ELSEVIER

Microprocessors and Microsystems 21 (1998) 319–327

MICROPROCESSORS AND
MICROSYSTEMS

A study of bus emulation. Application to M68000 based systems

José M. Rodríguez*, A. Civit-Balcells, Gabriel Jiménez, José L. Sevillano, F. Díaz

Facultad de Informática, Universidad de Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain

Received 14 March 1997; received in revised form 23 May 1997; accepted 18 June 1997

Abstract

With the important development of computer architectures, bus emulation becomes a very important tool. In this paper, methods for bus emulation are generally studied. We present several actual systems using this approach and introduce a new practical application. This application permits a PC-compatible system with an ISA BUS to control an embedded system originally based on a MC68000 microprocessor. This paper demonstrates the easy use of bus emulation techniques to modernize obsolete industrial controllers. © 1998 Published by Elsevier Science B.V.

Keywords: Bus emulation; Embedded systems; Standard buses; Controller update

1. Introduction

Bus emulation [1,2], as is considered in this paper, can be loosely defined as the techniques used to make a machine (Substitute System) with a particular native bus look, from the point of view of an external device, as if it also includes a bus that is characteristic of another system (Target System).

Bus emulation can thus serve two main objectives:

1. it can be used as a means of interfacing with a standard bus; and
2. it can be used to increase the performance of obsolete systems.

Before going into the details of these two objectives it is important to point out the two main approaches to bus emulation:

1. The address space of the target system is included as part of the substitute system's address space. We shall refer to this technique as address inclusion.
2. The address space of the target system is generated by a state machine that is considered as a peripheral by the substitute system.

These methods are related to the objectives above but they are also strongly influenced by the characteristics of the target and substitute buses.

* Corresponding author.

2. Bus emulation as an interface to a standard bus

Considering now the motivations for bus emulation, if we want to have a standard bus in a system with a new processor with few support chips, it might be simpler to emulate the bus of a popular processor and use the interface chips that are available for it rather than implementing the standard bus specifications directly. This has two important side advantages:

- other peripherals for the emulated processor can also be used; and
- the target bus only has to be emulated to the accuracy required by the chips that are interfaced to it.

This second consideration is especially interesting because it means that only the signals that are really used in the system have to be emulated and that the timing requirements are imposed by those chips. These timing requirements are usually much simpler to meet than the strict ones imposed by standard buses. Some buses like Future bus, MultiBusII, EISA or PCI are very difficult to implement without specific VLSI support.

Thus it is not strange that several manufacturers of Risc based computers have emulated i486 or Pentium buses in order to be able to use one of the standard EISA or PCI chip sets. As an example, the DEC2000/300 [3] (which uses the DECchip 21064 Alpha AXP 64 bit processor) implements the EISA bus in this way. As this is an ARM compliant machine (required to run Windows NT) it shares much of its I/O subsystem with a standard high-end PC and, thus, it makes strong use of the emulated bus (SCSI controllers, Video and LAN adapters).

The techniques used in emulation, as well as the performance of the interface circuits, should depend clearly on the fraction of system I/O accesses and the part of these that go to the emulated bus. We introduce the following notation: $f =$ access fraction (exact meaning depending on the subscript), $t_a =$ access time, $t_d =$ interface delay and $h_w =$ write buffer hit rate.

We shall also use the following subscripts: r (for read access), w (for write access), m (for memory access), io (for I/O access), n (for native bus), t (for target bus) and s (for standard bus).

With this notation system the access time can be calculated as in [4]. It is given by

$$t_a = f_r \cdot t_{ar} + f_w \cdot t_{aw} \tag{1}$$

$$t_{ar} == f_m \cdot t_{arm} + f_{io} \cdot t_{ario} \tag{2}$$

$$t_{ario} = f_n \cdot t_{aion} + f_t \cdot \left[t_{dt} + f_{tn} \cdot t_{aiotn} + f_{ts} \cdot \left(t_{ds} + t_{aiots} \right) \right] \tag{3}$$

For write accesses with a write buffer:

$$t_{aw} = h_w \cdot t_{bh} + \left(1 - h_w\right) \cdot \left(f_m \cdot t_{awm} + f_{io} \cdot t_{awio}\right) \tag{4}$$

where t_{awio} is usually very similar to t_{aior}.

Considering Eqs. (3) and (4) it is clear that the performance of this type of systems depends greatly on the fraction of I/O access, the fraction of these that are to the target and standard buses and the delays imposed by the translation. Thus in a system where the fractions of access to the buses are naturally high like in the DEC 2000/300, the only way to preserve performance is to keep translation delays as low as possible.

In the whole of the above discussion we have maintained that an access to the target system, or even to a standard bus, maps to a single access in the substitute system. This is the case when address inclusion is used. This technique maps the address space of the target subsystem to a region (or a set of regions) in the address space of the substitute processor. Thus, in the case of read accesses, the substitute processor must wait until the emulated bus cycle ends. All high-performance processors use posted writes and, thus, they do not affect processor performance unless they represent a large fraction of the accesses. This is not usually the case in real-world systems. The address inclusion solution gives optimal performance when the speeds of the target and substitute buses are of the same order of magnitude.

