Application of bus emulation techniques to the design of a PCI/MC68000 bridge

J.M. Rodríguez Corral[a], A. Civit Balcells[b], G. Jiménez Moreno[b], A. Morgado Estévez[a], A. Linares Barranco[b]

[a] Escuela Superior de Ingeniería, Universidad de Cádiz, C/Chile 1, 11003 Cádiz, Spain

[b] Escuela Técnica Superior de Ingeniería Informática, Universidad de Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain

Abstract

Bridges easy the interconnection and communication of devices that operate using different buses. In fact, we can see a computer as a hierarchy of buses to which devices are connected. In this paper we design a PCI/MC68000 bridge in order to improve communications between a Personal Computer and a MC68000 based system. The previous interface between both devices was based on the old 16-bit ISA bus, which represented a bottleneck in their communication. However, the methodology described here is generic and can be applied to the design of PCI bridges to other buses. We finish this work with an analysis of the bridge performance improvement which can also be easily adapted to other situations. As an example our interface is used in an interesting situation, i.e., updating the obsolete control unit of a highly valuable system (an industrial robot).

Keywords: Bridge; Bus emulation; Direct memory access; Embedded system; PCI

1. Introduction

In a previous work, using bus emulation techniques [1,2], we designed an ISA bus interface to control a Hitachi A4010S scara robot, whose control unit was based in a MC68000 processor. In that case, we were able to write new trajectory generation programs and control algorithms running in a 486 PC. Now, we want to update our interface to use it in a PCI bus [3,4] based computer thus increasing the communication bandwidth between the PC and the MC68000 system. Furthermore, as the i486 processor, Pentium II [5] and latter processors are also suitable to emulate the MC68000 bus and substitute this processor in industrial applications [6].

Bus emulation [1,2], as is considered in this paper, can defined as the set of techniques that allows the communication of two systems with different buses. Using these techniques, a machine (substitute system) with a native bus also implements a bus that is characteristic of another system (target system). In our original implementation [2] an ISA interface implements the MC68000 bus using a state machine (one of the bus emulation techniques), which emulates the control bus signals of the target system and is considered as a peripheral by the substitute system. In fact, the bus emulator device is seen by the substitute system as a set of I/O registers which must be programmed with the access address and the data to transfer to the target system (for a write access). Finally, the user must write the access direction (read or write) bit into the control register thus starting the emulation of the target system bus cycle. In a read access, after the emulation of the bus cycle the substitute system must read the data from the interface data register. An interrupt register also exists which allows to mask the interrupt requests from the target system and indicates the current interrupt priority level. This is used to select the corresponding service routine as, in our example, the target system devices request MC68000 autovectored interrupts [7].

2. Bus emulation by input/output modules with direct memory access

In the present paper we want to improve our previous approach by applying direct memory access (DMA) controller design techniques [8], in order to design emulator interfaces as I/O modules which are able to transfer data

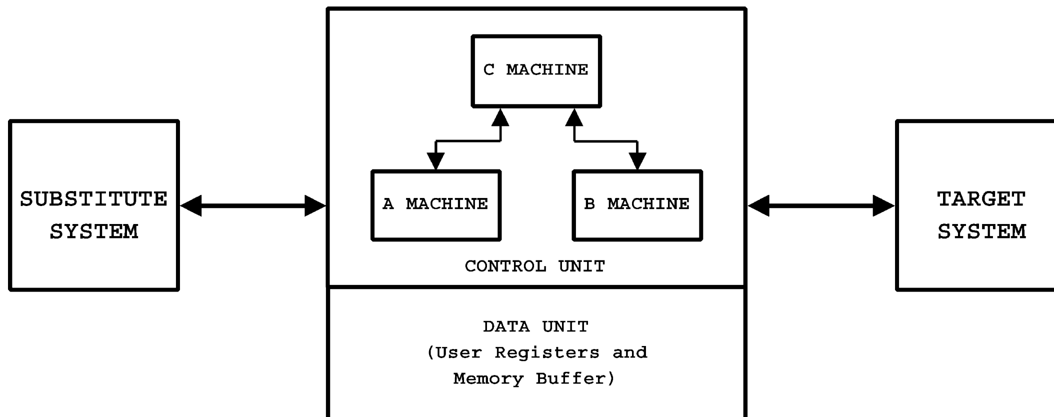

Fig. 1. Bus emulator interface diagram.

blocks between the substitute and the target system. Thus, we will release the processor from the responsibility of programming the interface on every access to the target system. In this way we will limit its task to programming the interface only at the start of the transfer of the whole data block (i.e. it can be dedicated to execute other tasks instead of wasting time when a new word of data must be transferred). When the data block transfer finishes the interface will request an interrupt to the substitute system processor.

Interfaces designed in this way consist of a data unit and a control unit (Fig. 1). The control unit consists of three state machines. The first one (A machine) performs burst transfers between the substitute system and the interface data buffer. The second state machine (B machine) performs bus emulation cycles to transfer data between the target system and the interface. Finally, a third state machine (C machine) coordinates operation of the previous ones. In our PCI/MC68000 bridge design we have integrated A and C into a single state machine in order to avoid delays due to synchronization problems between them. State machine B is just the one designed in Refs. [1,2] for emulating the MC68000 control bus and thus, we are reusing an important part of the old ISA-MC68000 interface design. This is a clear advantage of our modular design approach.

The data unit mainly consists of the user register set, which is used by the substitute system to program the interface. These registers are accessed by a decoder circuit whose design depends directly on the substitute system bus characteristics. The data unit incorporates a data buffer, which replaces the data register in the old ISA-MC68000 interface [1,2] and temporally stores the data block during its transfer between substitute and target system. The data buffer word size must match the substitute system data bus width in order to accelerate the block transfer rate between this system and the bridge.

3. PCI and MC68000 buses

In this section we will examine the PCI and the MC68000 buses by considering only those signals that are usually required to implement processor substitution in embedded controllers. The most common case in industrial MC68000 systems are boards with reduced address spaces (usually 64 Kb or less) and selfvectored interrupts. The PCI (Peripheral Component Interconnect) bus Version 1.0 was designed by Intel Corporation, but later versions have been defined by a consortium known as PCI Special Interest Group (PCI SIG). PCI bus [3,4] can operate at a maximum frequency of 66 MHz. and has a 64 bit bus extension. It uses burst mode for reads and writes reaching a 132 Mb/s peak transfer rate at 33 MHz. with a 32-bit bus width.

Devices that require a fast access or fast transfer rates to the system memory can be connected to the PCI bus. Burst lengths are negotiated between master and slave devices and they are not limited a priori. The main features of the PCI bus are processor independence, support for a maximum of 256 PCI functional devices on the same bus, concurrent bus operation, master support, hidden bus arbitration, transaction integrity checking and autoconfiguration. In our design, from all PCI bus signals, we will use only those that are necessary for our bridge. These are:

1. Address/Data bus (AD0-AD31), Command bus (C/#BE0–C/#BE3) and PAR signal.
2. Interface Control signals: #FRAME, #TRDY, #IRDY, #STOP, #DEVSEL and IDSEL.
3. Arbitration signals: #REQ and #GNT.
4. System signals: CLK and #RST.
5. Interrupt Request signal: #INTA.

A PCI transaction starts with a one clock address phase, where the initiator device addresses the target device (using the address/data bus) and indicates the transaction type (using the command bus). Then one or several data phases occur. In each data phase #BE0–#BE3 enable signals indicate the number of bytes to transfer between the initiator

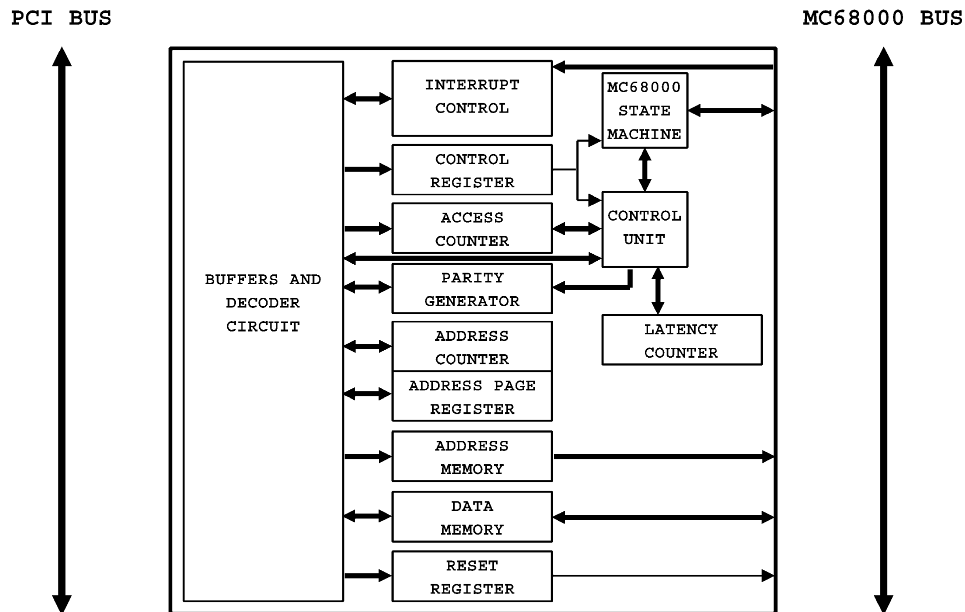

Fig. 2. PCI/MC68000 bridge block diagram.

and the target device. In our design the substitute system uses single data phase configuration read and write transactions to configure the PCI/MC68000 bridge and single data phase I/O read and write transactions to access its user registers. The bridge performs multiple data phase memory read and write transactions in order to transfer data blocks between its data buffer and the substitute system memory.

