidateAll

Syntax:

RETURN_CODE IOP480CpuMMUInvalidateAll(void);

Description:

This function invalidates all the mapped pages in the Memory Management Unit.

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
#include "CPU480.h"

. . .
IOP480CpuMMUInvalidateAll();
. . .
```

IOP480CpuMMUEnable

Syntax:

RETURN_CODE IOP480CpuMMUEnable(IN BOOLEAN *translateInstruction*,
IN BOOLEAN *translateData*);

Description:

Enable or disable instruction and/or data address translation.

- *translateInstruction* is the boolean value to indicate whether to translate instruction or not;
- *translateData* is the boolean value to indicate whether to translate data or not;

***Note**: This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
#include "CPU480.h"

BOOLEAN     bTranslateInstruction;
BOOLEAN     bTranslateData;

bTranslateInstruction = FALSE;
bTranslateData = TRUE;

IOP480CpuMMUEnable(bTranslateInstruction, bTranslateData);
```

IOP480CpuMMUMap

Syntax:

RETURN_CODE IOP480CpuMMUMap(IN PTLB_ENTRY_DESC *pTLBEntryDesc*,
IN U32 *tlbEntryNumber*);

Description:

Map a region of memory into the memory management unit without validating or invalidating the TLB entry.

- *pTlbEntryDesc* is the TLB entry descriptor;
- *tlbEntryNumber* is the TLB entry number.

Cross Reference:

Referenced Item	Page
PTLB_ENTRY_DESC	3-11

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
#include "CPU480.h"

TLB_ENTRY_DESC    tlbEntryDesc;

memset(&tlbEntryDesc, 0, sizeof(TLB_ENTRY_DESC));
tlbEntryDesc.EffectivePageNum = 0x200000;
tlbEntryDesc.Size = TLB_SIZE_16MB;
tlbEntryDesc.Valid = 1;
tlbEntryDesc.Endian = 0;
tlbEntryDesc.Compression = 0;
tlbEntryDesc.Tid = 2;
tlbEntryDesc.RealPageNum = 0x100000;
tlbEntryDesc.ExecuteEnable = 1;     /* Executable but not writable */
tlbEntryDesc.WriteEnable = 0;
tlbEntryDesc.ZoneSelect = 0;
tlbEntryDesc.WriteThrough = 0;
tlbEntryDesc.CachingInhibited = 0;
tlbEntryDesc.MemoryCoherent = 0;
tlbEntryDesc.Guarded = 0;

if (IOP480CpuMMUMap(&tlbEntryDesc, 1) != ApiSuccess )
    PlxPrintf("\nFailed to map logical address to real address.");
```

IOP480CpuMMUValidate

Syntax:

RETURN_CODE IOP480CpuMMUValidate(IN U32 *tlbEntry*);

Description:

Validate a Translation Lookaside Table(TLB) entry in the memory management unit.

- *tlbEntry* is the TLB entry number to validate.

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
#include "CPU480.h"

U32   tlbEntryNum;

tlbEntryNum = 2;

      .     .     .

      IOP480CpuMMUValidate(tlbEntryNum);

      .     .     .
```

IOP480CpuMMUInvalidate

Syntax:

RETURN_CODE IOP480CpuMMUInvalidate(IN U32 *tlbEntry*);

Description:

Invalidate a TLB entry in the memory management unit.

- *tlbEntry* is the TLB entry number to invalidate.

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
#include "CPU480.h"

U32   tlbEntryNum;

tlbEntryNum = 2;

      .     .     .

      IOP480CpuMMUInvalidate(tlbEntryNum);

      .     .     .
```

IOP480CpuMMUSearch

Syntax:

U32 IOP480CpuMMUSearch(IN U32 *logicalAddress*);

Description:

Search the Memory Management Unit for a valid entry for the address specified.

- *logicalAddress* is the logical address to search for in the TLB entries.

Return Value:

Return the TLB entry number if the *logicalAddress* is found. Otherwise, it will return 0xFFFFFFFF.

Usage:

```
#include “CPU480.h”