When the emulated bus is much slower than the substitute system bus then it is better to use the state machine technique. In this case the target system is considered as a peripheral by the substitute system. The emulated bus cycles are generated by using a set of state machines [2]. With this approach at least two substitute accesses are used for each target bus access (one to start the access, one to collect the data and maybe some to check if access is finished). In this case we can start an access and do something else while it is going on. If we use address inclusion, external processor activity is usually blocked while the access takes place. In some systems (like i960 RP [5]) a split access technique is used where, in read accesses, the first read starts the access and a second read, to the same position, captures the data.

3. Bus emulation as a solution for obsolete systems

Two very different important cases exist here: replacement of obsolete CPUs in compatible PCs and industrial I/O controllers.

Other systems might be studied but we think that only in the two cases above can bus emulation techniques be economically efficient. The reasons for this are completely different in each case but the basis is that, for bus emulation to be justified, either a large market must exist for the adaptation or we must find systems where the I/O peripherals are the most expensive and hardest-to-replace part of the system.

PC replacement processors appear in three different flavours:

- Daughterboards that use a standard current-generation processor. In this case the address inclusion method is always used (usually address spaces are equal). An example of this type of board is the IBM designed "SLC now", which replaces an i386 by an IBM 486SLC.
- Chips designed as replacement processors like the CYRIX M1 486 DLC, SLC and DRX2. In this case the processor's external native bus is the older processor external bus. If we consider this as a bus emulation we could as well say that every processor nowadays emulates its external bus.
- Boards that include a socket for a next-generation processor. The idea is that when the current processor becomes obsolete or is not powerful enough for the user, a new processor can be inserted in the free socket to substitute for the original one without actually removing it from the system. This method is used by Intel overdrive processors.

The case of industrial controllers is very different. In order to make a CPU replacement based on bus emulation attractive, several conditions must hold:

- the process interface hardware must be much more expensive than the processor (or processors) used;
- the performance of the interface hardware is adequate and thus it is not necessary to replace it;
- it is not necessary to maintain software compatibility with the old system (if it were necessary, the substitute processor would have to be software-compatible with the old one); and

- no processor upgrade option had originally been designed into the system.

4. MC68000 emulation on an ISA bus PC

To show the trade-ins that appear in this type of emulation we shall use an example system that is quite typical of what we can find in an industrial controller. The actual system we consider is a Hitachi A4010S Scara Robot. This is an industrial assembly robot which uses an 8 MHz MC68000 microprocessor in its control unit. The emulation requirements are very similar to those found in many other M68000 embedded systems.

The original objectives of this project were to increase the communication capabilities of the robot so that it could be integrated easily into a multiple robot system [6] and to experiment with adaptive controllers. Neither of these goals could be met with the original microprocessor and thus we had three different choices:

- build a custom robot control unit;
- use a commercial "open" robot control unit like those available from Adept; or
- emulate the M68000 processor bus.

The first solution is the most expensive, both in terms of labour and final cost. The second is very attractive but, in our case (and this is not infrequent for sophisticated industrial controllers), we could not find a commercial system with the required specs. Thus bus emulation became a natural choice.

An important problem is to select a microprocessor that is adequate to meet the objectives of the substitution project. In our example we are fortunate because there have been several studies (e.g., [2,7]) of processor performance requirements for robot applications. On the other hand, performance is not the only requirement that must be met by the substitute system: the difficulty of implementation and the availability of development tools are equally as important.

Thus we consider a substitute system which meets the performance requirements, is easy to design with and has good development tool support. This is the Pentium based PC-compatible computer. The Pentium microprocessor has a much higher process speed than the MC68000 microprocessor for several reasons. The most important are the internal superscalar architecture, greater bus bandwidth (64 bits), the higher clock rate (over 200 MHz) and the inclusion of the maths coprocessor and memory cache on the same chip. Of course, the same interface circuit is valid for a Pentium Pro (or any other processor) based PC if an ISA bus is available. In the rest of this paper we shall refer to the Pentium, as this is the processor we actually used in our system.

Because the substitute processor is not software-compatible with the original one this means that the control software must be rewritten. In our example, Control and Trajectory Generation Algorithms were rewritten for the PC, which can access the devices in the Robot Control Unit memory map by means of the interface device.

The Hitachi A4010S Robot Controller, like many other MC68000 based embedded systems, uses an address space of only 64 kB, where we can find the system memory and the I/O devices. In the PC real address mode it is not easy to find enough space to easily include even this reduced 64 kB target address space, and therefore it has to be included in the Extended Memory address space, which cannot be accessed easily under DOS.

Therefore, the system software is easier to implement if we design using a state machine that generates MC68000 bus cycles. Thus, we have an ISA bus expansion card programmable by means of a small number of I/O ports, which are available in any PC. From the point of view of the substitute microprocessor, the interface device is simply a peripheral.

4.1. ISA and MC68000 buses

In this section we shall examine the ISA and the M68000 buses by considering only those signals that are usually required to implement processor substitution in the most usual embedded controllers. The most common case in industrial M68000 based systems are boards with reduced address spaces (usually 64k) and self-vectored interrupts.