As already mentioned for the MC68000 bus [7], only those aspects usually found in embedded controllers will be emulated. These are:

1. Asynchronous bus cycle: used to access memory, MC68000 peripherals and fast custom I/O devices.
2. Synchronous bus cycle: used by M68XX peripherals and some synchronous custom I/O.
3. Self-vectored interrupts: used by M68XX and custom I/O.

Provided that the target system address space size is limited to 64 Kb, only the address lines A1–A15 must be considered (A0 line exists only for software for M68000 based systems). The data bus size will be 16 bits (D0–D15) in order to allow 16 bit transfers.

4. PCI/MC68000 bridge design

We have chosen the most popular PCI configuration. The bus operates at 33 MHz. frequency and with a 32 bit-address/data width. The analysis we undertake is still valid, and the performance can be further improved if we were to design using a 66 MHz. and 64-bit address/data PCI bus. Anyhow the improvement would clearly be limited by the slow nature of the proposed target.

Next, we will present the PCI/MC68000 bridge I/O map, in which positions correspond to the different user registers. As in our original design this map also includes a register for interrupt control. ‘R’ denotes a read accessible register and ‘W’ denotes a write accessible register.

P0	ADDRESS MEMORY (EVEN)	A[0–7]	(W)
P1	ADDRESS MEMORY (EVEN)	A[8–15]	(W)
P2	ADDRESS MEMORY (ODD)	A[0–7]	(W)
P3	ADDRESS MEMORY (ODD)	A[8–15]	(W)
P4	ADDRESS COUNTER	AD[0–7]	(W)
P5	ADDRESS COUNTER	AD[8–15]	(W)
P6	PAGE REGISTER	AD[16–23]	(W)
P7	CLEARS PCI/MC68000 BRIDGE IRQ		(R)
	ENABLES RESET SIGNAL		(W)
P8	ACCESS COUNTER		(W)
P9	ACCESS COUNTER		(W)
P10	PCI/MC68000 BRIDGE IRQ STATUS		(R)
	CONTROL REGISTER		(W)
P11	INTERRUPT CONTROL REGISTER		(R,W)

Fig. 2 shows a block diagram of the bridge, which provides a general view of its design. For an easy understanding of these diagram we have simplified it and shown only those connections we consider most important. As stated in Section 2, the bridge control unit is subdivided into a main module (control unit state machine), which coordinates the bridge general operation and controls data block transfers between the substitute system and the bridge intermediate data memory (the functions of the original C and A machines), and a secondary module (B machine or MC68000 state machine), which emulates MC68000 bus

Table 1
State machine signals

Signal	Meaning
#ITAC	Turn-around cycle (initiator)
#ISTS	S/T/S signal disable cycle (initiator)
#TSTS	S/T/S signal disable cycle (target)
#IADDRPH	Address phase (initiator)
#IDATAPH	Data phase (initiator)
#TDATAPH	Data phase (target)
#IOR	I/O address space read
#IOW	I/O address space write
#CNFR	Configuration address space read
#CNFW	Configuration address space write
#BADDR	PCI/MC68000 bridge base address
#T0	Zero type configuration access
#F0	Configuration access over device 0 function
#RSTFIFO	Resets address and data FIFO memories
#LDCONT	Loads access counter
#ZCONT	Access counter reaches zero
IOW4S	Starts a MC68000 cycle
#AS	MC68000 address strobe
AD0	MC68000 cycle finished
#ADDEN	MC68000 address enable
#LDLTCONT	Loads PCI latency counter
#DLTCONT	Decrements PCI latency counter
#ZLTCONT	PCI latency counter reaches zero
#ULTRANSF	Last PCI 32-bit data phase
READ	0 ≡ PC ⟶ MC68000 system
	1 ≡ PC ⟵ MC68000 system

cycles in order to transfer each 16-bit word between the target system and the bridge data buffer.

4.1. Decoder circuit

The decoder circuit allows read and write access to the bridge configuration and user registers. In our example the control software uses the former to program the PCI/MC68000 bridge, and PCI configuration startup firmware uses the latter to set up the bridge for its operation. Table 1 shows the meaning of the decoder and the control unit state machine signals. Table 2 describes the correspondence between the C/#BE bus signals provided by an initiator during the address phase of a transaction and its type.

The state machine that controls the access to the bridge registers is shown in Fig. 3. Its complexity is due to the fact that it must control both configuration and I/O accesses. In the initial state (state A), the bridge data unit captures the value in the AD and C/#BE buses on every transaction address phase. In a configuration space access, the physical PCI device is selected by its corresponding IDSEL signal during the transaction address phase. However, the configuration access must be type zero and act over device function zero [3]. Finally, only if the transaction is a configuration read (#CNFR) or a configuration write (#CNFW) (Table 2) then an access to the corresponding configuration register will occur. If the access is to the I/O address space (which is 16 bytes long), then the data unit address comparer enables #BADDR (base address) when the 28 most significant registered address bits match the 28 most significant bits of the zero base address configuration register (Fig. 7). Finally, only if the transaction is an I/O read (#IOR) or an I/O write (#IOW) (Table 2), an access to the corresponding user register will occur.

Table 2
Transaction types

Type	#C/BE3	#C/BE2	#C/BE1	#C/BE0
#IOR	0	0	1	0
#IOW	0	0	1	1
#CNFR	1	0	1	0
#CNFW	1	0	1	1

Table 3
Transaction termination types

Termination type	#TRDY	#STOP	#DEVSEL
Normal	0	1	0
Disconnect A or B	0	0	0
Disconnect C	1	0	0
Retry	1	0	0
Target Abort	1	0	1

Configuration and I/O read accesses takes longer due to the turnaround cycle, which is translated into an additional state (state B). In order to simplify the decoder design, it only allows single data phase transactions and thus, it finishes all multiple data phase transactions initiated by the Host/PCI bridge issuing a disconnect C (state C) in the second data phase (Table 3). Anyway, many PCI I/O targets are not designed to support multiple data phase transactions and, on the other hand, almost all configuration transactions have a single data phase [3], although the PCI bus specification allows multiple data phase configuration transactions [4].

Logical expressions for signals that control I/O and configuration registers are described concisely by Eqs. (1)–(4), where f and g are combinatory functions that will have specific values when referring to an individual configuration or user register. QAi stands for AD bus i-order bit value registered in the current transaction address phase and #BEi is C/#BE bus i-order bit value in the current transaction data phase. Finally, as a read access is a non-destructive operation, configuration and user register reads always provide a full 32-bit datum regardless of the C/#BE bus value in current transaction data phase.

$$\#IORx^{*} = \#IOR + \#BDEVSELo + f(QA[2,3]) \quad (1)$$

$$\#IOWx = \#IOW + \#BTRDYo + f(QA[2,3]) + g(\#BE[0\ldots3]) \quad (2)$$

$$\#CNFRx^{*} = \#CNFR + \#BDEVSELo + f(QA[2\ldots7]) \quad (3)$$

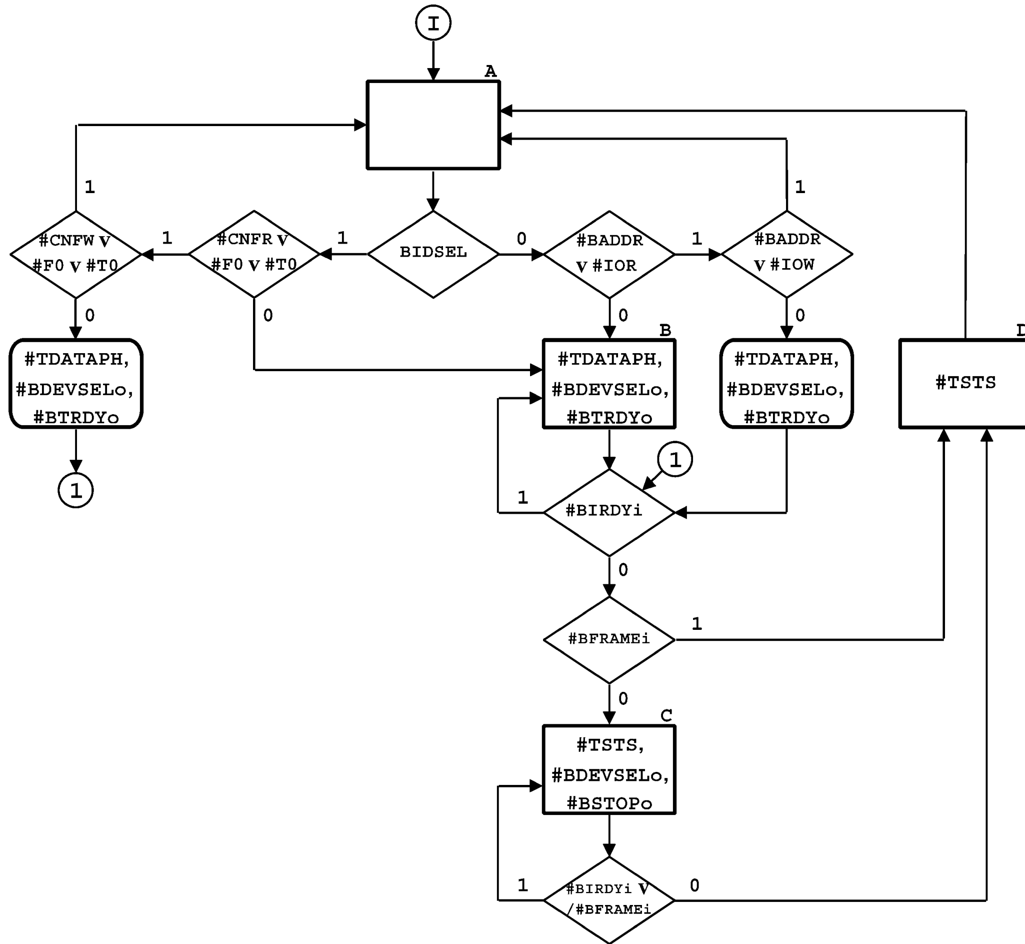

Fig. 3. Decoder circuit state machine.

$$\#CNFWx = \#CNFW + \#BTRDYo + f(QA[2\ldots7]) + g(\#BE[0\ldots3]) \tag{4}$$

4.2. Data unit

Compared to our original ISA-MC68000 interface, user registers experiment important modifications (Fig. 2): First, the address register is substituted by an address buffer, which stores a set of target system address and, in this way, allows accessing the target system positions in any order. We must also include an access counter in which the substitute system stores the number of read or write accesses to the target system and is decremented on every access. We also need an address counter to store the initial address of the accesses to the substitute system memory. This counter will be incremented every time a double word of the data block has been transferred between this system and the interface data buffer.