U32   tlbEntryNum;
U32   logicalAddress = 0x10000000;

tlbEntryNum = IOP480CpuMMUSearch(logicalAddress);
if ( tlbEntryNum == 0xFFFFFFFF )
      PlxPrintf(“\nFailed to find logical address in the TLB.”);
```

IOP480CpuZoneProtectionRead

Syntax:

U32 IOP480CpuZoneProtectionRead(IN U32 *zone*);

Description:

Read the protection settings of a zone.

- *zone* is the zone to access Zone Protection Register (ZPR);

***Note**: This function can only be called from privileged mode.*

Return Value:

Return the protection settings of a zone as a U32 value. Usage:

```
#include "CPU480.h"

U32   zone, zoneSetting;

zone = 0x1;
zoneSettings = IOP480CpuZoneProtectionRead( zone );
```

IOP480CpuZoneProtectionWrite

Syntax:

RETURN_CODE IOP480CpuZoneProtectionWrite(IN U32 *zone*, IN U32 *zoneSetting*);

Description:

Set up the protection settings of a zone.

- *zone* is the zone to access Zone Protection Register (ZPR);
- *zoneSetting* is the zone settings to write to that zone indexed by the *zone* parameter.

***Note**: This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480
`ApiInvalidData`	The data parameters are invalid	IOP 480

Usage:

```
#include “CPU480.h”

U32   zone = 0x1;
U32   zoneSetting = 0;

If ( IOP480CpuZoneProtectionWrite(zone, zoneSetting) != ApiSuccess )
      PlxPrintf(“\nFailed to write the zone protection register.”);
```

IOP480CpuLockIopBus

Syntax:

void IOP480CpuLockIopBus(IN PU32 iop480ConfigBase);

Description:

This function call will make the local bus arbiter to grant the local bus only to the internal IOP 480 CPU.

Return Value:

Usage:

```
#include “CPU480.h”

IOP480CpuLockIopBus(0x50000000);

/* do something here */

IOP480CpuUnlockIopBus(0x50000000);
```

IOP480CpuUnlockIopBus

Syntax:

void IOP480CpuUnlockIopBus(IN PU32 iop480ConfigBase);

Description:

This function call will make the local bus arbiter to operate normally.

Return Value:

Usage:

```
#include “CPU480.h”

IOP480CpuLockIopBus(0x50000000);

/* do something here */

IOP480CpuUnlockIopBus(0x50000000);
```

IOP480CpuGoWaitState

Syntax:

RETURN_CODE IOP480CpuGoWaitState(void);

Description:

Go to the wait mode. When the IOP 480 enters this mode, it stops normal execution processing until a reset or until an exception occurs.
Note: *This function will not return until an exception occurs. This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	IOP 480

Usage:

```
#include "CPU480.h"
void WaitForInterrupt(void)
{
      IOP480CpuGoWaitState();
}

void main(void)
{
      ...
      IOP480CpuExceptionEnable(EXTERNAL_INTERRUPT);

      ...

      WaitForInterrupt();

      ...

      IOP480CpuExceptionDisable(EXTERNAL_INTERRUPT);
}
```

IOP480CpuGetVersion

Syntax:

RETURN_CODE IOP480CpuGetVersion(PPROCESSOR_VERSION *pProcessVersion);*

Description:

This function returns the version of the IOP 480 CPU chip.

- *pProcessVersion* points to the structure to receive process information.

***Note**: This function can only be called from privileged mode.*

Return Value:

Return Value	Description	IC Support
ApiSuccess	The function returned successfully.	IOP 480
ApiNullParam	The caller provided a NULL pointer in the paramter.	IOP 480

Cross Reference:

Referenced Item	Page
PROCESSOR_VERSION	3-13

Usage:

```
#include "CPU480.h"
void main(void)
{
    PROCESSOR_VERSION version;
    ...

    PlxPrintf("\nProcessor info:\n");
    IOP480CpuGetVersion(&version);
    PlxPrintf("Family: 0x%X\n", version.Version.Family);
    PlxPrintf("Family member: %d\n", version.Version.Member);
    PlxPrintf("CPU revision: %d\n", version.Revision.CPU);
    PlxPrintf("Chip revision: %d\n", version.Revision.Chip);

    ...
}
```

3. IOP 480 CPU Data Structures and Constants

The following is an example of a data structure or data type definition.

Sample Data Structure