Thus, from all the ISA bus signals [8], we only use those that are necessary in order to design the interface:

1. Signals to control user registers, which are used to program the interface. These are: Data Bus (D0...D15), Address Bus (A0...A9) and those to allow 8 and 16 bit read/write accesses (see Fig. 1).
2. Clock signal CLK (8 MHz) to synchronize state machines and RESET DRV signal to initialize them.
3. The IRQ lines, through which the interface device can communicate with the PC. These allow the PC to serve interrupts requested by the target device.

From the MC68000 bus [9] only those aspects most usually found in embedded controllers will be emulated. These are:

1. Asynchronous bus cycle: used to access memory, M68000 peripherals and fast custom I/O devices.
2. Synchronous bus cycle: used by M68XX peripherals and some synchronous custom I/O.
3. Autovectored interrupts: used by M68XX and custom I/O.

In this example vectored interrupts are not supported because they are not present in many embedded systems. However, we shall discuss a simple extension to support this type of interrupts.

Provided that the target system address space size is 64 kB, only lines A1...A15 will be considered (line A0

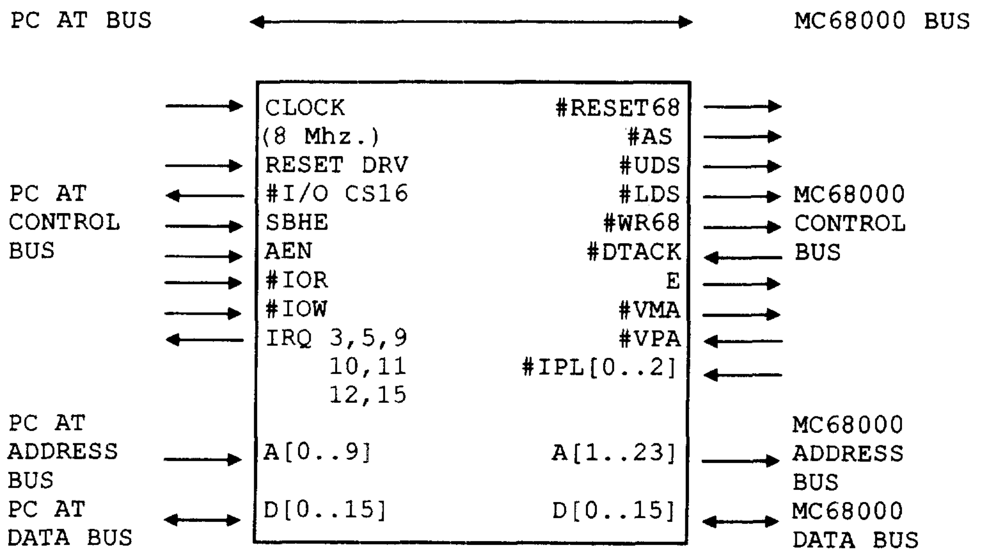

Fig. 1. Bus PC AT with bus MC68000 interfacing.

exists for software only). Likewise, the data bus size will be 16 bits (D0...D15) in order to allow 8 and 16 bit transfers.

Fig. 1 shows a general scheme of the interface device and Fig. 2 shows a functional diagram.

4.2. User registers

As we have indicated in previous sections, the interface device consists of an ISA bus expansion card which is programmable by means of a reduced set of I/O ports [10]. Therefore, it is a device with a peripheral configuration where the user registers are mapped into these I/O ports.

In order to carry out a MC68000 cycle bus, the user must write an access address into the Address Register. If it is a write cycle, previously the user would have to put data into Data Registers. Finally, the transfer direction (read or write) and size (8 or 16 bits) is written into the Status Register to start a bus cycle.

When the Status Register indicates the end of the current bus cycle, the interface device will be able to carry out a new one. If it was a read operation, the PC will capture read data accessing the Data Registers.

If a target system device requests an interrupt, the PC will be able to obtain its priority level by accessing the Interrupt Control Register. Questions about interrupt handling will be treated in the corresponding section.

Finally, a write access to P6 I/O position resets interface

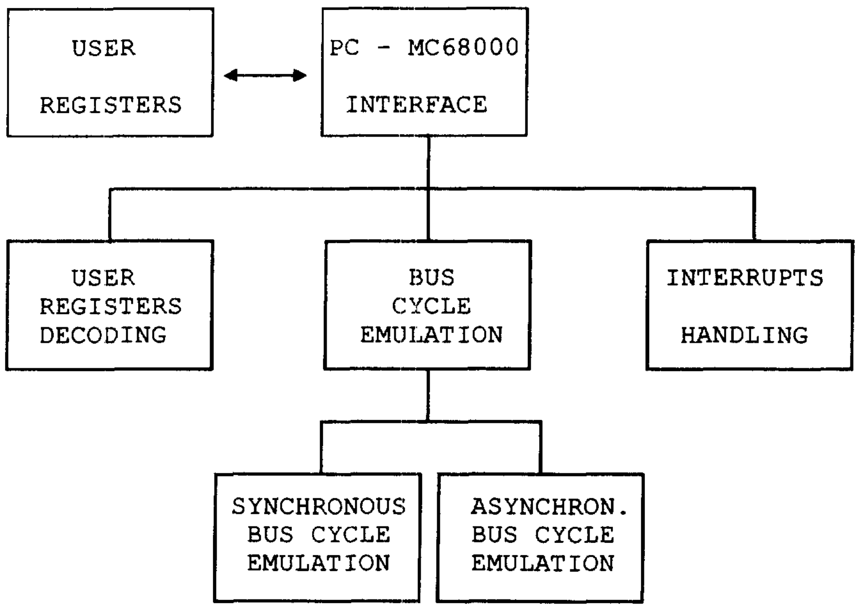

Fig. 2. Interface device functional diagram.