The PCI bus #RST signal resets the address and data FIFO memories during the system initialisation. The substitute system stores in the first FIFO the set of MC68000 system access addresses by writing two of them in each 32bit access, as our example system (Hitachi A4010S) address space is limited to 64 K (like many other MC68000 based embedded systems) and, thus, the 8 most significant bits (MSB) of a MC68000 24-bit address are always zero [1,2]. The data memory replaces the data register and is used by the bridge to temporally store the data block during its transfer between the PC and the MC68000 system. Although the MC68000 state machine performs 16-bit accesses to the target system, the substitute system makes 32-bit accesses to the bridge data buffer in order to use the whole PCI bus bandwidth.

The address counter register stores the 16 least significant bits (LSB) of a PC RAM address. Page register stores address bits 16–23, that will be also sent to the corresponding PCI address bus lines during the transaction address phase. Finally, the 8 highest address bus lines are always low, as we locate the data block zone in the PC low RAM area (the first 16 Mb). The #RESET signal from the reset register initialises control and interrupt registers as well as the MC68000 state machine and all the target system devices. The reset register output goes low (active) after a PC power-on or reset.

The MC68000 system access counter is initialized by the

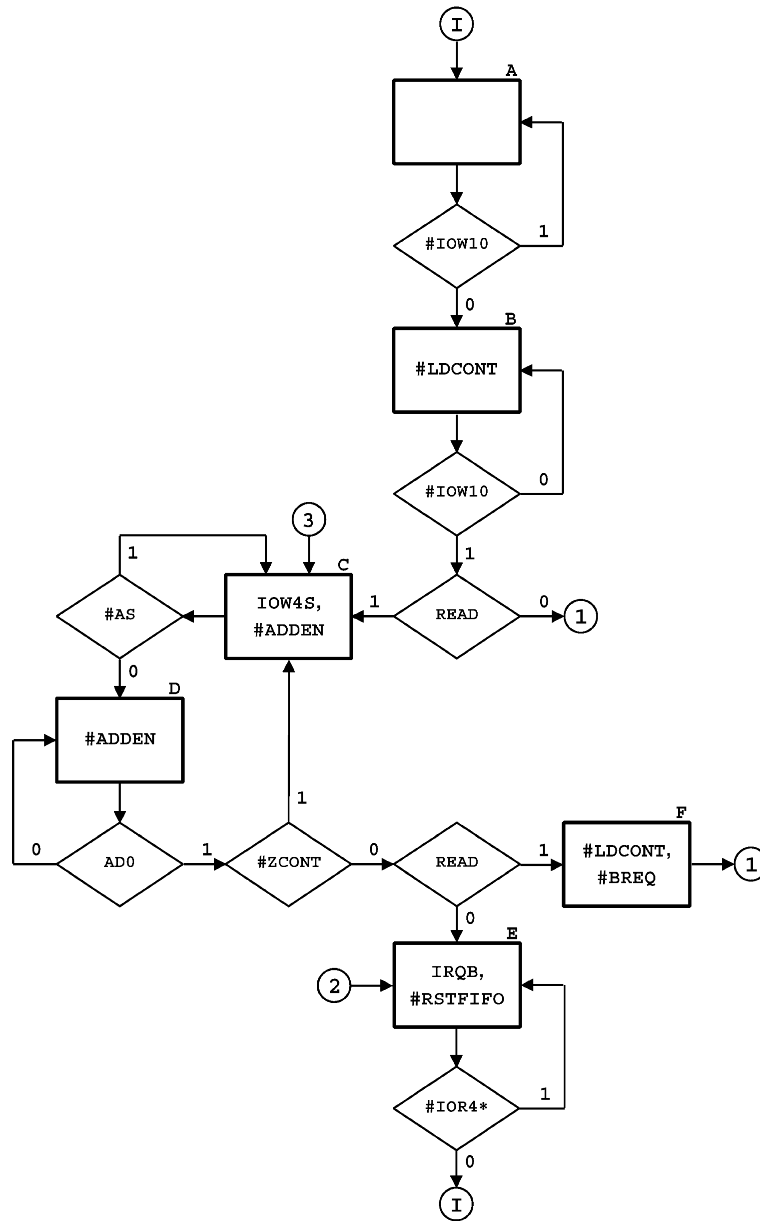

Fig. 4. Control unit state machine (I).

substitute system with the number of 16-bit words to transfer between the substitute and the target system. In every read access to the target system, the MC68000 state machine decrements the access counter value by one. However the bridge control unit decrements these counter value by two when it performs 32-bit data transfers between the PC memory and the bridge data buffer. After completing the last transfer, the control unit decrements the countdown value to zero. In any moment, the access counter can be reset by the control unit to the value loaded at the start of the bridge programming sequence.

The parity generator circuit calculates the parity bit of AD and C/#BE buses during the address phases of transactions initiated by the PCI/MC68000 bridge, data phases of memory write transactions initiated by the bridge and data phases of configuration read and I/O read transactions initiated by the Host/PCI bridge when the target device is the PCI/MC68000 bridge. The control unit initialises the latency counter with the value of the PCI/MC68000 bridge latency timer configuration register (Fig. 7). It also decrements the latency counter value by one unit in every PCI clock cycle. This decrement will continue until countdown value reaches zero, a premature transaction termination occurs or the last data phase of the transaction is being performed.

In the bridge programming sequence, the substitute system writes into the control register a bit (READ) that indicates the direction of the data transfers. Finally, the interrupt control register is very similar to that in the ISA-MC68000 interface designed in Ref. [1] with a single

Fig. 5. Control unit state machine (II).

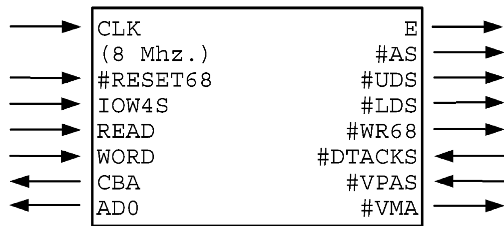

Fig. 6. MC68000 state machine external diagram.

difference, the connection of two additional signals (*IRQA* and *IRQB*) to its tri-state buffers. The first signal indicates that the interrupt request comes from a MC68000 device, whereas the second one informs that the interrupt request has been generated by the control unit at the end of the data block transfer (Fig. 4). Thus, when one of these two signals becomes active (or both ones), the corresponding buffer output leaves the high-impedance state and supplies a zero to the PCI #INTA interrupt pin.

When the PCI/MC68000 bridge interrupt service routine (ISR) starts, it reads the content of the interrupt register (P10 and P11 bridge I/O positions) to determine the IRQ source. If it is the bridge control unit, then the interrupt treatment consists of setting the flag that indicates the end of current data block transfer to the control program. After this, the ISR must read I/O P7 to clear the interrupt request. If there is also a pending IRQ from one or several MC68000 devices then the ISR will have to serve the requesting device/s. Unlike ISA interrupts, PCI interrupts are shareable [3,4]. Thus, when the current IRQ is not due to a MC68000 device or to the bridge control unit, our ISR will jump to the previous interrupt vector value since the IRQ will be related to the interrupt routine pointed by the old vector value.

4.3. Control unit

The state machine that implements the control unit (Figs. 4 and 5), named *PCI-MC68000 machine*, leaves its initial state once the data transfer direction has been written into the control register. This is the last operation in the PCI/MC68000 bridge programming sequence. If the READ bit has been set then the state machine that emulates asynchronous and synchronous MC68000 bus cycles [1,2], named MC68000 state machine (Fig. 6), must transfer a data block from the target system to the bridge data buffer (states C and D).

When MC68000 cycles have finished and, thus, a data block is in the bridge data buffer, the PCI/MC68000 bridge requests the PCI bus and resets the access and latency counters (states F and G). Once the PCI bus has been granted and is idle [3,4], the PCI-MC68000 machine enables the signals corresponding to the transaction address phase (state H), which is a memory write since the READ bit in the control register is set and so there is no turnaround cycle. Afterwards, data phases (state L) are performed in order to transfer the data block from the bridge data buffer to the PC system memory.

The PCI-MC68000 machine design (Fig. 5) considers the causes that originate a premature transaction termination (states K and M) by a slave device (Table 3), which in our context is the PC main memory, accessed by the PCI/MC68000 bridge through the Host/PCI bridge. The only transaction termination not considered here is *target abort*, since it is not frequent that the PC main memory or the Host/PCI bridge fails and thus the control unit design is simplified.

When the access counter value reaches one or two units, it indicates to the PCI-MC68000 machine that the next transfer to perform between the PCI/MC68000 bridge and the PC is the last one. During the data phase corresponding to this transfer (state J) the #FRAME signal is high and the latency counter value is no longer used until the next transaction since the control unit is about to surrender the PCI bus. Once the whole data block has been transferred from the PCI/MC68000 bridge data buffer to the substitute system, the PCI-MC68000 machine resets the address and data FIFO memories (state E) and also generates an interrupt request to the PC (Fig. 4), that will be cleared by the corresponding ISR by reading the bridge I/O P7. Then the PCI-MC68000 machine will go back to the initial state (state A).