```
typedef struct _SAMPLE
{
  U32 someRegister;
  U32 someNumber;
  U32 someSize;
  U32 someBuffer[SOME_BUFFER_SIZE];
}SAMPLE, *PSAMPLE;
```

Affected Register Location

Structure Element	PCI 9080	PCI 9054	IOP 480
SomeRegister	xxx, yy	xxx, yy	xxx, yy

The registers that are affected by changing the values in the structure for each PLX device. All register offsets are from the IOP side unless otherwise stated. “xxx” is the register offset and “yy” is the bits in the register that are affected.

Purpose

The reasons for using this structure.

Members

An explanation of the members contained within the structure. Possible values are given when applicable.

Comments

Extra comments on how and when this structure is used.

Changes

Indicate whether it is necessary to change or modify the data structure from the previous SDK version.

IOP 480 CPU Register Constants

The following list contains all the IOP 480 CPU registers. For the Special Purpose Registers set, please consult the IOP 480 date sheet.

```
#define CPU480_R0           0       /* 32 General Purpose Register */
#define CPU480_R1           1
#define CPU480_R2           2
#define CPU480_R3           3
#define CPU480_R4           4
#define CPU480_R5           5
#define CPU480_R6           6
#define CPU480_R7           7
#define CPU480_R8           8
#define CPU480_R9           9
#define CPU480_R10          10
#define CPU480_R11          11
#define CPU480_R12          12
#define CPU480_R13          13
#define CPU480_R14          14
#define CPU480_R15          15
#define CPU480_R16          16
#define CPU480_R17          17
#define CPU480_R18          18
#define CPU480_R19          19
#define CPU480_R20          20
#define CPU480_R21          21
#define CPU480_R22          22
#define CPU480_R23          23
#define CPU480_R24          24
#define CPU480_R25          25
#define CPU480_R26          26
#define CPU480_R27          27
#define CPU480_R28          28
#define CPU480_R29          29
#define CPU480_R30          30
#define CPU480_R31          31
#define CPU480_MSR          32      /* Machine State Register */
#define CPU480_CR           33      /* Condition Register */
#define CPU480_CDBCR        34      /* Special Purpose Registers */
#define CPU480_CTR          35
#define CPU480_DAC          36
#define CPU480_DBCR         37
#define CPU480_DBSR         38
#define CPU480_DCCR         39
#define CPU480_DCWR         40
#define CPU480_DEAR         41
#define CPU480_ESR          42
#define CPU480_EVPR         43
#define CPU480_IAC          44
#define CPU480_ICCR         45
#define CPU480_ICDBDR       46
#define CPU480_LR           47
#define CPU480_PID          48
#define CPU480_PIT          49
```

```
#define CPU480_PVR          50
#define CPU480_SGR          51
#define CPU480_SKR          52
#define CPU480_SLER         53
#define CPU480_SPRG0        54
#define CPU480_SPRG1        55
#define CPU480_SPRG2        56
#define CPU480_SPRG3        57
#define CPU480_SRR0         58
#define CPU480_SRR1         59
#define CPU480_SRR2         60
#define CPU480_SRR3         61
#define CPU480_TBHI         62
#define CPU480_TBHU         63
#define CPU480_TBLO         64
#define CPU480_TBLU         65
#define CPU480_TCR          66
#define CPU480_TSR          67
#define CPU480_XER          68
#define CPU480_ZPR          69
```

Exception Type Constants

Purpose:
The definition of IOP 480 CPU exception constants is to make it clear to the programmer what an exception might be.

Constants:

CPU480EX_CRITICAL
: This exception occurs when the critical interrupt line goes low. This exception type share the same enable bit as the watchdog exception type.

CPU480EX_MACHINE
: This exception occurs when an external bus error occurs on an instruction fetch and execution of that instruction is subsequently attempted; or when an external bus error occurs while attempting data access.

CPU480EX_DSTORAGE
: The data storage exception occurs on a cache line locking error.

CPU480EX_ISTORAGE
: This exception occurs when instruction translation is enabled and execution is attempted for an instruction whose fetch is not allowed for some reasons.

CPU480EX_EXTERNAL
: This exception occurs when the external interrupt line goes low.

CPU480EX_ALIGNMENT
: This exception occurs when misaligned data accesses are executed. This exception is always enabled and cannot be disabled.

CPU480EX_PROGRAM
: This exception occurs when the IOP 480 CPU tries to execute an illegal instruction or when executing a trap instruction. This exception cannot be disabled.

CPU480EX_SYSTEM
: This exception occurs when executing a system call instruction. This exception cannot be disabled.

CPU480EX_PIT
: This exception occurs when the Programmable Interval Timer goes to 0. At the hardware level, this exception cannot be enabled without enabling the external interrupt exception.

CPU480EX_FIT
: This exception occurs regularly when the Fixed Interval Timer exception is enabled. At the hardware level, this exception cannot be enabled without enabling the external interrupt exception.

CPU480EX_WATCHDOG
: This exception occurs regularly when the watchdog timer exception is enabled. Under some conditions, a reset will be generated instead of an exception. At the hardware level, this exception cannot be enabled without enabling the critical interrupt exception.

CPU480EX_DTLB
: This exception occurs when data translation is enabled and access for a data is not mapped in the Memory Management Unit.

CPU480EX_ITLB
: This exception occurs when instruction translation is enabled and execution of an unmapped instruction is attempted.

CPU480EX_DEBUG
: This exception occurs when a debug event occurs.