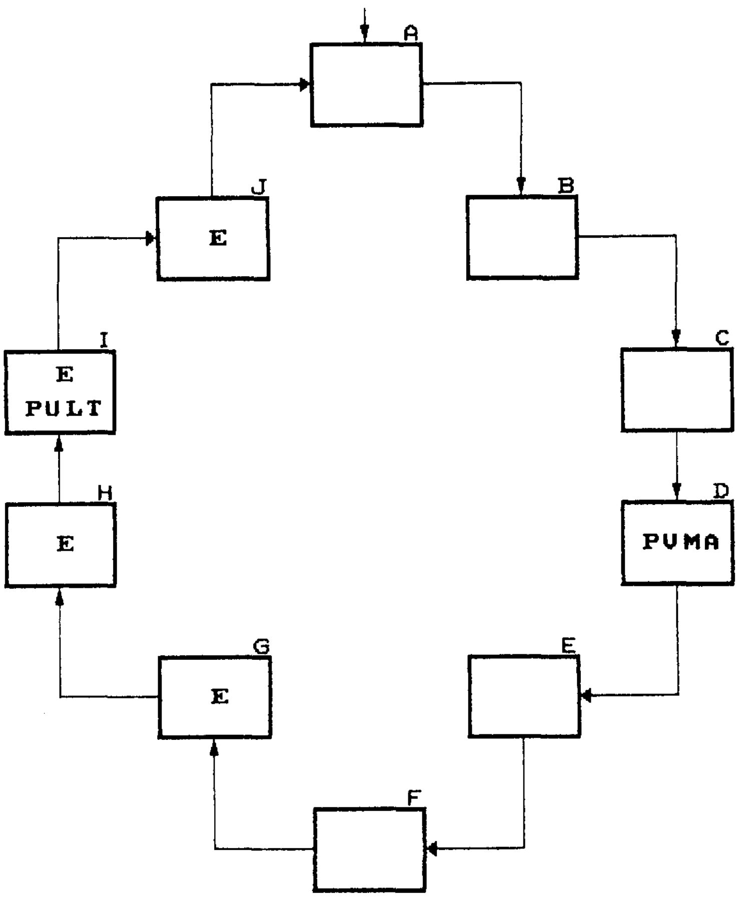

Fig. 3. State machine A.

device State Machines and activates a #RESET signal. This initializes the target system.

P0	...	ADDRESS REGISTER A[0...15]	(W)
P2	...	DATA REGISTERS D[0...15]	(R)
P2	...	DATA REGISTER D[0...7]	(R)
P3	...	DATA REGISTER D[8...15]	(R)
P4	...	STATUS REGISTER	(R, W)
P5	...	INTERRUPT CONTROL REGISTER	(R, W)
P6	...	CLEAR PREVIOUS IRQ	(R)
		#RESET SIGNAL ACTIVATION	(W)

where "R" denotes a read access and "W" denotes a write access.

4.3. State machines

In order to carry out asynchronous and synchronous bus cycle emulation, two State Machines have been designed.

The first, called *state machine A* (Fig. 3), implements the E signal (Enable). It is simply a 10-state ring counter: in the first six states the E signal is set to zero (low), whereas in the last four states it is set to one (high).

The second, called *state machine B* (Fig. 4), is more complex and carries out the proper asynchronous or synchronous bus cycle emulation. In order to execute the synchronous bus cycle, synchronization with state machine A becomes necessary since M6800 family peripherals use the E signal as a clock to carry out their bus cycles.

Motorola's M6800 peripherals [9] are directly compatible with the MC68000 microprocessor. In order to interface the synchronous M6800 peripherals with the asynchronous MC68000, the original microprocessor modifies its bus cycle to meet the M6800 cycle requirements whenever a M6800 device address is detected. This is possible since both microprocessors use memory-mapped I/O.

When an M6800 device is being accessed, the decode circuit answers activating a #VPA (Valid Peripheral Address) signal instead of a #DTACK signal. Thus, when this happens, state machine B executes the Synchronous Cycle, in which evolution of an E signal must be considered. First, it is necessary to wait until the E signal reaches the fifth subcycle (AT PC clock cycle) of its low subperiod, because then state machine B activates the #VMA signal. Finally, the latter waits until the last (fourth) subcycle of the E signal because a high subperiod occurs, where synchronous bus cycle ends capturing data in the case of a read access.