Once the PCI/MC68000 bridge timeslice for the current transaction has finished (data unit latency counter reaches zero), its #GNT input remains low if no higher priority device has requested the bus. In this case, the bridge may keep on transferring data (state O) until the #GNT input becomes high. Then, the control unit will be allowed to make an additional transfer (state P) and next it will have to yield the bus for the requesting master. If the #GNT input is high for the bridge (because the arbiter has granted PCI bus ownership to other master during current transaction) when the latency counter reaches zero, the control unit will be allowed to make an additional transfer (state P) and then it will have to yield the bus [3,4].

If during the bridge programming sequence the PC has written a zero into the control register then its READ bit will be reset and, thus, the data block transfer will occur from the PC memory to the MC68000 system. The control unit operation is analogous to previous case (the transfer of a data block from the target system to the substitute system) though access order is now interchanged. First, the control unit must access the PC main memory zone where the data block resides for transferring it to the bridge data buffer (states H, I and L). As it is a memory read transaction, after the address phase a turn-around cycle (state I) must occur before the first data phase.

Once the whole data block is in the bridge data buffer, the MC68000 state machine (Fig. 6) will emulate the corresponding write cycles (states C and D). However, as they are now write access, the MC68000 state machine will activate the #WR68 signal in every cycle for the MC68000 selected

DEVICE ID		VENDOR ID	
00H	01H	00H	01H
STATUS		COMMAND	
02H	00H	00H	04H + 01H * R/W
CLASS CODE			REVISION ID
06H	80H	00H	01H
BIST (*NOT IMPL.*)	HEADER TYPE	LATENCY TIMER	CACHE LINE SIZE
00H	00H	R/W	(*NOT IMPLEMENTED*)
BASE ADDRESS 0			
R/W	R/W	R/W	F0H * R/W + 01H
BASE ADDRESS 1 (*NOT IMPLEMENTED*)			
00H	00H	00H	00H
BASE ADDRESS 2 (*NOT IMPLEMENTED*)			
BASE ADDRESS 3 (*NOT IMPLEMENTED*)			
BASE ADDRESS 4 (*NOT IMPLEMENTED*)			
BASE ADDRESS 5 (*NOT IMPLEMENTED*)			
CARDBUS CIS POINTER (*NOT IMPLEMENTED*)			
SUBSYSTEM ID (*NOT IMPLEMENTED*)		SUBSYSTEM VENDOR ID (*NOT IMPLEMENTED*)	
00H	00H	00H	00H
EXPANSION ROM BASE ADDRESS (*NOT IMPLEMENTED*)			
00H	00H	00H	00H
RESERVED			
RESERVED			
MAX_LAT	MIN_GNT	INTERRUPT PIN	INTERRUPT LINE
01H	FFH	01H	R/W

Fig. 7. Configuration registers.

device to capture the data [1], that will be placed on the lines M68D[0–15] by the bridge data buffer. When all cycles have been emulated, the PCI-MC68000 machine will reset bridge address and data FIFO memories (state E) and will request an interrupt to the PC (IRQB), which will be cleared by the corresponding ISR as in previous case. Then the PCI-MC68000 machine will go back to the initial state (state A).

4.4. Configuration space

In order to complete the PCI/MC68000 bridge design we must implement the necessary registers into PCI configuration address space. The contents of these registers are shown in Fig. 7. The logical expressions for signals that allow reads and writes to these registers (#CNFRx and #CNFWx) were concisely described in Section 4.1. The control program running in the PC can access the configuration registers using PCI BIOS [9]. Finally, a read access to configuration space always places a 32-bit datum on AD bus in order to preserve the PCI green bus nature [3].

A functional PCI device (or PCI function) has an individual configuration address space of 64 double words, numbered from 0 to 63. The 16 first double words are the configuration header region and the rest is used to store the information about user-defined PCI function specific features. PCI bus 2.1 specification [4] defines two formats for the header region: type zero for all devices except for PCI to PCI bridges and type one for these bridges [10]. Configuration registers shown in Fig. 7 belong to the header region. The registers marked in grey are mandatory. For every double word the low order byte corresponds to the right side.

As it is an experimental design, for command and status registers we will only consider those bits that are essential and will omit others that should be considered in a definitive design as, for example, those related to parity errors. Thus, implemented registers are: Vendor ID, Device ID, Command, Status, Revision ID, Class Code, Latency Timer,

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.functions.all;

entity decoder is
Port (CLK : in std_logic; ---  Control bus (PCI)
      RSTz : in std_logic;
      FRAMEz : in std_logic;
      TRDYz : out std_logic;
      IRDYz : in std_logic;
      STOPz : out std_logic;
      DEVSELz : out std_logic;
      IDSEL : in std_logic;
      F0z: in std_logic;
      T0z: in std_logic;
      CBEz : in std_logic_vector(3 downto 0); --- Ctrl bus (PCI)
      AD : in std_logic_vector(31 downto 0);  --- D/A bus (PCI)
      BASE_ADDRESS_0: in std_logic_vector (31 downto 0);
      TDATAPHz: out std_logic;
      TSTSz: out std_logic;
      IOR4z: out std_logic;
      IOR8z: out std_logic;
      IOW0123z: out std_logic;
      IOW45z: out std_logic;
      IOW6z: out std_logic;
      IOW7z: out std_logic;
      IOW89z: out std_logic;
      IOW10z: out std_logic;
      IOW11z: out std_logic;
      CNFR0z: out std_logic;
      CNFR4z: out std_logic;
      CNFR8z: out std_logic;
      CNFR12z: out std_logic;
      CNFR16z: out std_logic;
      CNFR20z: out std_logic;
      CNFR44z: out std_logic;
      CNFR48z: out std_logic;
      CNFR60z: out std_logic;
      CNFW4z: out std_logic;
      CNFW13z: out std_logic;
      CNFW16z: out std_logic;
      CNFW17z: out std_logic;
      CNFW18z: out std_logic;
      CNFW19z: out std_logic;
      CNFW60z: out std_logic);
end decoder;