```
#define CPU480EX_CRITICAL       0x00000001
#define CPU480EX_MACHINE        0x00000002
#define CPU480EX_DSTORAGE       0x00000004
#define CPU480EX_ISTORAGE       0x00000008
#define CPU480EX_EXTERNAL       0x00000010
#define CPU480EX_ALIGNMENT      0x00000020
#define CPU480EX_PROGRAM        0x00000040
#define CPU480EX_SYSTEM         0x00000080
```

```
#define CPU480EX_PIT            0x00000100
#define CPU480EX_FIT            0x00000200
#define CPU480EX_WATCHDOG       0x00000400
#define CPU480EX_DTLB           0x00000800
#define CPU480EX_ITLB           0x00001000
#define CPU480EX_DEBUG          0x00002000
```

FIT_TIMER_PERIOD Enum Data Type

```
typedef enum _FIT_TIMER_PERIOD
{
    Fit2E9Clocks,
    Fit2E13Clocks,
    Fit2E17Clocks,
    Fit2E21Clocks
} FIT_TIMER_PERIOD;
```

Purpose

Enumerated type for choosing one of the Fixed Interval Timer periods.

Members

`Period2E9Clocks`
: The period is 2^9 clocks.

`Period2E13Clocks`
: The period is 2^{13} clocks.

`Period2E17Clocks`
: The period is 2^{17} clocks.

`Period2E21Clocks`
: The period is 2^{21} clocks.

WDG_TIMER_PERIOD Enum Data Type

```
typedef enum _WDG_TIMER_PERIOD
{
    Wgt2E17Clocks,
    Wgt2E21Clocks,
    Wgt2E25Clocks,
    Wgt2E29Clocks
}WDG_TIMER_PERIOD;
```

Purpose

Enumerated type for choosing the Watchdog Timer period.

Members

Period2E17Clocks
: The period is 2^{17} clocks.

Period2E21Clocks
: The period is 2^{21} clocks.

Period2E25Clocks
: The period is 2^{25} clocks.

Period2E29Clocks
: The period is 2^{29} clocks.

RESET_TYPE Enum Data Type

```
typedef enum _RESET_TYPE
{
    NoReset,
    CoreReset,
    ChipReset,
    SystemReset
}RESET_TYPE;
```

Purpose

Enumerated type for the 480 resets.

Members

NoReset,
There would be no reset forced by the watchdog.

CoreReset,
A CPU reset would be forced by the watchdog.

ChipReset,
A chip reset would be forced by the watchdog.

SystemReset
A system reset would be forced by the watchdog.

DAC_SIZE Enum Data Type

```
typedef enum _DAC_SIZE
{
    D1SCompareAllBits,
    D1SIgnoreOneLsb,
    D1SIgnoreTwoLsbs,
    D1SIgnoreFourLsbs
} DAC_SIZE;
```

Purpose

Enumerated type for the data address compare size as defined in Debug Control Register (DBCR) bits 18:19

Members

D1SCompareAllBits,
: Exact address compare;

D1SIgnoreOneLsbt,
: Byte within halfword address compare;

D1SIgnoreTwoLsbs,
: Byte within word address compare;

D1SIgnoreFourLsbst
: Quadword address compare.

DEBUG_STATUS Data Structure