4.4. Interrupt processing

MC68000 microprocessor can serve two different types of interrupt: vectored and autovectored [9]. In the first case,

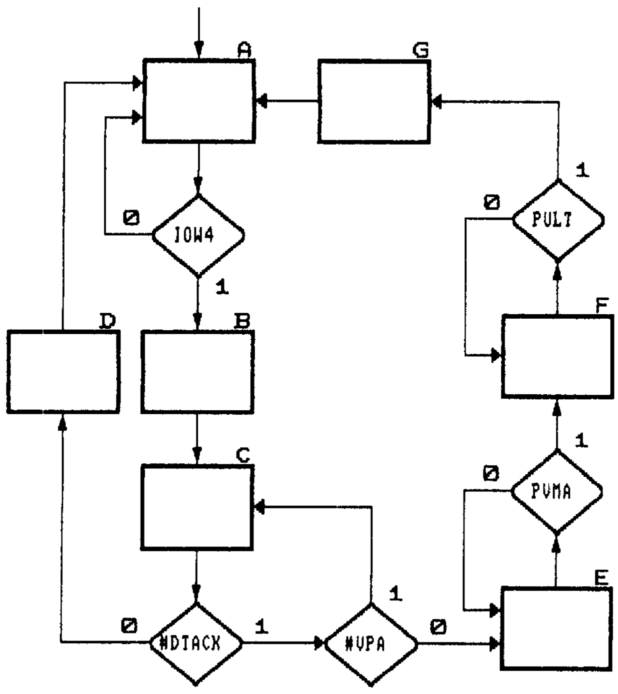

Fig. 4. State machine B.

the microprocessor fetches the interrupt vector number from the interrupting device. In autovectored interrupts, the MC68000 internally generates a vector number which is determined by the interrupt priority level number. MC6800 family devices can only request autovectored interrupts.

In our example target system, as in many others, all the interrupts — periodic, system error and serial transmission (MC6850 ACIA) — are autovectored.

In our design the Interrupt Control Register supports read and write operations. When a target system device requests an interrupt, it sends its priority level through lines #IPL0...#IPL2. This interrupts the PC and so the corresponding ISR should perform a read access to the Interrupt Control Register in order to obtain the Target interrupt priority level, and execute the corresponding service routine.

Furthermore, at any time during its execution, the control program can write a new value for the interrupt mask, which consists of a 3 bit number. As Eqs. (5)–(8) indicate, only those interrupts whose priority level is greater than the interrupt mask value will be considered by the interface device, which corresponds to the behaviour of a real M68000 based system. Note that interrupt level seven is always served regardless of the interrupt mask value. Therefore, it plays

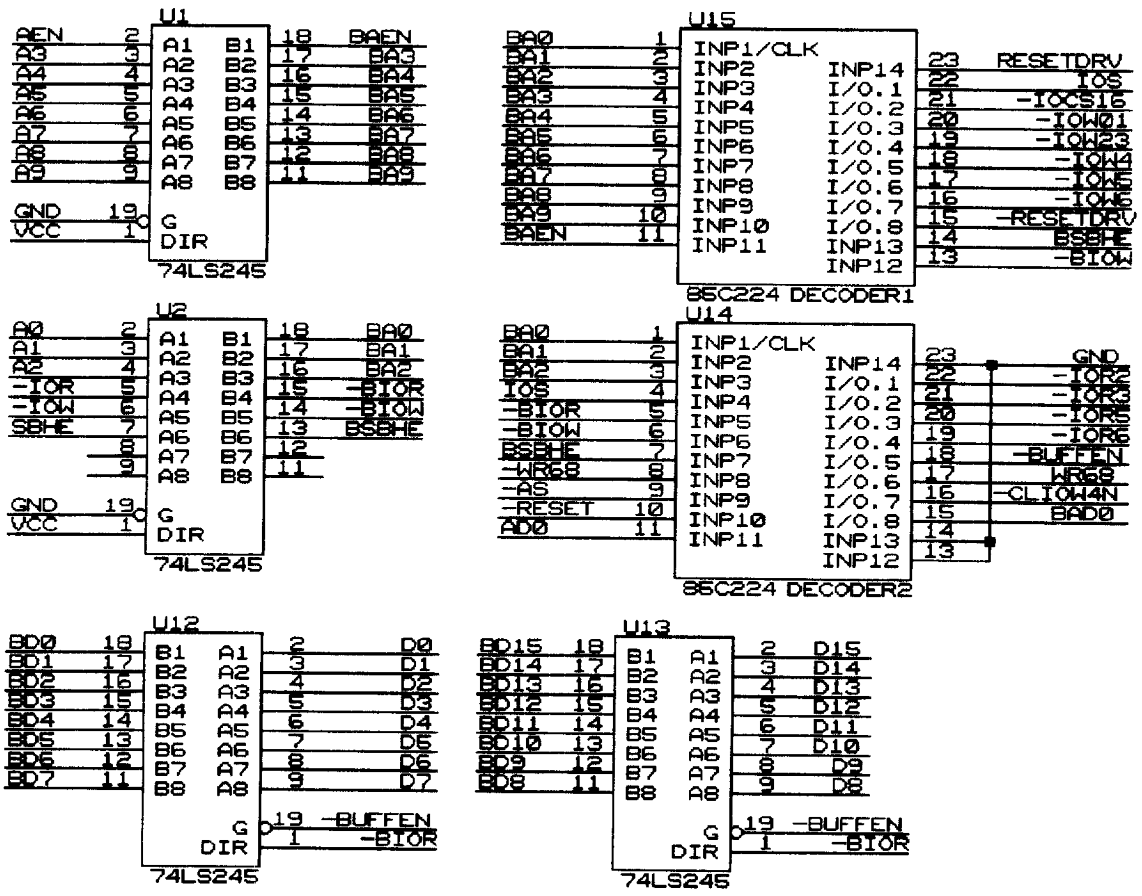

Fig. 5. Decodification of I/O ports (PC AT bus).