architecture Behavioral of decoder is
    type state_machine is (A,B,C,D);
    signal cs,ns: state_machine;
    signal BIDSEL,BIRDYiz,BADDRz,BFRAMEiz,BDEVSELoz,BTRDYoz,BSTOPoz,
           BIORz,BIOWz,BCNFRz,BCNFWz: std_logic;
begin
    B_SYN: process (CLK,RSTz)
    begin
       if RSTz='0' then
          cs <= A;
       elsif CLK'event and CLK='1' then
          cs <= ns;
       end if;
    end process;
    BIDSEL <= IDSEL;
    BIRDYiz <= IRDYz;
    BADDRz <= '0' when AD=BASE_ADDRESS_0 else '1';
    BFRAMEiz <= FRAMEz;
    BIORz <= '0' when CBEz="0010" else '1';
    BIOWz <= '0' when CBEz="0011" else '1';
    BCNFRz <= '0' when CBEz="1010" else '1';
    BCNFWz <= '0' when CBEz="1011" else '1';
    B_ASY: process (cs,BIDSEL,BIRDYiz,BADDRz,BFRAMEiz,BIORz,BIOWz,
                    BCNFRz,BCNFWz,F0z,T0z)
    begin
       case cs is
           when A =>
               if (BIDSEL and not (BCNFRz or F0z or T0z)) ='1' or
                  (BIDSEL and (BCNFRz or F0z or T0z) and
                   not (BCNFWz or F0z or T0z) and BIRDYiz)='1'
                  or (not BIDSEL and not (BADDRz or BIORz)) ='1'
                  or (not BIDSEL and (BADDRz or BIORz)
                      and not (BADDRz or BIOWz) and BIRDYiz)='1' then
                  ns <= B;
               elsif (BIDSEL and (BCNFRz or F0z or T0z) and
                     (BCNFWz or F0z or T0z))='1'
                  or (not BIDSEL and (BADDRz or BIORz) and
                     (BADDRz or BIOWz)) ='1' then
                  ns <= A;
               elsif ((BIDSEL and (BCNFRz or F0z or T0z) and
                      not (BCNFWz or F0z or T0z)
                      and not BIRDYiz and BFRAMEiz)='1' or
                      (not BIDSEL and (BADDRz or BIORz)
                      and not (BADDRz or BIOWz) and not BIRDYiz and
                      BFRAMEiz)='1') then ns <= D;
               else ns <= C;
               end if;
           when B => if BIRDYiz='1' then ns <= B;
                     elsif BIRDYiz='0' and BFRAMEiz='1' then
                        ns <= D;
                     else ns <= C;
                     end if;
           when C => if (BIRDYiz or not BFRAMEiz) ='1' then
                        ns <= C;
                     else ns <= D;
                     end if;
           when D => ns <= A;
       end case;
       case cs is
           when A => if ((BIDSEL and (BCNFRz or F0z or T0z) and
                          not (BCNFWz or F0z or T0z)) or
                          (not BIDSEL and (BADDRz or BIORz) and
                          not (BADDRz or BIOWz)))='1' then
                        TDATAPHz<='1';BDEVSELoz<='1';BTRDYoz<='1';
                        TSTSz<='0';BSTOPoz<='0';
                     else TDATAPHz<='0';BDEVSELoz<='0';BTRDYoz<='0';
                          TSTSz<='0';BSTOPoz<='0';
                     end if;
           when B => TDATAPHz<='1';BDEVSELoz<='1';BTRDYoz<='1';
                     TSTSz<='0';BSTOPoz<='0';
           when C => DATAPHz<='0';BDEVSELoz<='1';BTRDYoz<='0';
                     TSTSz<='1';BSTOPoz<='1';
           when D => TDATAPHz<='0';BDEVSELoz<='0';BTRDYoz<='0';
                     TSTSz<='1';BSTOPoz<='0';
       end case;
    end process;
    IOR4z <= BIORz or BDEVSELoz or f2(AD(2),AD(3));
    IOR8z <= BIORz or BDEVSELoz or f3(AD(2),AD(3));
    IOW0123z <= BIOWz or BTRDYoz or f1(AD(2),AD(3)) or g1(CBEz);
    IOW45z <= BIOWz or BTRDYoz or f2(AD(2),AD(3)) or g2(CBEz);
    IOW6z <= BIOWz or BTRDYoz or f2(AD(2),AD(3)) or g3(CBEz);
    IOW7z <= BIOWz or BTRDYoz or f2(AD(2),AD(3)) or g4(CBEz);
    IOW89z <= BIOWz or BTRDYoz or f3(AD(2),AD(3)) or g2(CBEz);
    IOW10z <= BIOWz or BTRDYoz or f3(AD(2),AD(3)) or g3(CBEz);
    IOW11z <= BIOWz or BTRDYoz or f3(AD(2),AD(3)) or g4(CBEz);
    CNFR0z <= BCNFRz or BDEVSELoz or f4(AD(7 downto 2));
    CNFR4z <= BCNFRz or BDEVSELoz or f5(AD(7 downto 2));
    CNFR8z <= BCNFRz or BDEVSELoz or f6(AD(7 downto 2));
    CNFR12z <= BCNFRz or BDEVSELoz or f7(AD(7 downto 2));
    CNFR16z <= BCNFRz or BDEVSELoz or f8(AD(7 downto 2));
    CNFR20z <= BCNFRz or BDEVSELoz or f9(AD(7 downto 2));
    CNFR44z <= BCNFRz or BDEVSELoz or f10(AD(7 downto 2));
    CNFR48z <= BCNFRz or BDEVSELoz or f11(AD(7 downto 2));
    CNFR60z <= BCNFRz or BDEVSELoz or f12(AD(7 downto 2));
    CNFW4z <= BCNFWz or BTRDYoz or f5(AD(7 downto 2)) or g5(CBEz);
    CNFW13z <= BCNFWz or BTRDYoz or f7(AD(7 downto 2)) or g6(CBEz);
    CNFW16z <= BCNFWz or BTRDYoz or f8(AD(7 downto 2)) or g5(CBEz);
    CNFW17z <= BCNFWz or BTRDYoz or f8(AD(7 downto 2)) or g6(CBEz);
    CNFW18z <= BCNFWz or BTRDYoz or f8(AD(7 downto 2)) or g3(CBEz);
    CNFW19z <= BCNFWz or BTRDYoz or f8(AD(7 downto 2)) or g4(CBEz);
    CNFW60z <= BCNFWz or BTRDYoz or f12(AD(7 downto 2)) or g5(CBEz);
    TRDYz <= BTRDYoz;
    STOPz <= BSTOPoz;
    DEVSELz <= BDEVSELoz;
end Behavioral;
```

Fig. 8. Decoder circuit VHDL description.

Header Type, Base Address 0, Interrupt Line, Interrupt Pin, Min_Gnt and Max_Lat.

5. Implementation

VHDL (VHSIC Hardware Description Language) [11] has been used for describing the bridge hardware. VHDL is frequently used in hardware synthesis and descriptions are easily portable to ASICs or FPGAs, as in this case.

The modular description of the bridge allows the implementation complexity to be reduced. This modularity requires no changes with respect to the bridge design, described in Section 4. A Virtex E FPGA [12] has been used since it allows to implement FIFOs in its internal RAM bits and, there is enough 'free space' in order to increase the size of FIFOs, for example, and make tests with different lengths. Figs. 8 and 9 show the decoder circuit and the control unit descriptions respectively.

6. Performance analysis

In this section we compare the PCI/MC68000 bridge with a bus emulator interface which performs single 16-bit data transfers between a Pentium II PC with PCI bus and our example MC68000 system (the Hitachi A4010S robot control unit). We want to obtain two results: the relationship between the time taken by a word transfer performed by the simple interface and by the bridge, and the number of words to transfer for which bridge performance is better than interface performance. Before obtaining these results, we must set some initial assumptions that will help us in our study:

- The substitute system is a Pentium II PC with a Host/PCI bridge, an IDE adapter and the PCI/MC68000 bridge. PCI bus operates at 33 MHz and the 64-bit extension is not used. We will assume that the IDE adapter operates in one of the possible

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control_unit is
Port (IOW10z : in std_logic;    READ : in std_logic;
      IOR4z : in std_logic;     IOW4S : out std_logic;
      ASz : in std_logic;       AD0 : in std_logic;
      LDCONTz : out std_logic;  ULTRANSFz : in std_logic;
      ZCONTz : in std_logic;    LDLTCONTz : out std_logic;
      DLTCONTz : out std_logic; ZLTCONTz : in std_logic;
      ADDENz : out std_logic;   RSTFIFOz: out std_logic;
      CLK : in std_logic;       RSTz : in std_logic;
      FRAMEz : inout std_logic; TRDYz : inout std_logic;
      IRDYz : inout std_logic;  STOPz : in std_logic;
      DEVSELz : in std_logic;   IRQB : out std_logic;
      REQz : out std_logic;     GNTz : in std_logic;
      ITACz : out std_logic;    ISTSz : out std_logic;
      IADDRPHz : out std_logic; IDATAPHz : out std_logic);
end control_unit;

architecture Behavioral of control_unit is
    type state_machine is (A,B,C,D,E,F,G,H,I,J,K,L,M,N,N2,O,P,Q);
    signal cs,ns: state_machine;
    signal BREQz,BTRDYiz,BSTOPiz,BDEVSELiz,BFRAMEiz,BIRDYiz,
           BIRDYoz,BGNTz,BFRAMEoz: std_logic;
begin
    B_SYN: process (CLK,RSTz)
        begin
           if RSTz='0' then cs <= A;
           elsif CLK'event and CLK='1' then cs <= ns;
           end if;
        end process;
        BTRDYiz <= TRDYz;   BSTOPiz <= STOPz; BDEVSELiz <= DEVSELz;
        BFRAMEiz <= FRAMEz; BIRDYiz <= IRDYz; BGNTz <= GNTz;
        B_ASY: process (cs,BTRDYiz,BSTOPiz,BDEVSELiz,BFRAMEiz,
                        BIRDYiz,BGNTz,IOW10Z,READ,ASz,
                        ZCONTz,IOR4z,ULTRANSFz,ZLTCONTz)
        begin
           case cs is
              when A => if IOW10z='1' then ns <= A;
                        else ns <= B;
                        end if;
              when B => if IOW10z='0' then ns <= B;
                        elsif READ='1' then ns <= C;
                        else ns <= G;
                        end if;
              when C => if ASz='1' then ns <= C;
                        else ns <= D;
                        end if;
              when D => if AD0='0' then ns <=D;
                        elsif ZCONTz='1' then ns <=C;
                        elsif READ='0' then ns <= E;
                        else ns <= F;
                        end if;
              when E => if IOR4z='1' then ns <= E;
                        else ns <= A;
                        end if;
              when F => ns <= G;
              when G => if (BGNTz or not BFRAMEiz or
                            not BIRDYiz)='1' then ns <= G;
                        else ns <= H;
                        end if;
              when H => if READ = '0' then ns <= I;
                        elsif ULTRANSFz='0' then ns <= J;
                        else ns <= L;
                        end if;
              when I => if ULTRANSFz='0' then ns <= J;
                        else ns <= L;
                        end if;
              when J => if (BSTOPiz or BDEVSELiz or
                            not BTRDYiz)='0' then ns <= K;
                        elsif (not BSTOPiz or BDEVSELiz or
                               BTRDYiz)='0' then ns <= N;
                        elsif (BSTOPiz or BDEVSELiz or
                               BTRDYiz)='0' then ns <= M;
                        else ns <= J;
                        end if;
              when K => ns <= Q;
              when L => if (BSTOPiz or BDEVSELiz)='0' then ns <= K;
                        elsif (not BSTOPiz or BDEVSELiz or
                               BTRDYiz)='1' then ns <= L;
                        elsif ZLTCONTz='1' then
                           if ULTRANSFz='0' then ns <= J;
                           else ns <=L;
                           end if;
                        elsif BGNTz='0' then
                           if ULTRANSFz='0' then ns <= J;
                           else ns <=O;
                           end if;
                        elsif ULTRANSFz='0' then ns <= J;
                        else ns <=P;
                        end if;
              when M => ns <= N;
              when N => if READ='0' then ns <= N2;
                        else ns <= E;
                        end if;
              when N2=> ns <= C;
              when O => if (BSTOPiz or BDEVSELiz)='0' then ns <= K;
                        elsif (not BSTOPiz or BDEVSELiz or
                               BTRDYiz) = '1' then ns <= O;
                        elsif BGNTz='0' then
                           if ULTRANSFz='0' then ns <= J;
                           else ns <=O;
                           end if;
                        elsif ULTRANSFz='0' then ns <= J;
                        else ns <=P;
                        end if;
              when P => if (BSTOPiz or BDEVSELiz)='0' then ns <= K;
                        elsif (not BSTOPiz or BDEVSELiz or
                               BTRDYiz)='0' then ns <= Q;
                        else ns <= P;
                        end if;
              when Q => ns <= G;
           end case;
           case cs is
              when A => LDCONTz<='0';IOW4S<='0';ADDENz<='0';
                        BREQz<='0';IRQB<='0';RSTFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='0';BIRDYoz<='0';
                        FRAMEoz<='0';IADDRPHz<='0';DLTCONTz<='0';
                        ITACz<='0';ISTSz<='0';
              when B => LDCONTz<='1';IOW4S<='0';ADDENz<='0';
                        BREQz<='0';IRQB<='0';RSTFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='0';BIRDYoz<='0';
                        BFRAMEoz<='0';IADDRPHz<='0';DLTCONTz<='0';
                        ITACz<='0';ISTSz<='0';
              when C => LDCONTz<='0';IOW4S<='1';ADDENz<='1';
                        BREQz<='0';IRQB<='0';RSTFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='0';BIRDYoz<='0';
                        FRAMEoz<='0';IADDRPHz<='0';DLTCONTz<='0';
                        ITACz<='0';ISTSz<='0';
              when D => LDCONTz<='0';IOW4S<='0';ADDENz<='1';
                        BREQz<='0';IRQB<='0';RSTFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='0';BIRDYoz<='0';
                        BFRAMEoz<='0';IADDRPHz<='0';DLTCONTz<='0';
                        ITACz<='0';ISTSz<='0';
              when E => LDCONTz<='0';IOW4S<='0';ADDENz<='0';
                        BREQz<='0';IRQB<='1';RSTFIFOz<='1';
                        LDLTCONTz<='0';IDATAPHz<='0';BIRDYoz<='0';
                        BFRAMEoz<='0';IADDRPHz<='0';DLTCONTz<='0';
                        ITACz<='0';ISTSz<='0';
              when F => LDCONTz<='1';IOW4S<='0';ADDENz<='0';
                        BREQz<='1';IRQB<='0';RSTFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='0';BIRDYoz<='0';
                        BFRAMEoz<='0';IADDRPHz<='0';DLTCONTz<='0';
                        ITACz<='0';ISTSz<='0';
              when G => LDCONTz<='0';IOW4S<='0';ADDENz<='0';
                        BREQz<='1';IRQB<='0';STFIFOz<='0';
                        LDLTCONTz<='1';IDATAPHz<='0';BIRDYoz<='0';
                        BFRAMEoz<='0';IADDRPHz<='0';DLTCONTz<='0';
                        ITACz<='0';ISTSz<='0';
              when H => LDCONTz<='0';IOW4S<='0';ADDENz<='0';
                        BREQz<='0';IRQB<='0';RSTFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='0';BIRDYoz<='0';
                        BFRAMEoz<='1';IADDRPHz<='1';DLTCONTz<='1';
                        ITACz<='0';ISTSz<='0';
              when I => LDCONTz<='0';IOW4S<='0';ADDENz<='0';
                        BREQz<='0';IRQB<='0';RSTFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='0';BIRDYoz<='1';
                        BFRAMEoz<='1';IADDRPHz<='0';DLTCONTz<='1';
                        ITACz<='1';ISTSz<='0';
              when J => LDCONTz<='0';IOW4S<='0';ADDENz<='0';
                        BREQz<='0';IRQB<='0';RSTFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='1';BIRDYoz<='1';
                        BFRAMEoz<='0';IADDRPHz<='0';DLTCONTz<='0';
                        ITACz<='0';ISTSz<='0';
              when K => LDCONTz<='0';IOW4S<='0';ADDENz<='0';
                        BREQz<='0';IRQB<='0';RSTFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='1';BIRDYoz<='1';
                        BFRAMEoz<='0';IADDRPHz<='0';DLTCONTz<='0';
                        ITACz<='0';ISTSz<='0';
              when L => LDCONTz<='0';IOW4S<='0';ADDENz<='0';
                        BREQz<='0';IRQB<='0';RSTFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='1';BIRDYoz<='1';
                        BFRAMEoz<='1';IADDRPHz<='0';DLTCONTz<='1';
                        ITACz<='0';ISTSz<='0';
              when M => LDCONTz<='0';IOW4S<='0';ADDENz<='0';
                        BREQz<='0';IRQB<='0';STFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='1';BIRDYoz<='1';
                        BFRAMEoz<='0';IADDRPHz<='0';DLTCONTz<='0';
                        ITACz<='0';ISTSz<='0';
              when N => LDCONTz<='0';IOW4S<='0';ADDENz<='0';
                        BREQz<='0';IRQB<='0';RSTFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='0';BIRDYoz<='0';
                        BFRAMEoz<='0';IADDRPHz<='0';DLTCONTz<='0';
                        ITACz<='0';ISTSz<='1';
              when N2=> LDCONTz<='1';IOW4S<='0';ADDENz<='0';
                        BREQz<='0';IRQB<='0';RSTFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='0';BIRDYoz<='0';
                        BFRAMEoz<='0';IADDRPHz<='0';DLTCONTz<='0';
                        ITACz<='0';ISTSz<='0';
              when O => LDCONTz<='0';IOW4S<='0';ADDENz<='0';
                        BREQz<='0';IRQB<='0';RSTFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='1';BIRDYoz<='1';
                        BFRAMEoz<='1';IADDRPHz<='0';DLTCONTz<='0';
                        ITACz<='0';ISTSz<='0';
              when P => LDCONTz<='0';IOW4S<='0';ADDENz<='0';
                        BREQz<='0';IRQB<='0';STFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='1';BIRDYoz<='1';
                        BFRAMEoz<='0';IADDRPHz<='0';DLTCONTz<='0';
                        ITACz<='0';ISTSz<='0';
              when Q => LDCONTz<='0';IOW4S<='0';ADDENz<='0';
                        BREQz<='1';IRQB<='0';RSTFIFOz<='0';
                        LDLTCONTz<='0';IDATAPHz<='0';BIRDYoz<='0';
                        BFRAMEoz<='0';IADDRPHz<='0';DLTCONTz<='0';
                        ITACz<='0';ISTSz<='1';
           end case;
        end process;
        REQz <= BREQz; IRDYz <= BIRDYoz; FRAMEz <= BFRAMEoz;
end Behavioral;
```

