developer

Grzegorz Krashan <krashan@matay.pl>

COLLABORATORS			
	TITLE : developer		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Grzegorz Krashan <krashan@matay.pl>	January 23, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

developer

1.1 Contents

P R O M E T H E U S

Information for programmers

Version 1.48 (24 May 2001)

Contents:

1.2 Introduction to PCI

Basic information about PCI

This chapter will provide you with necessary programming information about PCI standard. PCI (Peripherial Component Interface) is a universal 32-bit bus designed to connect the CPU and peripherals of a computer system. The popularity of PCI interface in PC systems resulted in a wide variety of cheap PCI cards available in the market. Prometheus is one of solutions that allow to use such cards with Amiga.

The PCI bus has two 32-bit memory addressing spaces - an input/output space and a memory space. Together, it gives 8 GB of continuous address space. The names are conventional, usually cards put in the I/O address area own control registers while in the memory space they put memory buffers (like video memory of a graphics card). The registers often are duplicated in both spaces. The I/O space area can be from 4 bytes to 4 GB long, the memory area can be from 16 bytes to 4 GB long. Amiga does not allow such large spaces, so Prometheus provides only a fragment of that space, 511 MB of the memory area and 960 kB of the I/O area to be exact (see memory map).

Every PCI card needs configuration before it is available for use. That configuration allocates dynamic addresses to a PCI card, which can demand from the system allocation of maximum six address areas in both address spaces. In a Prometheus equipped system, the prometheus.library does that work automatically while the system is booting. The drivers of each card can locate and read their base addresses using functions implemented in that library.

1.3 Memory address map

Division of the Prometheus address space

Informations provided below are not necessary for programmers working with AmigaOS. They should use prometheus.library functions instead. Knowledge of the Prometheus memory map can however be necessary for those who will write drivers for other systems, like for example Linux or NetBSD.

Prometheus takes 512 MB of the Amiga address space. The card address depends on what other Zorro III cards are installed in the system. In case Prometheus is the only Z III card, it is allocated by the system in the $40000000 - $5FFFFFFF address range. However, do not assume this address is fixed and always read it from the dynamic tables created by the operating system.

As you have already read, PCI has three address spaces that must fit together into one address space of the Amiga system. Therefore, different Prometheus address space fragments are assigned to specific address spaces of the PCI bus.

Here is the memory map, the addresses given are shifts to the base address of the card:

$00000000 - $000EFFFF Input/output PCI space (960 kB)

$000F0000 - $000F00FF Configuration space of the slot 0 (256 B)

$000F0100 - $000F1FFF Reserved

$000F2000 - $000F20FF Configuration space of the slot 1 (256 B)

$000F2100 - $000F3FFF Reserved

$000F4000 - $000F40FF Configuration space of the slot 2 (256 B)

$000F4100 - $000F5FFF Reserved

$000F6000 - $000F60FF Configuration space of the slot 3 (256 B)

$000F6100 - $000FFFFF Reserved

$00100000 - $1FFFFFFF PCI memory area (511 MB)

While allocating the address spaces to the PCI cards, please remember about the address space requirements of the card. Let me remind you once more that in AmigaOS the work is done by the prometheus.library.

1.4 Byte ordering problem

Byte ordering problem

While programming PCI cards, one can face the byte ordering problem in 16 and 32-bit words. The majority of CPU designs, M68k and PPC processors included, use a convention according to which bytes are put in a word in a more-less importance order:

0123

bits 31-24bits 23-16bits 15-8bits 7-0

Unfortunately, in x86 Intel compatibles, bytes are put in memory in the reversed order:

0123

bits 7-0 bits 15-8bits 23-16bits 31-24

As majority of PCI cards is designed to work with PC systems, such a byte order is the most probable to appear. To make things easier, Prometheus is equipped with a hardware byte swapping circuit that swaps bytes on-the-fly, without time delays. Therefore, you can program PCI cards in the same way they are programmed in PC systems. The circuit works in both directions, so if you want to write, for example, $DEADBACA in the 32-bit card register, the Prometheus follows the Intel convention and in the following register bytes $CA, $BA, $AD, $DE will be put. However, when you read the register, you get $DEADBACA again.

Authors of drivers for systems other than AmigaOS will face one exception from the rule. The configuration registers of PCI cards compatible with PCI 2.1 specification have bytes ordered according to the Motorola convention. As the swapping circuit

in Prometheus cannot be switched off, while accessing the configuration area the driver should neutralise the conversion. It can be done by such assembler sequence:

ROL.W #8,d0

SWAP d0

ROL.W #8,d0 You will find appropriate swapl() and swapw() macros for GCC compiler in the SDK includes. In the AmigaOS, the card configuration is done by the prometheus.library, so the problem does not exist.

1.5 Prometheus operation basics

Prometheus operation basics

Prometheus is a transparent Zorro III to PCI bridge. Each Zorro III transaction is translated to the appropriate PCI transaction. Transparency means that no special data read/write functions for PCI cards are needed. The cards can be treated as CPU address areas of the Amiga system. However, please pay attention to the CPU cache memory. AutoConfig procedures of the AmigaOS automatically disable the CPU cache for the address spaces taken by Prometheus. It is possible for MMU equipped CPUs, so problems appear when we deal with a 68EC030 CPU which does not have a MMU unit. In such cases, the cache memory can be flushed by the CacheClearU() or better CacheClearE() function that can flush only data cache.

1.6 prometheus.library and its functions

prometheus.library and its functions

prometheus.library is added to the system by AutoConfig procedures while booting. To be exact, it is done when the BindDrivers command is executed in the startup-sequence. The library detects PCI cards put in the Prometheus slots and configures them. Then it stays in the system to allow seeking for available PCI cards and provides information about them. If you want to check if a given PCI card is installed, use the Prm_FindBoardTagList() function. The Prm_GetBoardAttrsTagList() function can be used to give information about the found card (addresses and allocated space lengths included). Both functions are described in the library autodoc in details. An example program that uses this prometheus.library functions is PrmScan which displays information about all PCI cards found in the system. Its source code is included in this SDK.

1.7 Slow PCI cards problems

Slow PCI cards problems

The PCI specification recommends each card to start the write (or read) operation not later than 8 clock cycles after the PCI cycle start. Unfortunately, some cards in some cases are not able to do that. Such situation occurs, for example, while reading some graphic cards ROM memory. In such case, the card sends a "Retry" reply and asks the transaction initiator to repeat the cycle. However, it is not always possible during one Zorro III bus cycle. The Amiga hardware design limits the Zorro III bus cycle time to about 1 microsecond. If the cycle cannot end during that time, it is interrupted and a bus error is generated. To prevent the system hang-up, Prometheus does not repeat the transaction when a "Retry" reply is sent and returns the $FFFFFFFF value in case of the read attempt. In such case, a read operation should be repeated. Such a solution is exemplified in the source code of the RomDump program.

NOTE:

The only example known so far of the case described above is the ROM memory read operation of the Voodoo3 graphics card.