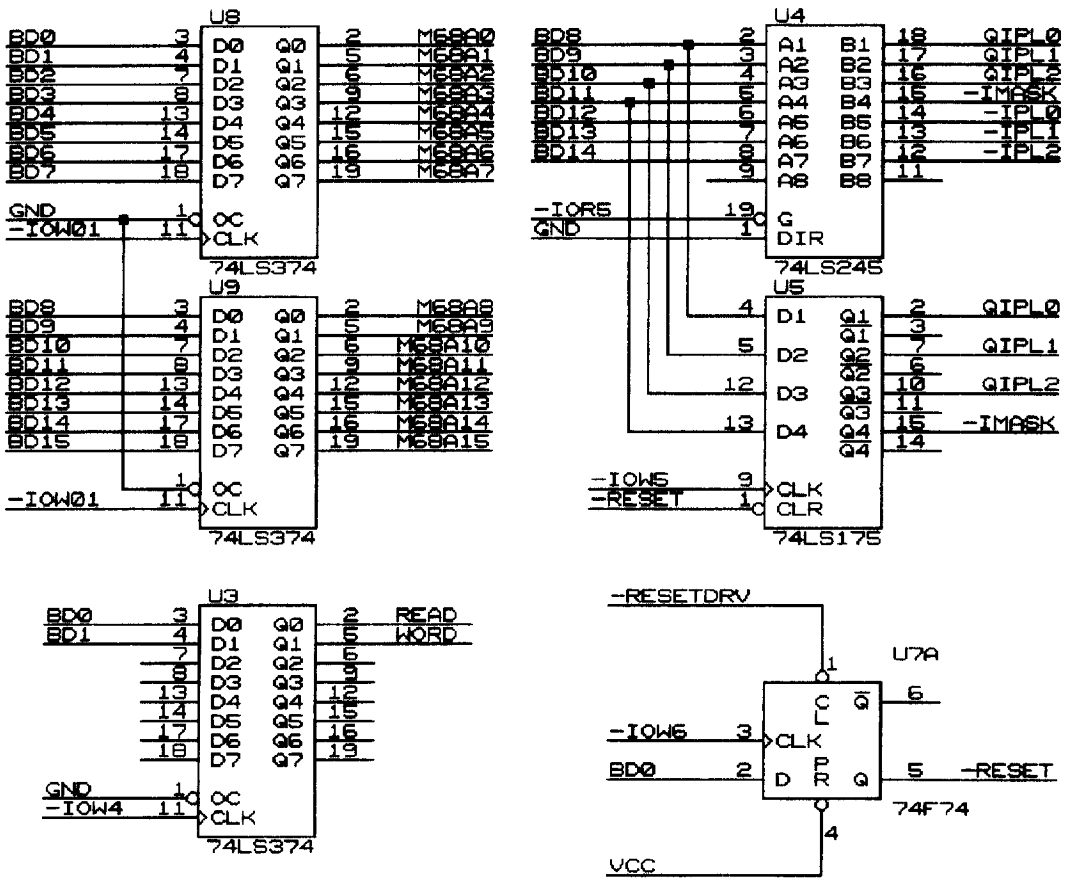

Fig. 6. User registers.

the role of Non Maskable Interrupt (NMI) from the interface point of view. Of course, this interrupt can always be masked by the interrupt controller in the PC chip set.

$$IPL0 := \text{NOT } IPL0 \qquad (5)$$

$$IPL1 := \text{NOT } IPL1 \qquad (6)$$

$$IPL2 := \text{NOT } IPL2 \qquad (7)$$

$$IRQ := (IPL0,\ IPL1,\ IPL2) > (QIPL0,\ QIPL1,\ QIPL2)$$
$$\text{OR } (IPL0,\ IPL1,\ IPL2) = (1,\ 1,\ 1) \qquad (8)$$

where the notation "(a, b, c)" means that bits a, b and c make up a 3 bit binary number. QIPL0, QIPL1 and QIPL2 indicate the interrupt mask value, which is stored in the Interrupt Control Register.

4.5. Physical implementation of interface device

The physical implementation of the interface device can be done in many different ways. The choice will depend mostly on the expected demand on the product. In the case of industrial controllers this demand is usually very low. In many cases we may need to update a few systems and, in such cases, the usual alternative is to use a mix of commercial fixed function and programmable devices.

The whole interface logic can be implemented in a Xilinx XC7354 CPLD, an AMD MATCH 130 or four generic 22V10. Originally Intel 85C224 PLD were used but currently we are using the commonplace 22V10. Any other type of flexible PLD can also be used for this design.