Fig. 9. Control unit VHDL description.

DMA modes [13] and so, it becomes bus master for transferring data between the peripheral and the PC main memory.

- The PCI arbiter uses a rotational priority scheme [3]. Furthermore, PCI/MC68000 bridge and the IDE adapter are continuously requesting PCI bus since both devices need to perform a high number of operations.
- The substitute system main memory is fast enough [14] to allow a double word per PCI clock transfer

rate through the PCI bus. We will assume that the PC main memory does not need to insert wait states in any of the transaction data phases initiated by the PCI/MC68000 bridge.

- The asynchronous MC68000 cycle will be used to transfer data between the target system and the PCI/MC68000 bridge data buffer since we only access asynchronous devices in order to control our example system (i.e. the Hitachi A4010S robot), mainly the encoders and PWM registers of its four axes [15]. Furthermore, we may assume that no wait states are generated during MC68000 cycles as the mentioned devices are fast enough.
- All data will be transferred from the target system to the substitute system.

First, we must know the total time taken by a bus cycle emulation performed by the simple PCI/MC68000 interface, which performs single 16-bit data transfers between the substitute and the target systems. The interface design is similar to the one designed in Refs. [1,2], though the decoder circuit has been modified since the substitute system bus is now the PCI instead of the ISA bus. Furthermore, the user register set must include the necessary logic to support the aspects related to the IRQ that the interface must generate once the MC68000 bus cycle emulation has finished. In the I/O map, initial 'R' denotes a read access and initial 'W' denotes a write access.

P0	ADDRESS REGISTER	A[0−7]	(W)
P1	ADDRESS REGISTER	A[8−15]	(W)
P2	DATA REGISTER	D[0−7]	(R,W)
P3	DATA REGISTER	D[8−15]	(R,W)
P4	PCI-MC68000 INTERFACE IRQ STATUS		(R)
	CONTROL REGISTER		(W)
P5	INTERRUPT CONTROL REGISTER		(R,W)
P6	CLEAR PCI-MC68000 INTERFACE IRQ		(R)
	#RESET SIGNAL ACTIVATION		(W)

Thus, to transfer a word from the MC68000 system to the PC, we must perform a series of access to the user register set. Furthermore, the MC68000 bus cycle time and interrupt latency in substitute system must be considered. So, the total time of the transfer, measured in PCI clock cycles, is due to following actions:

- MC68000 system access address is written into the address registers (address and data phase: 2 cycles).
- Transfer direction (read) is written into the control register (address and data phase: 2 cycles).
- MC68000 bus cycle emulation (4.5 ISA clock cycles in worst case [1], which are equivalent to 18.75 PCI clocks assumed that ISA bus operates at 8 MHz.).
- Latency of the interrupt generated by the PCI/MC68000 interface. We will use the result obtained in Ref. [15] as a valid approximation (11 ISA clocks which are equivalent to 46 PCI clocks). We may discard the propagation delay due to the PCI/ISA bridge programmable interrupt router, which connects PCI interrupt pins (#INTA, #INTB, #INTC and #INTD) to IRQ lines in the PC chipset..
- Data available flag is set by the ISR for robot control program running in the PC. We may discard the flag setting time since the PC system memory access time is small compared with the duration of a PCI clock [14].
- Data registers are read by the control program in order to get the 16-bit data coming from MC68000 system (address phase, turn-around cycle and data phase: 3 cycles).
- MC68000 data is written into the PC system memory by control program. Again, we may discard this time as it is an access to PC system memory [14].

Adding all the terms, the read of a MC68000 system position (i.e. the sum of all previous times), including all the necessary read and write accesses to the user register set and the handling of the interrupt generated at the end of the bus cycle emulation, is equal to 72 PCI clock cycles using a simple PCI MC68000 interface.

Next, we will consider the PCI/MC68000 bridge control unit design (Figs. 4 and 5), in order to know the duration of a data block transfer $T_R(x)$ between the MC68000 system and the Pentium II based PC As the substitute system bus is PCI, we must consider the causes that originate a premature transaction termination since bus acquisition penalizes the total duration of data block transfer seriously. First, if a data block has a size of x words as the PCI bus is 32-bit wide, $\lceil x/2 \rceil$ double words must be transferred between PCI/MC68000 bridge data buffer and PC main memory. The next items we will describe the actions that originate the different terms of $T_R(x)$, which are given by Eqs. (5)–(12) and are expressed in PCI clock cycles.