```
typedef struct _DEBUG_STATUS
{
    U32 InstrDE                :1;
    U32 BranchDE               :1;
    U32 ExceptionDE            :1;
    U32 TrapDE                 :1;
    U32 UnconditionalDE        :1;
    U32 IACDE                  :1;
    U32 Reserved1              :1;
    U32 DACReadDE              :1;
    U32 DACWriteDE             :1;
    U32 Reserved2              :2;
    U32 ImpreciseDE            :1;
    U32 Reserved3              :10;
    U32 MostRecentReset        :2;
    U32 Reserved4              :8;
} DEBUG_STATUS;
```

Purpose

Have a structure where the debug status are easily available.

Members

InstrDE
: This bit is set when an instruction completion debug event occurred.

BranchDE
: This bit is set when a determined branch is about to be taken.

ExceptionDE
: This bit is set when an exception debug event occurred.

TrapDE
: This bit is set when a trap debug event occurred.

UnconditionalDE
: This bit is set when an unconditional debug event occurred. This debug event is useful only when using the JTAG port (external debug mode).

IACDE
: This bit is set when an instruction address compare debug event occurred.

DACReadDE
: This bit is set when the data read address compare debug event occurred.

DACWriteDE
: This bit is set when the data write address compare debug event occurred.

ImpreciseDE
: This bit is set when an imprecise debug event occurred.

MostRecentReset
: These bits indicate the type of the most reset that occurred. These bits can be casted to the RESET_TYPE type. See IOP480CpuGetLastReset or IOP480CpuReset for more information.

TLB_ENTRY_DESC Data Structure

```
typedef struct _TLB_ENTRY_DESC
{
    U32 EffectivePageNum        :22;
    U32 Size                    :3;
    U32 Valid                   :1;
    U32 Endian                  :1;
    U32 Compression             :1;
    U32 NotUsed1                :4;
    U32 RealPageNum             :22;
    U32 ExecuteEnable           :1;
    U32 WriteEnable             :1;
    U32 ZoneSelect              :4;
    U32 WriteThrough            :1;
    U32 CachingInhibited        :1;
    U32 MemoryCoherent          :1;
    U32 Guarded                 :1;
    U32 NotUsed2                :24;
    U32 Tid                     :8;
} TLB_ENTRY_DESC, * PTLB_ENTRY_DESC;
```

Purpose

Allow easier interpretation of the TLB entries.

Members

EffectivePageNum
Effective Page Number of one TLB;

Size
Page size, varying from 1KB to 16MB;

Valid
Indicate whether a TLB entry is valid or not;

Endian
When set, indicates that data in the associated page is stored in true Little Endian format;

Compression
When set, indicates that data in the associated page is stored in compressed mode;

NotUsed1
Not used

RealPageNum
Real Page Number;

ExecuteEnable
When set, allows instruction execution at addresses within a page;

WriteEnable
When set, permits store operations to addresses in a page;

ZoneSelect
Selects one of 16 zone fields from the Zone Protection Register (ZPR);

WriteThrough
Whet set, stores are specified as write-through.

CachingInhibited
When set, a memory access is completed by referencing the location in main memory, bypassing the cache arrays.

MemoryCoherent
For multiprocessing support, no effect in IOP480 CPU;

Guarded

When set, indicates that the hardware cannot speculatively access the location for pre-fetching or out-of-order load access.

NotUsed2

Not used

Tid

Translation ID

PROCESSOR_VERSION Data Structure

```
typedef struct _PROCESSOR_VERSION
{
  U32 ProcessorFamily:12;
  U32 ProcessorCoreFunction:4;
  U32 ProcessorCoreRevision:5;
  U32 ASICFunction:7;
  U32 ASICRevision:4;
} PROCESSOR_VERSION;
```

Purpose

Allow easier interpretation of the processor version register.

Members

ProcessorFamily
: PowerPC family. 0x401 for the IOP 480 CPU.

ProcessorCoreFunction
: Processor Core implementation.

ProcessorCoreRevision
: Revision of the processor Core.

ASICFunction
: ASIC Function.

ASICRevision
: ASIC Revision.

This page is intentionally left blank.