User register decoder (Figs. 5 and 6) has been implemented by using two 22V10 devices. Two further devices were used for the two State Machines (Fig. 7). In this case, for reasons of circuit optimization, one PLD contains the synchronous sequential circuits, whereas the other one has been programmed with combinatory functions, which depend directly on the states, generated by the first PLD.

Data registers (Fig. 7) are two 74AS652 transceivers. They are very fast devices, which guarantees that read and write data operations performed by the interface device during MC68000 bus cycles meet the allowed maximum response times.

In order to synchronize the signals that enter the State Machines which perform bus emulation (#DTACK and #VPA mainly), a 74F175 IC is used. A 74F74 IC

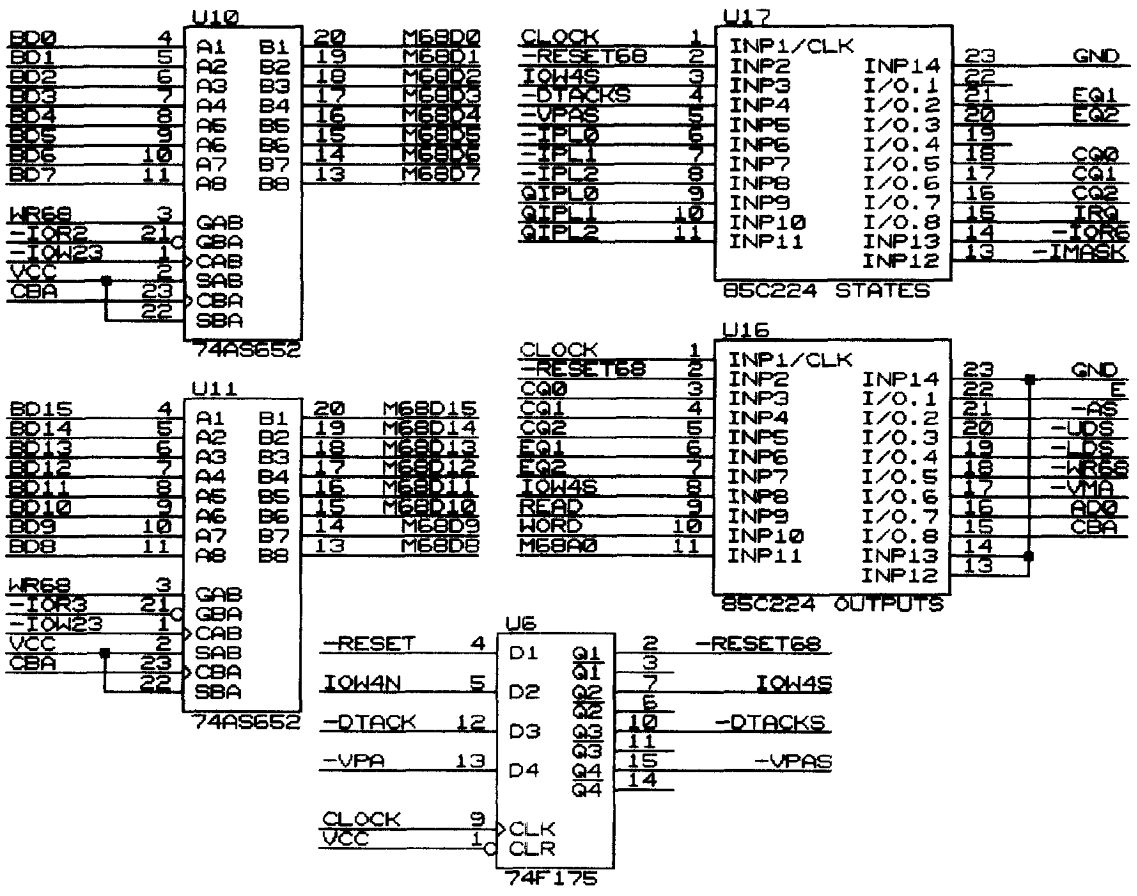

Fig. 7. Data registers and bus MC68000 emulation (Sequential Synchronous Circuit).

(containing two D flip-flops) is used for other functions. The remaining ICs, whose response time requirements are not so severe, belong to S and LS subfamilies.

On the expansion card there are two connectors from which cables transmit emulated MC68000 bus signals to the socket where the original MC68000 microprocessor was inserted. Use of this technique requires no modification of the rest of the target system.

4.6. Target system control software

Once that correct operation of the target system has been checked with adequate software routines, we must write the control SW for the host processor. In the case of our example a program has been developed in order to evaluate the robot's performance.

The program is written in C and runs under MS DOS with uC/OS Real Time Kernel [11]. The uC/OS is a very small, portable, pre-emptive kernel that eases the design of real-time control systems in PC and several other environments. Thus it is a good choice when using the PC as a replacement processor in outdated control systems. This kernel incorporates:

- task creation, deletion, and priority change;
- Mailbox and Queue task communication; and
- semaphores for shared resource control.

There are many other real-time kernels for the PC or even full featured RT/OSs. We chose uC/OS mainly because it fits our requirements and several versions are freely available on the Internet.