• PCI/MC68000 bridge programming task. It writes the x MC68000 addresses to the bridge address memory (P0, P1, P2 and P3 I/O positions), writes the PC memory zone base address 16 LSB to the bridge address counter (P4 and P5 positions) and the next 8 bits to page register (P6 position) and, finally, writes the transfer direction bit to the control register (P10 position). As a PCI I/O write cycle consists of an address and a data phase, the total time is calculated in Eq. (5).

• MC68000 bus cycle emulation. First, #IOW10 transition from low to high is detected with a delay of one PCI clock due to the co-ordination between decoder circuit state machine (Fig. 3) and the PCIMC68000 machine (Figs. 4 and 5). The design of the last machine allows chaining the end of a MC68000 cycle with the beginning of another one, so each cycle lasts only 4 ISA clocks (and not 4.5 ISA clocks as in PCI-MC68000 interface). Finally, the control unit spends another PCI clock on initializing bridge access

counter and activating #REQ signal in order to request PCI bus. The total time is calculated in Eq. (6).

• PCI Bus use latency (the meaning of the LAT_REQ variable will be explained immediately after describing all $T_R(x)$ terms), transaction address phase (one PCI clock) and $\lceil x/2 \rceil$ data phases of one PCI clock each corresponding to the $\lceil x/2 \rceil$ double words to be transferred from the bridge data buffer to main memory. The total time is calculated in Eq. (7).

• Penalty for bus ownership time (timeslice) expiration. In the PCI/MC68000 bridge the Min_Gnt configuration register (Fig. 7) is stored with the highest possible value. Thus, if the transaction does not finish prematurely, we will assume that the bridge may perform 255 one clock data transfers without releasing the bus (the Latency Timer 8-bit configuration register maximum value and the value of *TIME_SLICE* variable) since although the address phase lasts a PCI clock, PCI bus specifications allows the initiator to make an additional transfer once it has exhausted its timeslice (Latency Timer reaches zero) before releasing the bus. This penalty is calculated in Eq. (8). Its first clock is due to #REQ signal activation, that will be detected on the rising edge (the end) of that clock. The last cycle of the penalty corresponds to the transaction address phase.

• Penalty for the last double word transfer within current cache line. The number of transfers that finish due to a *disconnect A* or *B* and cause the associated penalty may be calculated as $(\lceil x/2 \rceil - 1) * P_{DISC_AB}$. Although $\lceil x/2 \rceil$ is the total number of PCI data transfers, a different penalty is associated with the last transfer, as we will explain in a latter paragraph. Disconnect penalty in Eq. (9) consists of following terms: a first clock used to hold #IRDY signal active, a second clock in which #REQ signal is activated for requesting PCI bus, the bus use latency (*LAT_REQ*) and a last clock due to the new transaction address phase (see Fig. 5).

• Penalty for a snoop hit on a modified line. The number of transfers that finish due to a retry and cause the associated penalty may be calculated as $\lceil x/2 \rceil * P_{RETRY}$. Finally, the retry penalty in Eq. (10) consists of the following terms: a first clock due to the failed data transfer, which does not appear in the previous penalty because in that case the data transfer did occur, a second clock used to hold #IRDY signal active, a third clock in which #REQ signal is activated for requesting PCI bus, the bus use latency (*LAT_REQ*) and a last clock due to the new transaction address phase (see Fig. 5).

• Time take by the last data transfer. If the last data transfer terminates normally then the PCI/MC68000 bridge control unit must wait only a PCI clock, during which #IRDY S/T/S signal is held high by the #ISTS signal (see Fig. 5) before being tristated. The probability of this event is calculated in the first term of Eq. (11) as a single PCI clock cycle in which premature transaction termination (disconnect and retry) does not occur. However, when the PCI/MC68000 bridge transaction data phase finishes due to a *disconnect A* or *B* from the Host/PCI bridge, the control unit must wait two PCI clocks (the second term of same equation). During the first one, the control unit must hold #IRDY signal active, and the second one corresponds to a normally terminated transfer as described at the beginning of this paragraph. Though it was a premature transaction termination, the data transfer has been completed and as it was the last double word from the data block, the control unit does not need to request the PCI bus again.

• PCI/MC68000 bridge IRQ to the substitute system and control program end-of-transfer flag setting by the ISR. This final time is mainly due to the interrupt latency of the bridge IRQ once the data block transfer has finished. As we did at the beginning of current section when determining the duration of a 16-bit transfer from the MC68000 system to the PC performed by the PCI-MC68000 interface, we will use again the result obtained in Ref. [15] as a valid approximation (11 ISA clocks which are equivalent to 46 PCI clocks). Furthermore, the substitute system spends two PCI clocks reading PCI/MC68000 bridge I/O P7 position in order to clear the IRQ generated by the last device. Finally, the corresponding ISR sets the flag which indicates to the control program running in the PC the end of current block data transfer. Total time is calculated in Eq. (12).

$$T_{R1} = 2 * \lceil x/2 \rceil + 2 + 2 \tag{5}$$

$$T_{R2} = 1 + x * T_{ACC_MC68000} + 1 \tag{6}$$

$$T_{R3} = LAT_REQ + 1 + \lceil x/2 \rceil \tag{7}$$

$$T_{R4} = \lfloor \lceil x/2 \rceil / TIME_SLICE \rfloor * (1 + LAT_REQ + 1) \tag{8}$$

$$T_{R5} = (\lceil x/2 \rceil - 1) * P_{DISC_AB} * (2 + LAT_REQ + 1) \tag{9}$$

$$T_{R6} = \lceil x/2 \rceil * P_{RETRY} * (3 + LAT_REQ + 1) \tag{10}$$

$$T_{R7} = (1 - P_{DISC_AB} - P_{RETRY}) + 2 * P_{DISC_AB} \tag{11}$$

$$T_{R8} = LAT_IRQB + 2 \tag{12}$$

In order to acquire PCI bus ownership, the PCI/MC68000 bridge must hold its #REQ signal active and wait for a PCI clock rising edge in which its #GNT input is low and the PCI bus has been released by both the Host/PCI bridge and the IDE adapter, which we suppose are the other bus masters in the substitute system, so the bus is in idle state (#FRAME and #IRDY signals are high) [3,4]. Then the bridge may activate #FRAME signal and start the transaction. The Host/PCI bridge and the IDE adapter access the PCI bus during a time that can be calculated as the sum of the following terms:

- Bus ownership time, controlled by *TIME_SLICE* variable, whose allowed values are from 0 to 255. We will suppose an average case where both the Host/PCI bridge and the IDE adapter are assigned the variable mean value in excess (*TIME_SLICE* $*$, which is equal to 128 PCI clocks.
- Time taken by one additional transfer, which is supposed to last generally one PCI clock.

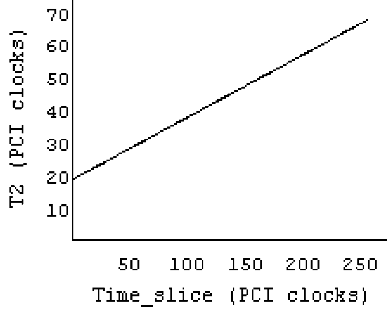

Fig. 10. Evolution of T_2 with Host/PCI bridge and IDE adapter timeslice.

- A PCI clock during which the bus is in idle state, that is necessary for next master to be able to access to PCI bus.

Once both masters have finished using the bus, the PCI/MC68000 bridge may access it. Assuming the initial considerations stated before (rotational priority scheme and continuous PCI bus requests by the Host/PCI bridge and the IDE adapter), the time in PCI clocks that the PCI/MC68000 bridge must wait from the instant when the PCI arbiter samples the bridge #REQ signal active (on a clock rising edge) until the bridge may start a transaction (LAT_REQ) is calculated in Eq. (13).

$$LAT_REQ = TIME_SLICE * +1 + 1 + TIME_SLICE * +1 + 1 \quad (13)$$

As we have assumed continuous bus request by the other bus masters, when the PCI/MC68000 bridge finishes its bus ownership time (or timeslice) it will not be allowed to perform more transfers, but it will have to yield the bus since PCI arbiter will previously have granted it to one of the other masters (hidden bus arbitration [4]). Thus, the probability that #GNT signal remains low for the PCI/MC68000 bridge once its timeslice has expired (P_{GNT}) will be zero under our assumptions.

In order to complete the calculation of $T_R(x)$, we must consider those aspects related to premature transaction termination, in particular to target-initiated termination [3].

- Disconnect: From all the possible causes by which make a PCI target device finish by issuing a *disconnect A* or *B* (Table 3), the only one to consider is the transfer of last

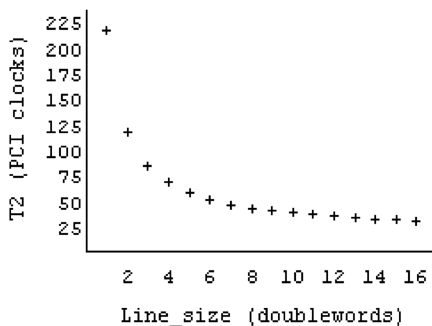

Fig. 11. Evolution of T_2 with cache line size.