Our Robot Control Unit example, in addition to communicating with the operator, performs two important tasks: Trajectory Generation (high-level low-priority task) and Position/Speed Control (low-level high-priority task).

5. Conclusions

1. We have established a set of scenarios where bus emulation is an attractive design alternative. The performance penalty incurred by this type of emulation has also been analysed.
2. Bus emulation techniques have been applied to obtain an Open Control Unit based on a Pentium PC for a fairly standard, M68000 based embedded controller. For this purpose, an interface device that emulates original system bus cycles from the substitute system bus (ISA) has been developed by using a State Machine approach.

3. We can consider that a double bus emulation is present in this type of system. First, PC motherboard circuits emulate the ISA bus with a Pentium microprocessor local bus. Then, an expansion card emulates the MC68000 bus.
4. The use of real-time kernels that work on PC-compatible hardware has been proposed as a very useful tool when converting old controllers to PC architecture.
5. As a good example of our approach we have converted an old Hitachi A4010S Scara Robot into an open system that can be used now as a platform to test new control and trajectory generation algorithms. Programs developed with this purpose will be able to use all PC resources, as well as its processor speed and memory capacity.

Acknowledgements

This work has been supported by CICYT of the Spanish Government under project TAP93-0443.

References

[1] G. Jiménez, J.L. Sevillano, A. Civit Balcells, E. Díaz, A. Civit Breu, Application to embedded controllers. A general study of bus emulation. In: Proceedings of the Twelfth IASTED International Conference Applied Informatics, France, 18–20 May 1994, IASTED, Canada.
[2] G. Jiménez, Diseño y evaluación experimental de una plataforma para la investigación en robótica basada en procesadores Risc (in Spanish). Ph.D. Thesis, Universidad de Sevilla, November 1992.
[3] Digital Equipment Corporation, Alpha AXP Systems Handbook, 1993, Maynard, Massachusetts, USA.
[4] J.L. Patterson, H. Hennesy, Computer Architecture: A Qualitative Approach, 2nd ed., Morgan Kaufmann Publishers Inc., San Francisco, California, USA, 1996.
[5] Intel Corporation, i960 RP Processor: A Single-Chip Intelligent I/O Subsystem. Technical Brief, Mt. Prospect, IL, USA, 1995.
[6] G. Jiménez, J.L. Sevillano, A. Civit Balcells, F. Díaz, A. Civit Breu, Risc-based architecture for multiple robot systems, Microprocessor and Microsystems 16 (4) (1992) 177–186.
[7] S. Casell, E. Faldella, F. Zanichelli, Performance evaluation of processor architectures for robotics. Proc. of IEEE COMPEURO'91: Advanced Computer Technology, Reliable Systems And Applications, Bologna, 1991.
[8] L.C. Eggebrecht, in: SAMS (Ed.), 11711 North College, Carmel, Indiana, USA, Interfacing to the IBM Personal Computer, 2nd ed., 1990.
[9] Motorola Semiconductor Products Inc., MC68000 16-bit Microprocessor, 1983.
[10] 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.
[11] J. Labrosse, μC/OS: The Real Time Kernel, R&D Publications, Lawrence, KA, USA, 1992.

Antón Civit Balcells received his Master degree in Physics (electronics) and his Ph.D. from the University of Sevilla, Spain in 1984 and 1987 respectively. After working for several months with Hewlett–Packard he joined the University of Seville where he is currently "Profesor Titular" of Computer Architecture. He is the author of various papers and research reports on computer architecture, rehabilitation technology and robotics. He is the director of the "Robotics and Computer Technology for Rehabilitation" group in the University of Seville. His research interests include advanced wheelchairs, robotics and real-time architectures.

José María Rodríguez received his Master degree in Computer Science from the University of Seville in 1993. From November 1993 to June 1995 he worked on robot control with the "Robotics and Computer Technology for Rehabilitation" Group at the University of Seville. He is currently Associate Professor at the University of Cádiz, and his research interests include multiple robot coordination and control.

Gabriel Jiménez received his Master degree in Physics (electronics) and his Ph.D. from the University of Seville, Spain. After working with Alcatel he was granted a fellowship from the Spanish Science and Technology Commission (CICYT). Currently he is "Profesor Titular" of Computer Architecture at the University of Seville. He is author of various papers and research reports on robotics, rehabilitation technology and computer architecture. He is also Assistant Editor for the Journal of Computer and Software Engineering.

José Luis Sevillano received his Master degree in Physics (electronics) and his Ph.D. from the University of Seville, Spain, in 1990 and 1993 respectively. From 1989 to 1991 he was a researcher supported by the Spanish Science and Technology Commission (CICYT). After being Assistant Professor of Computer Architecture at the University of Seville, he is currently "Profesor Titular" at the same University. He is author of various papers and research reports on robotics, real-time computer communications and advanced wheelchairs.

Fernando Díaz received his degree in Electronic Physics from the University of Seville in 1990. From July 1990 to April 1991 he was with Abengoa-Control Data developing a new control system in electrical supply systems. After working as Associate Professor at the University of Huelva from 1995 to 1996, he currently holds the same position at the University of Seville. His research interests include mobile robots, multiple robot coordination and control. He is expected to obtain his Ph.D. in July 1997.