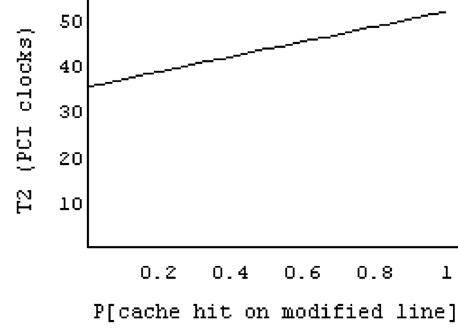

Fig. 12. Evolution of T_2 with probability of cache hit on a modified line.

double word within the current cache line. As the PCI/MC68000 bridge accesses the PC main memory sequentially, the probability P_{DISC_AB} of accessing the last double word within the current main memory line (which corresponds to a cache line in a cache hit) is calculated in Eq. (14), where the $LINE_SIZE$ variable is the cache line size expressed in double words. In the Pentium II processor [5], the size of L1 data cache and L2 unified cache line is 8 double words (32 bytes) for both caches and so, in this case, P_{DISC_AB} is equal to 1/8.

$$P_{DISC_AB} = 1/LINE_SIZE \quad (14)$$

- Retry: From all the possible causes by which a PCI target device finishes by issuing a *retry* (Table 3), the only one to consider is a snoop hit on a modified cache line. As the PCI/MC68000 bridge accesses the PC main memory sequentially, it accesses a different main memory line of $LINE_SIZE$ double words with a frequency of $1/LINE_SIZE$, this is the probability of accessing a new memory line in current access. On the other hand, we must know the probabilities for this line to exist in the cache (P_{CACHE_HIT}) and to be modified (P_{MODIF_LINE}). These two probabilities are difficult to calculate as current content of PC caches (L1 and L2) will depend on the characteristics of previous memory access performed by the control program running in the PC. Furthermore, the processor is also executing the operating system and other tasks and so, the execution of all these programs also affects the content of caches. We will assume, as an initial approximation, that the product of both probabilities, due to the conjunction of ‘cache hit’ and ‘modified cache line’ events, is equal to 0.5. Using Eq. (15), the probability of a snoop hit on a modified cache line when PCI/MC68000 bridge is accessing to the PC main memory zone is equal to 1/16.

$$P_{RETRY} = (1/LINE_SIZE) * PCACHE_HIT * PMODIF_LINE \quad (15)$$

Target Abort: A PCI target device must signal target abort when it detects a fatal error or it will never be able to respond to a new transaction. In normal conditions it is very difficult that PC main memory (the transaction target device) fails or that a parity error is generated in a

PCI/MC68000 bridge transaction address phase. In our context, these two events are the possible causes by which the slave device would respond to a transaction by signalling target abort [3]. Thus, the probability that this situation occurs (P_{ABORT}) may be discarded.

Once $T_R(x)$ has been calculated as the sum of terms from Eqs. (5)–(12), we must divide it between the data block size in words (x) in order to obtain $T_C(x)$. Next, we will let the data block size tend to infinity in order to calculate T_2(16), which represents the ideal case in which the number of words to transfer is so high that the amount of PCI clocks in $T_R(x)$, caused by mandatory operations that must be performed only once (i.e. PCI/MC68000 bridge programming, bridge interrupt handling, etc.), may be discarded. Thus, the time the bridge takes to transfer one word in this case is equal to 43 PCI clocks.

$$T_2 = \mathrm{Lim}(x \to \infty)[T_R(x)/x]$$
$$= 1 + T_{ACC_MC68000} + 1/(2 * TIME_SLICE)$$
$$* (1 + LAT_REQ + 1) + P_{DISC_AB} * (2 + LAT_REQ + 1)/2$$
$$+ P_{RETRY} * (3 + LAT_REQ + 1)/2 \qquad (16)$$

Figs. 10–12 show three graphs which describe the evolution of T_2 depending on a range of possible values for variables LAT_REQ (which depends on $TIME_SLICE$), $LINE_SIZE$ and P_{RETRY} respectively. These graphs have been developed using 'Mathematica' [16]. The first graph shows T_2 as a function of the value of $TIME_SLICE$ for the Host/PCI bridge and IDE adapter, using values between 0 and 255. As LAT_REQ depends on $TIME_SLICE$ (see Eq. (13)), penalties due to bus ownership time expiration and premature transaction termination increase with it and so does T_2, whose initial and final values are 18.9 and 67.7 respectively. Thus, a high timeslice for the Host/PCI bridge and the IDE adapter has a negative influence on PCI/MC68000 bridge performance, which is obvious.

The second graph (Fig. 11) describes the evolution of T_2 as a function which depends on PC cache line size. The range for $LINE_SIZE$ variable is between 1 and 16 double words. LAT_REQ and 'cache hit on a modified line' probability remain constant, so the $TIME_SLICE$ variable will be equal to the mean value ($TIME_SLICE*$) and the mentioned probability equal to 0.5. The current graph consists of a set of points from an hyperbola where T_2 has as initial and final values 216.2 and 31 respectively. It is obvious that PC cache line size has a notable influence on bridge performance since P_{RETRY} and P_{DISC_AB}, to which important penalties are associated (see Eqs. (14)–(16)), depend on this variable.

The third graph (Fig. 12) shows the evolution of T_2 as a function of the probability of a cache hit on a modified line ($P_{CACHE_HIT} * P_{MODIF_LINE}$). This probability varies from 0 to 1 and is one of the factors of P_{RETRY}. The other two variables remain constant. The result is a linear segment where T_2 varies between 35.1 and 51.6. As it was predictable, the increase of P_{CACHE_HIT} and P_{MODIF_LINE} has a negative influence on the PCI/MC68000 bridge performance because of the associated penalty in T_2 (see Eq. (16)). This influence is smaller than in the two previous cases.

Once we have obtained T_2 (43 PCI clocks), the relationship between T_1 (72 PCI clocks) and that one determines the performance improvement ideal factor, which is approximately equal to 1.7. This means that the PCI/MC68000 bridge ideally takes a little more than half the time that the simple non DMA based PCI-MC68000 interface to transfer a word of the data. This performance improvement is not very high mainly due to two reasons. First, we have assumed that the other bus masters in the substitute system (Host/PCI bridge and IDE adapter) are continuously requesting the PCI bus which, in fact, is a very extreme case. Furthermore, the slow MC68000 bus emulation cycle time ($T_{ACC_MC68000}$) prevents higher bridge performance improvements. In fact, if we eliminate this time from T_1 (4.5 ISA clocks) and T_2 (4 ISA clocks), the improvement factor rises above 2.

Finally, in order to calculate the minimum data block length from which the PCI/MC68000 bridge takes less time to transfer a data block than the simple PCI-MC68000 interface and, thus, performance of the first is better than that of the second, we only must solve Eq. (17). As x is equal to 10, we may conclude that the minimum data block length (X_{MIN}) for which the bridge is faster is equal to 10 words (or 20 bytes).

$$T_R(x) \leq x * T_1(x) \qquad (17)$$

7. Conclusions

- We have applied bus emulation and direct memory access techniques in order to design and implement a bus emulator interface as an I/O module with DMA (PCI/MC68000 bridge), which can transfer data blocks between a substitute system (Pentium II PC) and a target system (using a Hitachi A4010S scara robot as an example).
- We have also quantified the PCI/MC68000 bridge performance improvement respect to a PCI interface which emulates MC68000 bus cycles but does not use DMA techniques for transferring data bursts between itself and the substitute system main memory. The results obtained from the mathematical model (5)–(16) developed to measure the bridge performance improvement are conclusive and satisfactory (see Figs. 10–12).
- The methodology described in this work to design the

PCI/MC68000 bridge (based on bus emulation and DMA techniques) is general (Fig. 1) and thus, it can be applied to the design of bridges that communicate any bus (not only the PCI bus) with any other bus (a system bus, an I/O bus or another kind of bus).

- Our modular design methodology has made possible to reuse an important part (one of the three main state machines) of our original design [1,2].

References

[1] J.M. Rodríguez Corral, Diseño y Evaluación de una Unidad de Control para Robot Industrial Basada en i486 (in Spanish). Research Report, Universidad de Sevilla, Spain, June 1995.

[2] J.M. Rodríguez Corral, A. Civit Balcells, G. Jiménez Moreno, J.L. Sevillano Ramos, F. Díaz del Río, A Study of Bus Emulation. Application to M68000 Based Systems, Microprocessors and Microsystems 21 (5) (1998) 319–327.

[3] T. Shanley, D. Anderson, PCI System Architecture. Mindshare, Inc., third ed., Addison-Wesley, USA, 1995.

[4] PCI Special Interest Group, PCI Local Bus Specification. Revision 2.1, USA, 1995.

[5] T. Shanley, Pentium Pro and Pentium II Processor System Architecture. Mindshare, Inc., second ed., Addison-Wesley, USA, 1998.

[6] S. Casell, E. Faldella, F. Zanichelli, Performance Evaluation of Processor Architectures for Robotics, Proceedings of IEEE COMPEURO'91: Advanced Computer Technology, Reliable Systems and Applications, May, IEEE Society, Bologna, Italy, 1991.

[7] Motorola Semiconductor Products Inc., MC68000 16-bit Microprocessor, Scotland, 1984.

[8] W. Stallings, Computer Organization and Architecture, fourth ed., Prentice-Hall, Inc, UK, 1995.

[9] PCI Special Interest Group, PCI BIOS Specification. Revision 2.1, USA, 1994.

[10] PCI Special Interest Group, PCI-to-PCI Bridge Architecture Specification. Revision 1.0, USA, 1994.

[11] IEEE Standard VHDL Language Reference Manual. IEEE Std. 1076, 2000 Edition.

[12] VIRTEXTM-E 1.8 V Field Programmable Gate Array. Xilinx, November 2000. http://www.xilinx.com/partinfo/ds022.pdf.

[13] F. Schmidt, The SCSI Bus and IDE Interface, Protocols, Applications and Programming, second ed., Addison-Wesley, England, 1997.

[14] Y. Katayama, Trends in semiconductor memories, IEEE Micro 17 (6) (1997) 10–17.

[15] M.W.S. Macauley, Interrupt Latency in Systems Based on Intel 80 × 86 Processors, Microprocessors and Microsystems 22 (2) (1988) 121–126.

[16] Wolfram Research, Inc., Mathematica: A System for Doing Mathematics by Computer. Version 3.0 for Windows. User's Manual, 1996.